ISSN 1213-6972 Volume 13, Number 1-3, 2005

Journal of WSCG

An international journal of algorithms, data structures and techniques for computer graphics and visualization, surface meshing and modeling, global illumination, computer vision, image processing and pattern recognition, computational geometry, human interaction and virtual reality, animation, multimedia systems and applications in parallel, distributed and mobile environment.

EDITOR – IN - CHIEF

Václav Skala
University of West Bohemia

Journal of WSCG

Editor-in-Chief:	Vaclav Skala
	University of West Bohemia, Univerzitni 8, Box 314
	306 14 Plzen
	Czech Republic
	skala@kiv.zcu.cz
Managing Editor:	Vaclav Skala

Author Service Department & Distribution:

Vaclav Skala UNION Agency
Na Mazinach 9
322 00 Plzen
Czech Republic

Hardcopy:	***ISSN 1213 – 6972***
CD ROM:	***ISSN 1213 – 6980***
On-line:	***ISSN 1213 – 6964***

WSCG 2005

International Programme Committee

WSCG 2005 Board of Reviewers

Leopoldseder,S. (Austria)
Lewis,J. (United States)
Lintu,A. (Germany)
Loizides,A. (Cyprus)
Loizides,A. (Cyprus)
Magnor,M. (Germany)
Maierhofer,S. (Austria)
Mandl,T. (Germany)
Mantler,S. (Austria)
Margala,M. (United States)
Marinov,M. (Germany)
Maughan,C. (USA)
McAllister,D. (USA)
McMenemy,K. (United Kingdom)
Mertens,T. (Belgium)
Moccozet,L. (Switzerland)
Mokhtari,M. (Canada)
Moltedo,L. (Italy)
Montrucchio,B. (Italy)
Moreton,H. (USA)
Mudur,S. (Canada)
Mueller,K. (United States)
Muller,H. (Germany)
Myszkowski,K. (Germany)
Neubauer,A. (Austria)
Nielsen,F. (Japan)
O'Sullivan,C. (Ireland)
Ozguc,B. (Turkey)
Pan,Z. (China)
Pandzic,I. (Croatia)
Pasko,A. (Japan)
Pedrini,H. (Brazil)
Perez,M. (Spain)
Peroche,B. (France)
Plemenos,D. (France)
Post,F. (Netherlands)
Prakash,E. (Singapore)
Pratikakis,I. (Greece)
Prikryl,J. (Czech Republic)
Puppo,E. (Italy)
Purgathofer,W. (Austria)
Rauterberg,M. (Netherlands)
Ravyse,I. (Belgium)
Renaud,c. (France)
Revelles,J. (Spain)
Rheingans,P. (United States)
Rodrigues,M. (United Kingdom)
Rokita,P. (Poland)
Rossignac,J. (United States)
Rudomin,I. (Mexico)
Sahli,H. (Belgium)
Sainz,M. (USA)
Sbert,M. (Spain)
Segura,R. (Spain)
Shamir,A. (Israel)
Schaller,N. (United States)
Schneider,B. (United States)
Scholz,V. (Germany)
Schumann,H. (Germany)
Sijbers,J. (Belgium)
Sips,M. (Germany)
Sirakov,N. (United States)
Sitte,R. (Australia)
Slusallek,P. (Germany)
Snoeyink,J. (United States)
Sochor,J. (Czech Republic)
Sorel,M. (Czech Republic)
Sroubek,F. (Czech Republic)
Stuerzlinger,W. (Canada)
Stylianou,G. (Cyprus)
Suarez Rivero,J. (Spain)
Sumanta,P. (United States)
Szekely,G. (Switzerland)
Szirmay-Kalos,L. (Hungary)
Tang,W. (United Kingdom)
Taubin,G. (United States)
Teschner,M. (Germany)
Theobald,C. (Germany)
Theoharis,T. (Greece)
Theußl,T. (Austria)
Tobler,R. (Austria)
Torres,J. (Spain)
Trahanias,P. (Greece)
Traxler,A. (Austria)
Van Laerhoven,T. (Belgium)
Velho,L. (Brazil)
Veltkamp,R. (Netherlands)
Vergeest,J. (Netherlands)
Vuorimaa,P. (Finland)
Weiskopf,D. (Germany)
Weiss,G. (Germany)
Westermann,R. (Germany)
Wu,S. (Brazil)
Wuethrich,C. (Germany)
Yilmaz,T. (Turkey)
Zach,C. (Austria)
Zachmann,G. (Germany)
Zara,J. (Czech Republic)
Zemcik,P. (Czech Republic)
Zhu,Y. (United States)
Zitova,B. (Czech Republic)

Journal of WSCG

Vol.13, No.1-3, 2005

Contents

Label Layout for Interactive 3D Illustrations

Kamran Ali, Knut Hartmann, and Thomas Strothotte
Department of Simulation and Graphics
Otto-von-Guericke University of Magdeburg
Universitätsplatz 2, D-39106 Magdeburg / Germany
{kamran, knut, tstr}@isg.cs.uni-magdeburg.de

ABSTRACT

Hand-made illustrations in scientific and technical textbooks commonly use internal and external labels or legends to establish co-referential relation between pictorial elements and textual expressions. By analyzing the most complex examples, we extracted several label layout styles and classified them. We propose a variety of real-time label layout algorithms that aim to produce nice and clean layouts. In order to achieve a frame-coherent label layout during user interactions, the algorithms consider layout decisions from previous frame. Moreover, several evaluation criteria to measure the quality of static as well as dynamic label layouts are presented.

Keywords

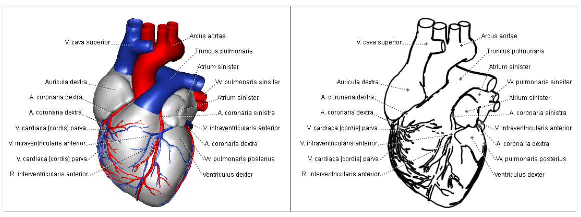
Label-Layout, External Labeling, Text-Image Integration, Multi-Modal Presentations

1 INTRODUCTION

Interactive tutoring systems aim at presenting information in the most effective way. The dual coding theory [CP86] suggests that humans posses two independent processing systems—one for visual and the other for verbal elements. Hence, using two channels, more material can be conveyed, but their content has to be integrated mentally.

Human illustrators employ a number of techniques to establish *co-referential* relation between visual and verbal elements. Labels, legends, and figure captions provide denotations, technical terms, and descriptions for visual elements. However, their automated integration within an interactive 3D environment remains a big challenge.

Text *labels* either overlay visual objects or placed outside (*internal* vs. *external* labels). *Connecting lines* reveal co-referring external labels and visual objects, whereas *anchor points* ease the identification of visual objects. In this work, the term *label layout* refers to the determination of the positions of anchor points and external labels, which are linked with connecting lines



using a specific line style. Moreover, the term *graphical model* refers to a complex visual object with separate individual visual objects.

We extracted requirements for several layout styles which are prevalent in hand-made illustrations and present techniques towards an automated generation of label layouts in real-time. In order to achieve a frame-coherent label layout during user interactions, these algorithms consider layout decisions from the previous frame. Moreover, the system facilitates automatically generated *legends* for graphical models and *textual explanations* for visual objects. The mental integration of information presented in 3D and legend viewer is aided through a synchronized object selection and highlighting mechanism.

The paper starts by giving a review of related work in Section 2. Section 3 states the requirements for dynamic labeling system. In Section 4 several label layout styles are classified. Section 5 presents the architecture of our label layout system and provides the algorithms to generate several layouts. Moreover, coherency aspects and the application of labels in legends are described. Section 6 states the *evaluation criteria* to measure the quality of layouts. Finally, Section 7 discusses directions of future research.

The Journal of WSCG, Vol. 13, ISSN 1213-6964
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency–Science Press

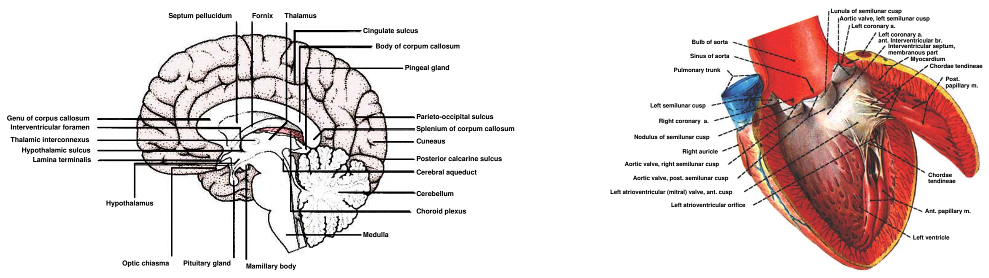


Figure 1: Variety in the layout styles (Source: [Rog92, p. 317] and [SPP97, p. 81]).

2 RELATED WORK

The label layout problem has received much attention in non-interactive cartographic applications [CMS95] where the labels have to be placed for point, line, and area features. However, the label placement is independent of the shape of graphical features and can be unified for all kinds of features [KT98]. Finding the optimal solution of the labeling problem (i.e., without overlapping labels) is proven to be NP-hard [MS91]. Therefore, several approximation methods have been developed to reduce the computational complexity.
A number of interactive multi-modal systems integrate external labels into the visualization of geometric objects. But most of them rely on fixed regions for visual and textual elements (e.g., [PRS97]) or a manual label layout (e.g., [RSHS03]). Only a few solutions have been proposed to integrate internal and/or external labels into interactive 3D applications (e.g., [BFH01]), but they lack the ability to generate a variety of layouts which are often seen in hand-made illustrations.

3 REQUIREMENTS

In dynamic environments an effective layout must fulfill a number of requirements ([Imh75, FP99]):

Readability: Labels must not overlap,
Unambiguity: Labels clearly refer to their objects,
Pleasing: Prevent visual clutter,
Real-Time: Compute layouts at interactive rates,
Frame-Coherency: Prevent visual discontinuities,
Compaction: Reduce the layout area.

These requirements may conflict with each other and with another demand: *label as many visual objects as possible.* Some of these requirements can be evaluated easily, whereas the extraction of criteria for the second and third aspect is less obvious. We use several heuristics to achieve a pleasant and unambiguous layout:

(i) Place anchor points over salient positions,
(ii) Place labels near to their corresponding objects,
(iii) Align labels mutually and with respect to the graphical objects, and
(iv) Eliminate line crossings.

The label layout algorithms are incorporated into an interactive application where visual discontinuities between subsequent frames must be avoided. Moreover, the layouts should be as compact as possible to fit on the limited screen space. Finally, the layout algorithms have to cope with situations where some labels do not fit into the given screen space. As the computation of an optimal solution is NP-hard, several simple yet effective methods are proposed that can be carried out in real-time.

4 LAYOUT STYLES

The material in this section is based on a manual analysis of label layouts in hand-drawn illustrations. For this purpose, we chose anatomic atlases, anatomic textbooks, and visual dictionaries because of their making extensive use of external labels and due to the extraordinary quality of their label layouts.
The manual analysis reveals that human illustrators use a number of different label layout styles with style-specific illustration techniques and properties (see Figure 1). Therefore, we classified them according to their common properties (see Figure 2):

Straight-Line: Labels and anchor points are connected with straight lines (see Figure 1-Right).
Orthogonal: Connecting lines are axis-aligned and the bends are made at orthogonal angles (see Figure 1-Left).
Flush Layout: Labels are assigned to distinct spatial areas (see Figure 3-a):

- *Flush Left-Right:* Labels are placed on the left and/or right side of the graphical model.
- *Flush Top-Bottom:* Labels are placed on the top and/or bottom of the graphical model.

Circular Layout: Labels are aligned on the silhouette of the graphical model in a circular fashion (see Figure 3-b):

- *Ring*: Labels are placed at regular intervals on a ring which encircles the graphical model.
- *Radial*: Labels are placed in radial form with respect to a common origin.
- *Silhouette-Based*: Labels are placed near the silhouette of graphical model at positions closest to their anchor points.

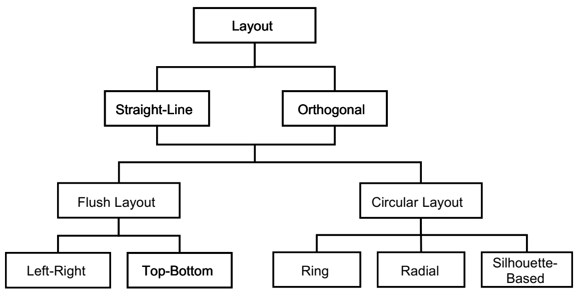


Figure 2: Layout Classification.

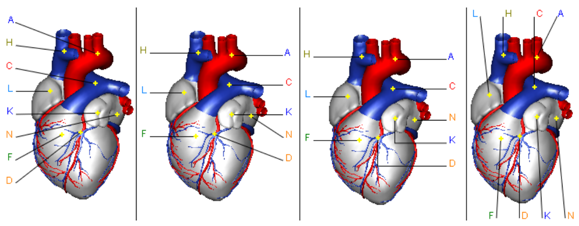
These styles are adopted in order to meet space requirements, to bring conformity in different illustrations, to maintain visual balance, and to ease reading. The most interesting observation is that there are some general but also several style-specific requirements.

5 AUTOMATED LABEL LAYOUT

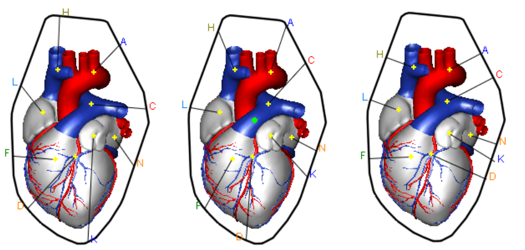
In this section, the properties and constraints of labels, anchor points, and connecting lines are defined. Moreover, we describe our approach towards the dynamic layout of label in interactive systems.

5.1 Object Properties and Constraints

The amount of text displayed in labels can range from one- or two-letter symbols to multi-line paragraphs. However, most often labels comprise few words on a single line. We classify labels into the following categories:



(a) *Flush* layouts combined with *straight-line* or *orthogonal* styles.



(b) *Ring*, *radial* and *silhouette-based* layouts.

Figure 3: Examples of various layout styles.

(i) *single-line* (max. 50 characters),
(ii) *multi-line* labels, or
(iii) *legend keys* (max. 2 characters).

For labels of different sizes, more constraints are needed to avoid label overlaps and line intersections. Thus, achieving a balanced layout becomes more problematic. Therefore, our layout strategy is restricted to single-line labels and legend keys which both have a fixed height and width. This constraint enables us to represent labels as *zero-sized points* and to maintain a minimal vertical and horizontal gap between them. Multi-line descriptions and legend text are provided on request. They do not alter their positions and have a semi-transparent background. User can pin them anywhere on the screen. The positions of all kinds of labels, anchor points, and connecting lines are specified in view-plane coordinates.
All objects are assigned *display priorities* (that consider projection size) and *user priorities*. For complex models, the labels can be filtered according to their *degree of interest*. If there is not enough place to display all label, objects with the smallest priorities are chosen and their labels are ignored.

5.2 System Architecture

Figure 4 presents an overview of our approach. The content presented in labels is provided by an external *domain expert*. The system works internally on 2D projection of 3D scene where individual visual objects are color-coded uniquely (*color-code image*). The rendered image is analyzed to determine visible objects and anchor points. Layout-specific algorithms determine initial positions for labels. Then label overlaps are eliminated and line intersections are resolved. If required, layout compaction is performed. Finally, the labels are rendered with chosen decoration style on top of the scene.

5.2.1 Domain Expert Initialization

The co-referential relation between textual annotations and visual objects is established by using an external knowledge base. When the system loads a 3D model, the domain expert defines a color-coding scheme for visual objects and provides textual descriptions.

5.2.2 Image Analysis

This module segments *color-code images*. For every rendered frame, it creates a list of all segments, their sizes, extents, and colors (to identify the visual objects). For each visible object, it determines one anchor point. Since an anchor point is intended to support the identification of visual object and its distinction from the remaining objects, its position is crucial to prevent co-referential mismatch. From observa-

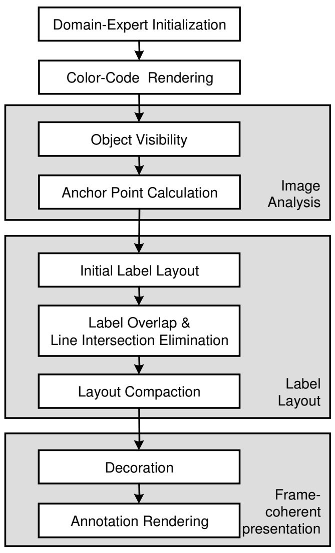


Figure 4: System architecture.

tions, we define the following heuristics to determine anchor points:

(a) They must overlay their corresponding objects.
(b) Place anchor points inside the biggest segments.
(c) Place them at the most internal locations of these segments.
(d) Avoid clusters of anchor points.

Anchor Point Calculation: If the objects have 'L' or 'U' shape, neither the center of the bounding box nor the centroid guarantee to fulfill the first condition. To compute the most internal pixel in a segment, we apply *distance function* on *color-coded images*. For every pixel this function computes its distance to the closest segment boundary and stores these values in a *distance image* (see Figure 5-Right).
There are several variants of this function which employ different metrics: *Euclidean*, *Manhattan*, and *Chessboard*.
The last two metrics are faster but less accurate. In order to reduce the computational expense of the euclidean metric, we adopt the *pseudo euclidean metric* d_{34} [AdB88][1]. The d_{34} metric assigns distance value 3 to horizontal and vertical neighboring pixels; the value 4 is assigned to diagonally connected pixels. We implemented a 2-pass algorithm to compute *distance image* [RP68] using the d_{34} metric. For each visual object a mask is placed over the *distance image* and the highest distance value is returned as anchor point.

Elimination of Anchor Point Clusters: In order to avoid referential mismatches or ambiguities, anchor points should not form clusters. Therefore, *repulsive*

[1]An approximation which purely uses integer operations and avoids square roots.

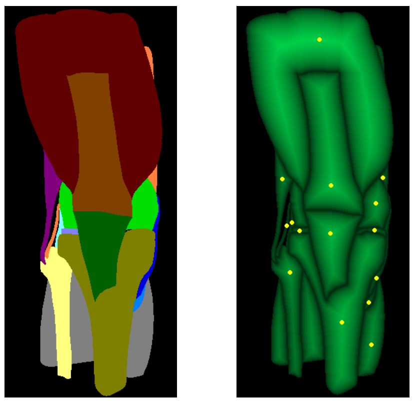


Figure 5: *Color-code image* (left) and *distance image* (right) with overlaid anchor points.

forces aim at separating anchor points by modifying the *distance image* D. An anchor point at position c adds a *subtractive function* which is centered at c and is applied to all pixels p of its influence region R:

$$D_p = D_p - f(||p-c||) * k \,;\, \text{for } p \in R$$

where f is a non-negative decreasing function, $||p-c||$ is the distance between p and c, and k is a scaling factor. The algorithm now determines the *distance image* D in a first phase. The second phase subsequently computes anchor points for visual objects by selecting the maximal distance values on their segments. After selecting each anchor point, the values in the *distance image* around the anchor are modified by the subtractive function.
In Figure 6, the green and red curves denote distances of two distinct visual objects to their segment boundaries. After placing an anchor point for the green object, the subtractive function (in blue dotted line) digs a valley into D and forms a new peaks for the red object (Peak 2 and 3). Finally, Peak 3 is selected as anchor position since it now has the highest quality.

5.2.3 *Label Layout*

All style-specific algorithms instantiate a generalized algorithm. In the following, we describe only the layout specific realizations of generalized tasks. All layout-specific algorithms strictly prevent label overlaps and intersections of connecting lines. Moreover, layouts can be made more compact. Our extensions to achieve a frame-coherent label layout are presented in the next section.

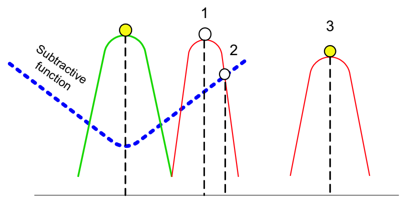


Figure 6: Separation of anchor points.

Generalized Algorithm: For a single frame,

1. Determine the positions of anchor points,
2. Determine the extents of empty space regions,
3. Allocate spatial regions for labels,
4. Compute an initial label layout,
5. Stack labels to eliminate overlap,
6. Resolve line intersections, and
7. Perform layout compaction.

All layout algorithms rely on the positions of anchor points. In Task 2, the extent and locations of the four biggest empty axis-aligned rectangles (left, right, top, bottom) around the graphical model are computed, as the *flush* layouts place labels solely in these regions. The Tasks 3, 4, and 7 are style specific.

Flush-Left-Right Layout: Modifies Tasks 3, 4, and 5.

Allocate spatial regions for labels (Task 3):

(a) Sort anchor points according to the x direction,
(b) Choose a *pivot* point (e.g., mean or median of anchor points), and
(c) Assign labels to the left and right region by comparing their anchors with the pivot point.

Compute an initial label layout (Task 4):

(a) Assign the y position of anchor points to y position of associated labels.
(b) Justify the labels on the bounding box of the graphical model.

Stack labels to eliminate overlap (Task 5):

(a) A recursive algorithm assigns new positions to the labels to eliminate label overlaps and minimize the average vertical length of connecting lines.

For *flush left* and *flush right* layout, we assign to the pivot element a minimal or maximal value. For *flush top* and *flush bottom* layout, the previous algorithm works by exchanging horizontal and vertical directions. Later on, in top and bottom regions, labels are stacked in vertical direction.

Radial Layout: Modifies Tasks 4 and 5.

Compute an initial label layout (Task 4):

(a) Select a center position o (e.g., mean or median of anchor points) and an appropriate radius r for a circle C which encloses the graphical model.
(b) Compute the radial projection of the anchor points on C.
(c) Align the corresponding label on this position. Labels on left-half of C are right-justified and labels on right-half of C are left-justified.

Stack labels up- or downwards to eliminate mutual label overlaps (Task 5).

The radial projection of anchor points produces no intersections of connecting lines. However, labels can overlap or lie very close to each other which is resolved by label stacking. An unbalanced distribution of labels might cause huge label stacks and increase the lengths of connecting lines. The *spring embedding* approach, which is described after the discussion of the individual layout styles, improves the layout considerably.

Ring Layout: Modifies only Task 4.

Compute an initial label layout (Task 4):

(a) Select a center position o (e.g., mean or median of anchor points) and an appropriate radius r for a circle C which encloses the graphical model.
(b) Choose n evenly spaced positions P on the circle.
(c) Determine a bijective mapping from the label set to P which minimizes the distance between labels and their anchor points.

Since the labels are already evenly spaced, there is no need to check for label overlaps for small n and big r.

Silhouette-Based Layout: Modifies Tasks 4 and 5.

Compute an initial label layout (Task 4):

(a) Compute the convex hull of the geometric model and enlarge it (pre-processing step)
(b) Project the anchor points on the edges of the convex hull silhouette boundary S. Choose the closest projection position to the anchor as label position.

Stack labels up- or downwards to eliminate mutual label overlaps (Task 5).

In our application the approximation of the silhouette boundary with convex hulls achieved a better quality compared to other bounding objects (e.g., circles or bounding boxes). But also this approach suffers from uneven label distribution. Again, the spring embedding approach is used to improve this layout.

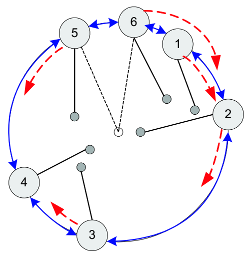
Spring Embedding Approach

To balance uneven label distribution in *radial* and *silhouette-based* layouts, we use a *force directed* approach [FR91] developed in graph drawing. We define a *repulsive force* between labels aiming to separate the labels, and an *attractive force* that moves the labels close to their anchor points. The configuration is done in a circular fashion. Hence, it is based on *angles* between the labels rather than on *distances*. For two labels v and u, Δ_r refers to an interior angle formed by them with respect to circle center o. The repulsive force f_r is inverse proportional to Δ_r:

$$f_r(\Delta_r) = -k^2/\Delta_r$$

where k is an ideal angle (e.g., 0.2 rad.) between v and u. Moreover, we establish an attractive force f_a between the labels v and associated anchors a_v. Let position p_v be the radial projection of a_v in radial layout. The position p_v on the circular ring attracts the label v. Δ_a refers to the interior angle between v and p_v with respect to o.

$$f_a(\Delta_a) = \Delta_a^2/k$$



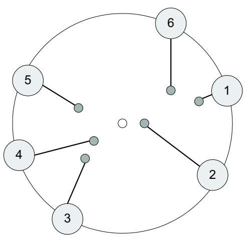


Figure 7: Using *spring embedding* to spread labels in the circle. Before (left) and after configuration (right).

where k is a tolerable angle (e.g., 0.2 rad.) between v and p_v. The algorithm configures the layout in many iterations. The amount of *temperature* t constrains the label displacement. The higher t, the bigger the displacement. The algorithm starts at high t and cools gradually. In each iteration, displacement angles for each label are computed by summing repulsive and attractive forces. After configuration, label overlaps are resolved. In our system, spring embedding approach can be performed in real-time as nearly 30 objects are labeled. We found 20 to 30 iterations enough to achieve an acceptable layout.
Figure 7 illustrates the spring embedding configuration. Filled enumerated circles represent labels which are arranged on a circle, whereas tiny filled circles represent anchor points. Solid blue lines indicate repulsive forces and dashed red lines indicate attractive forces between labels and their ideal positions.

5.2.4 *Line Intersection Elimination*

To resolve intersections of connecting lines, the restriction on *fixed size* labels is a big advantage. For any two intersecting lines, we can interchange their label positions without introducing new label overlaps:

```
Do until there are no intersections left
    if any two connecting lines intersect
        interchange their label positions
```

Orthogonal Layout
This style requires axis-aligned connecting lines with bend at orthogonal angles. It can be combined with all *flush* or *circular* layouts:

1. Compute label positions using any layout method,
2. Draw the connecting lines in orthogonal style.
3. Resolve intersections of orthogonal lines, and

The current implementation imposes two restrictions: (i) only one bend is allowed (i.e., connecting lines can employ a vertical and a horizontal segment) and (ii) vertical segments connect anchor points and bends, while horizontal segments connect bends and labels.
In order to detect line intersections in the orthogonal layout, each horizontal segment is tested for intersection with all vertical segments. Every time an intersection is found, the label positions are exchanged. This procedure is continued until no intersections remain.

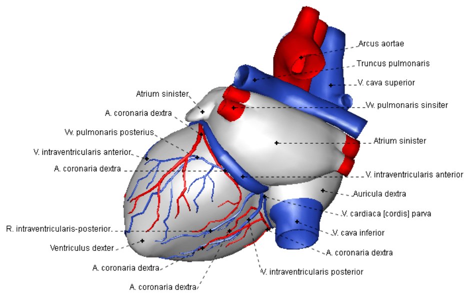


Figure 8: Layout Compaction.

5.2.5 *Layout Compaction*

In order to keep the connecting lines short, this step aims at moving the labels towards their anchor points. We implemented two methods based upon the kind of silhouette they use. The first method approximates the silhouette of graphical model by a convex hull. It computes intersections between connecting lines and the edges of the convex hull and re-targets the labels on the intersection points. Furthermore, label overlaps are resolved analogue to Task 5 in *silhouette-based* layout (see Figure 3-b).
The second compaction method uses the original silhouette boundary, and is preferred only for *flush left-right* layout as the height of labels is much smaller than the width. Each label in the left region is shifted horizontally towards the right until it hits some foreground pixel, or the horizontal distance between the label and the anchor point becomes zero. This test is performed using the *color-code image*. The labels are placed with some margin to the final position. Similarly, the labels in the right region are moved in. New line intersections may arise which are again resolved. After compaction, the labels in both sides closely follow the boundary of the model, and the layout looks more pleasing (see Figure 8).

5.2.6 *Frame Coherent Presentation*

Our discussion so far was restricted to static aspects. In interactive systems the label layout has to be recomputed after the user interacts. An independent layout for individual frames without considering continuity aspects results in *layout flickering*. Jumping labels and anchor points are both irritating and distracting. In order to achieve a *frame coherent* label layout, our algorithms are revised to consider the outcomes from previous frames.

Stabilizing Anchor Points: All of the layout algorithms rely heavily on the positions of anchor points. Therefore, movements of anchor points might induce a global change in the layout. To enhance the frame coherency, a new heuristic for placing anchor points

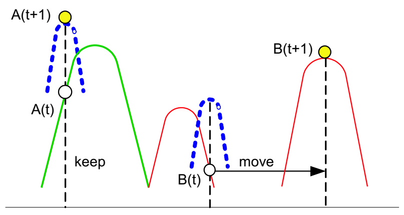


Figure 9: Stabilizing anchor points.

is added: *If possible, keep the anchor points at their previous locations.* However, anchor points must not leave the region of its corresponding visual object and might now reside at a very poor position. In both cases, the anchor points should shift.

In order to implement this new heuristics, we add an *attractive force* which aims at keeping anchor points close to their previous positions. For each anchor point at position c and its associated visual object O_i an *additive function* is applied on the *distance image D*. It affects the distance values for all pixels p of the corresponding object within an influence region R:

$$D_p = D_p + f(||p-c||) * k \quad ; \; p \in O_i$$

where f is a non-negative decreasing function, $||p-c||$ is the distance between p and c; and k is a scaling factor.

This function creates high peaks on previous anchor positions and increases their probability for being selected as new anchor points. In Figure 9, the $A(t)$ and $B(t)$ refer to the anchor points of the green and red object in frame t. In the next frame $t+1$ the additive function (in blue dotted line) modifies D and creates new peaks. The global maxima $A(t)$ and $A(t+1)$ for the green object are identical, so that its anchor point remains stable. However, there is now a new global maximum $B(t+1)$ for the red object, so that its anchor point moves to another location. This illustrates how we try to retain old positions as long as they are acceptable and jump to better candidates otherwise.

Stabilizing Label Layout: The assignment of labels to spatial regions is a very crucial for the appearance of a layout. In order to prevent frequent label jumps between different regions, the pivot point of flush layouts should also be stabilized. However, if it remains steady for a long time while the user interaction continues, the numbers of labels per region can get very unbalanced. To handle this problem, a pivot element is nailed as long as it provides an acceptable ratio of anchor points in two regions, otherwise it is set to new location.

Label Animation: In order to prevent visual discontinuities, changes of anchor points and label positions are animated. We prefer a *slow-in slow-out* interpolation. To avoid a distraction by floating labels within user interactions, layouts can be *frozen* until a new stable point-of-view is chosen.

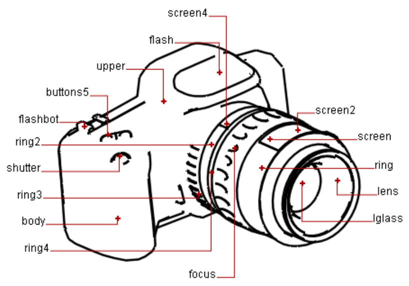


Figure 10: An *orthogonal layout* with NPR rendering.

5.2.7 *Decoration*

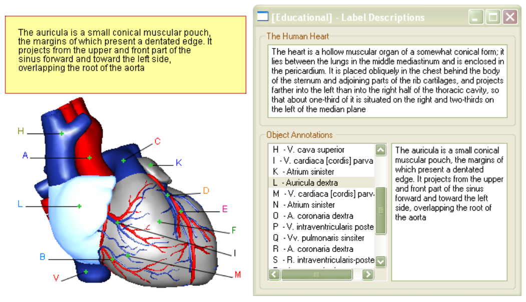
By default, we use white as background color and black as text color. For legends, different colors for indices help to find the associated elements. We suggest (i) to use dotted instead of solid lines (otherwise strips between lines show up), (ii) to use line shadows, and (iii) to decrease the color intensity of labels and connecting lines during animations. The system facilitates the user to change the color, size, and style for anchor points and connecting lines.

For creating abstract versions of illustrations, we integrated a non-photorealistic rendering system [HIR+03]. Figure 10 shows the results with orthogonal layout style. Long textual descriptions can be presented in both 3D and in a separate *legend viewer*. Object selection and highlighting is synchronized in both views (see Figure 11).

5.3 Selection of Layout Style

No layout is ideal under all circumstances, however, the knowledge of their specific advantages and constraints helps to select an appropriate one. The choice of a layout for external labeling depends largely on the shape and orientation of visual object, the spatial distribution of anchor points, the amount and distribution of free space available to insert labels, and personal preference for a particular layout.

Determining a suitable layout automatically can be very difficult. Therefore, layout selection is performed


The auricula is a small conical muscular pouch, the margins of which present a dentated edge. It projects from the upper and front part of the sinus forward and toward the left side, overlapping the root of the aorta

[Educational] - Label Descriptions

The Human Heart

The heart is a hollow muscular organ of a somewhat conical form; it lies between the lungs in the middle mediastinum and is enclosed in the pericardium. It is placed obliquely in the chest behind the body of the sternum and adjoining parts of the rib cartilages, and projects farther into the left than into the right half of the thoracic cavity, so that about one-third of it is situated on the right and two-thirds on the left of the median plane

Object Annotations

H - V. cava superior
I - V. cardiaca [cordis] parva
K - Atrium sinister
L - Auricula dextra
M - V. cardiaca [cordis] parv
N - Atrium sinister
O - A. coronaria dextra
P - V. intraventricularis poste
Q - Vv. pulmonaris sinsiter
R - A. coronaria dextra
S - R. intraventricularis-poste

The auricula is a small conical muscular pouch, the margins of which present a dentated edge. It projects from the upper and front part of the sinus forward and toward the left side, overlapping the root of the aorta

Figure 11: Synchronized legend and 3D viewer.

by the user via real-time previews. It also gives the user more freedom in choosing from a variety of available layouts if one layout does not look promising.

6 EVALUATION

An automatic evaluation of the label layouts can be made on the basis of the following parameters (mainly taken from the field of graph drawing [DBET+99]):

- Number of unlabeled visual objects,
- Number of line intersections,
- Number of label-label overlaps,
- Number of line-label overlap,
- Average length of connecting lines,
- Number of bends in connecting lines,
- Average number of labels which change positions between frames,
- Average label displacement between frames,
- Illustration size and aspect ratio (1 is the best), and
- Frame rate.

Our system measures the values of evaluation parameters to help us in comparing the layouts at runtime. Moreover, a user evaluation should consider the following parameters:

- Contrastive comparison of different layouts,
- Personal layout score (pleasing, symmetry),
- Distinguish automatically generated layouts from hand-made ones,
- Time taken to match the co-referring labels and visual objects, and
- Error rates in the matching process.

7 DISCUSSION AND FUTURE WORK

The *layout compaction* is currently performed in local regions. Improved versions should consider the global distribution of empty space and reduce the amount of label stacking. In our approach, a single anchor point is used for each object. Long, thin, and branched objects are often marked with multiple anchor points, which are connected with branching connecting lines (see Figure 1-Right). For bigger objects (area features) internal labels should be used. Moreover, the system should support the integration of multiple layout style within an illustration. Finally, common semantic classifications of visual objects should be visualized through *labeling grouping* A full-fledged labeling system has to integrate all these aspect, and should be based on the notion of relevance to select an appropriate label number and content dynamically.

REFERENCES

[AdB88] C. Arcelli and G.S. di Baja. Finding Local Maxima in a Pseudo-Euclidean Distance Transform. *Computer Vision, Graphics, and Image Processing*, 43(3):361–367, 1988.

[BFH01] B. Bell, S. Feiner, and T. Höllerer. View Management for Virtual and Augmented Reality. In *Proc. of Symposium on User Interface Software and Technology*, pages 101–110, 2001.

[CMS95] J. Christensen, J. Marks, and S. Shieber. An Empirical Study of Algorithms for Point-Feature Label Placement. *ACM Transactions on Graphics*, 14(3):203–232, 1995.

[CP86] J.M. Clark and A. Paivio. Dual Coding Theory and Education. *Educational Psychology Review*, 3(3):149–210, 1986.

[DBET+99] G. Di Battista, P. Eades, R. Tamassia, , and I.G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Upper Saddle River, NJ, 1999.

[FP99] J.-D. Fekete and C. Plaisant. Excentric Labeling: Dynamic Neighborhood Labeling for Data Visualization. In *Proc. of SIGCHI*, pages 512–519, 1999.

[FR91] T.M.J. Fruchterman and E.M. Reingold. Graph Drawing by Force-Directed Placement. *Software-Practice and Experience*, 21(11):1129–1164, 1991.

[HIR+03] N. Halper, T. Isenberg, F. Ritter, B. Freudenberg, O. Meruvia, S. Schlechtweg, and Th. Strothotte. OpenNPAR: A System for Developing, Programming, and Designing Non-Photorealistic Animation and Rendering. In *Proc. of Pacific Graphics*, pages 424–428, 2003.

[Imh75] E. Imhof. Positioning Names on Maps. *The American Cartographer*, 2(2):128–144, 1975.

[KT98] K.G. Kakoulis and I.G. Tollis. A Unified Approach to Labeling Graphical Features. In *Proc. of the 14th Annual Symposium on Computational Geometry*, pages 347–356, 1998.

[MS91] J. Marks and S. Shieber. The Computational Complexity of Cartographic Label Placement. Technical Report TR-05-91, Center for Research in Computing Technology, Harvard University, 1991.

[PRS97] B. Preim, A. Raab, and Th. Strothotte. Coherent Zooming of Illustrations with 3D-Graphics and Text. In *Proc. of Graphics Interface*, pages 105–113, 1997.

[Rog92] A.W. Rogers. *Textbook of Anatomy*. Churchill Livingstone, Edinburgh, 1992.

[RP68] A. Rosenfeld and J. Pfaltz. Distance Functions in Digital Pictures. *Pattern Recognition*, 1(1):33–61, 1968.

[RSHS03] F. Ritter, H. Sonnet, K. Hartmann, and Th. Strothotte. Illustrative Shadows: Integrating 3D and 2D Information Displays. In *Proc. of Int. Conf. on Intelligent User Interfaces*, pages 166–173, 2003.

[SPP97] J. Sobotta, R. Putz, and R. Pabst, editors. *Sobotta: Atlas of Human Anatomy. Volume 2: Thorax, Abdomen, Pelvis, Lower Limb*. Williams & Wilkins, Baltimure, 12. English edition, 1997.

3D Free-Form Modeling with Variational Surfaces

Alvaro Cuno, Claudio Esperança, Paulo Roma Cavalcanti, Ricardo Farias
Universidade Federal do Rio de Janeiro
Programa de Engenharia de Sistemas e Computação/COPPE
Rio de Janeiro, Brazil
{alvaro, esperanc, roma, rfarias}@lcg.ufrj.br

ABSTRACT

We describe a free-form stroke-based modeling system where objects are primarily represented by means of variational surfaces. Although similar systems have been described in recent years, our approach achieves both a good performance and reduced surface leak problems by employing a coarse mesh as support for constraint points. The prototype implements an adequate set of modeling operations, "undo" and "redo" facilities and a clean interface capable of resolving ambiguities by means of suggestion thumbnails.

Keywords
Free-form modeling, stroke based modeling, RBFs.

1. INTRODUCTION

Typical 3D modeling systems are mostly designed to handle the creation of technical models, i.e., objects with precise measures or which must obey well-defined geometric rules. Such systems are not well-suited to handle so-called *free-form* models, which can be regarded as 3D models akin to 2D free-hand sketches. One reason for this is the fact that interaction in 3D relies almost exclusively on 2D projections, since the only feasible alternative for effectively working in 3D space is by employing costly and cumbersome virtual reality gear. Thus, the user must ultimately manipulate 2D features in order to accomplish 3D editing tasks.

Perhaps the most salient features of any given 3D model are its edges and silhouette lines. Igarashi et al. [Iga99] used this observation to build a prototype 3D free-form modeler called *Teddy*. In contrast with common 3D modelers, Teddy is easy and intuitive enough to be used even by small children. It relies on a scheme by which free-hand drawing strokes representing silhouette lines are used to build and modify smooth closed surfaces. Teddy also innovates over other 3D modelers by not using the standard *WIMP* (Windows, Icons, Menus and Pointers) interface paradigm. Rather, all interaction is based upon stroke recognition and a very small number of command buttons.

The Journal of WSCG, Vol.13, ISSN 1213-6964
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency – Science Press

Another key aspect that must be addressed in the construction of stroke-based interfaces is the resolution of ambiguities that may arise during a modeling session. For instance, a new stroke drawn by the user may be interpreted either as the cue for creating a new shell or as the profile of an extrusion operation. Our system copes with this problem by using a *suggestive interface* similar to the approach described in [Iga01]. Namely, thumbnail images representing the alternative results are displayed in a corner of the main display window, which must then be clicked by the user in order to select the desired outcome.

The remainder of this paper is organized as follows. Section 2 presents some relevant work related to the problem at hand. An overall description of the proposed system is presented in Section 3 and some concepts of the variational surfaces are introduced in Section 4. The involved algorithms are described in detail in Section 5. Some key aspects of the implementation are discussed in Section 6 and some results and limitations are presented in Section 7. Finally, some concluding remarks and suggestions for future work can be found in Section 8.

2. RELATED WORK

In the last few years, several experimental systems have been proposed which offer interfaces for the specification and construction of different types of three-dimensional scenes starting from 2D strokes [Zel96, Tol99, Mar99, Coh99, Coh00, Tol01, Iga01, Tai04]. Specially worthy of note is the *Teddy* system proposed by Igarashi et al. [Iga99], which can be used to create simple models with spherical topology with only a few strokes. An initial model is created by drawing a simple closed curve which is then inflated resulting in a blob-like object such that the curve approximates its silhouette. Additional strokes can then be used to extrude protrusions, cut, bend or smooth the model.

Modeling operations in *Teddy* are performed on a polygonal mesh representation of the surface. Some of these operations necessarily require the subsequent use of smoothing algorithms on the edited mesh. Nevertheless, some models end up with undesirable protuberances and wrinkles due to triangles with awkward characteristics. Besides, *Teddy* does not support the creation of multiple objects in the same scene and therefore operations to combine these are unavailable.

Karpenko et al. [Kar02] deal with the problem of undesired surface roughness by using *Variational Surfaces* as the main representation scheme. These surfaces are zero-sets of a class of implicit functions known as *RBF-based implicits*. The term *RBF* -- or Radial Basis Functions -- refers to the fact that the basis functions used in the creation of the implicit are radially symmetric. The key advantage of variational surfaces lies in that they are naturally smooth, since their construction can be regarded as an energy minimization process. This, however, leads to other problems. For instance, models with creases and tips cannot be easily created. Also, the performance of the system is heavily dependent on the number of constraint points used in defining the implicit. This is worsened by the fact that model editing operations are performed using a great number of mesh vertices produced by the visualization process.

Owada et al. [Owa03] present a system that generates volumetric models from 2D strokes. Besides making it possible to create, cut and extrude surface features, their approach also allows the specification of internal structures in the models with arbitrary topology. The main disadvantage of that system is that simple smooth surfaces can be modeled only with high storage and computation costs.

Blobmaker [Ara03] is prototype system quite similar to the one presented by Karpenko et al. Its main contribution lies in the use of skeletons for model construction. This allows the creation of objects with arbitrary topology and an efficient application of edition operations. However, the use of constraint points positioned irregularly on the surface may lead to surface leaks after a few modeling steps.

Recently, Tai et al. [Tai04] described a system based on convolution surfaces for the construction of free-form models starting from a silhouette curve. The resulting shape has circular cross-section, but can be conveniently modified through a sketched profile or shape parameters. But, unlike the prototypes discussed above, their system employs menus and sliders in its modeling interface.

3. SYSTEM DESCRIPTION

The prototype system allows the user to quickly create simple 3D models by drawing 2D strokes directly on the system window. Once the model is created, it can be further edited with operations such as merging, extrusion and piercing, which are also specified by inputting additional 2D strokes. Thus, the execution of an operation depends solely on the stroke form and where it was made, making it unnecessary to press any button or select menu options.

The user interface is composed of a design window and five command buttons. The ***init*** button starts a new modeling session, ***save*** saves the polygonal mesh of the modeled object, ***undo*** cancels the effects of the last operation, ***redo*** cancels the most recent ***undo*** command, and the ***quit*** button exits the system. Operations ***undo/redo*** work on a linear history of editing operations starting at the most recent invocation of the ***init*** command. This mechanism enables the user to review all operations made during a modeling session.

Input strokes are drawn by dragging the mouse with the left button pressed. A model can be moved on the *xy* plane by positioning the mouse over the model and then dragging it with the right button pressed. Translation along the *z* axis is accomplished in a similar way, but the middle button is used instead. Rotation uses an arc-ball interaction style: first, the center of rotation is specified by clicking on the model with the right button, the rotation angle and direction is then input by dragging the mouse with the right button.

Operations

A modeling session begins with an empty design window. The user specifies the model silhouette to be constructed by drawing a simple closed curve with a single stroke. The system then constructs a plausible 3D model based on the input silhouette. This is accomplished by inflating the curve in both directions by an amount proportional to its width, this is, narrow areas will become thin regions while wide areas generate fat regions [Iga99]. Figure 1 shows examples of input strokes and the corresponding 3D models constructed by the system.

Object creation operations may be performed many times, thus allowing the construction of scenes with multiple objects (see Figure 1(d)).

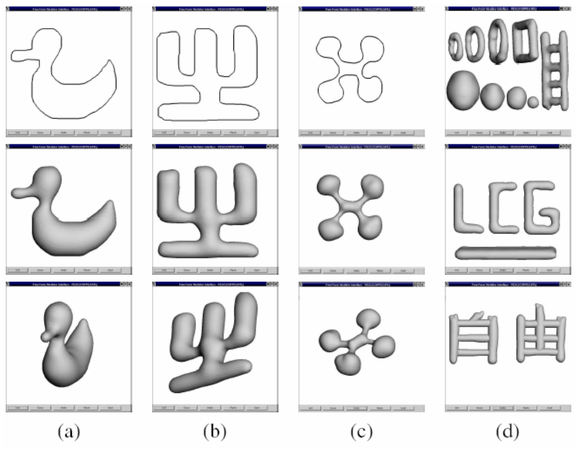


Figure 1. (a), (b) and (c) Object creation examples. (d) Scenes with multiple objects.

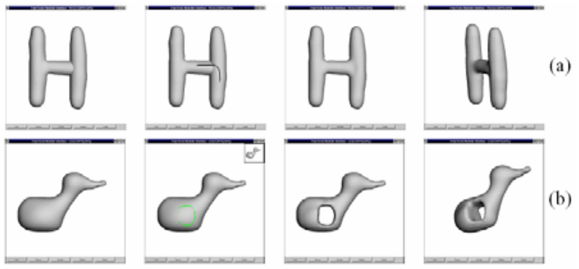


Figure 2. (a) Model merging. (b) Model piercing.

Model ***merging*** creates a new surface that approximates two previously existing models which are then discarded. The effect is to obtain a single implicit representation that smoothly blends two given shapes. The user commands this operation by drawing a simple open stroke starting inside the first input model and ending inside the second. The two input models must overlap in space for this operation to take place. Figure 2(a) shows an example.

The ***piercing*** operation can be used to make a hole in a model. The user must first draw a closed curve lying entirely inside the silhouette of the target model. This stroke can be interpreted in two ways by the system: either as a cue for performing a piercing operation or as an auxiliary element for performing an extrusion. At this point, the system will signal the ambiguity by displaying in the upper-left corner of the window a thumbnail image showing the result of the piercing operation. The user must click on this image in order to accept the operation (see Figure 2(b)). Any other action will trigger the other interpretation.

Extrusion is a modeling operation which allows the creation of a new protrusion on some part of a model. The extruded feature is described by a profile curve which is input as a simple open curve starting and ending inside the model's silhouette but extending beyond it. The area on which the protrusion will be "glued" can be defined either *explicitly* or *implicitly*. In the former case the gluing area is delimited by a closed curve drawn previously --see the preceding paragraph. In the latter case, the gluing area will correspond to a roughly circular region touching the two endpoints of the profile curve. Figure 3 illustrates this operation.

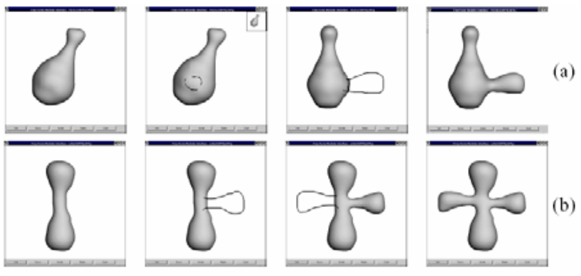


Figure 3. Extrusion examples: (a) using a base curve, and (b) automatic extrusion.

4. VARIATIONAL SURFACES

Although a through discussion of the math of implicit object modeling is outside the scope of this paper, for the sake of completeness, we try to lay down a few key concepts below. The interested reader is referred to the excellent introduction to the subject in [Tur99a].

The term "Variational Surface" refers to the zero-set of a RBF-based implicit function. Such functions are used in the context of scattered data interpolation. This is a problem where, given a set of *n* distinct constraint points $\{c_1, c_2, \ldots, c_n\}, c \in \Re^3$ and a set of *n* function values $\{v_1, v_2, \ldots, v_n\}$, it is sought a smooth function $f : \Re^3 \rightarrow \Re$ such that $f(c_i) = v_i$, for $i = 1 \ldots n$. The smoothness criteria usually involve some "deformation" energy that must be minimized. This entails the solution of a linear system with *n* equations. Solving this system is perhaps the most computationally intensive part of the system. We use a standard LU-decomposition algorithm for this task.

A variational surface can be modeled simply by choosing an adequate set of constraint points and associated values. The most used approach requires the placement of *n*/2 points with value equal to zero --these are known as *boundary constraint points*. Another set of *n*/2 points are obtained by displacing each boundary point by a small amount along the direction of the estimated surface normal at that point. These points, known as *normal constraint points*, are associated with a small positive constant *w* (see Figure 4).

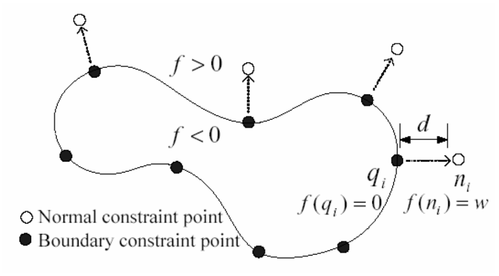


Figure 4. The normal constraint points *n*$_i$ are placed along the estimated normal vector at a distance d from boundary constraint points *q*$_i$. The function *f* is such that *f*(*x*) < 0 for *x* inside the curve and *f*(*x*) > 0 outside the curve.

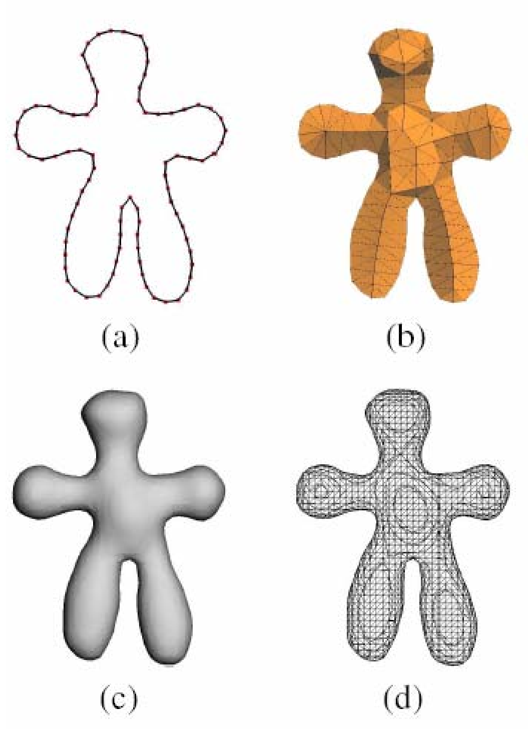


Figure 5. (a) 2D input stroke. (b) Coarse polygonal mesh of support for surface specification (177 vertices and 350 triangles). (c) Visualization of implicit surface *f* = 0, using smooth shading and (d) the triangular mesh (3620 vertices and 7236 triangles)

Any standard method for visualizing implicit objects can be used to render the modeled surface. In most cases, a polygonization scheme is employed and the resulting set of polygons is rendered using standard graphics hardware. It should be noted, however, that the polygonization scheme should be carefully chosen in order to minimize the number of function evaluations, since these are costly operations. We use a hierarchical variant of the Marching Cubes algorithm [Lor87].

5. ALGORITHMS

Creation

The creation algorithm consists essentially in specifying an adequate set of constraint points based on the user's input silhouette curve. The constraint points are chosen to coincide with the vertices of a coarse mesh built from the input stroke using an inflation algorithm. Figure 5 illustrates a global idea of the algorithm.

The construction of the coarse mesh follows the method described in [Iga99]. We found that this approach yields more pleasing results than the simpler algorithm adopted in [Kar02].

Merging

The merging operation consists in creating a new variational surface whose shape approximates the union of two other given surfaces. The algorithm consists of eliminating constraint points which are contained in the intersection of the two input shapes. Let us call *h* the resulting function and *f* and *g* the two input functions. Then, *h* contains a boundary constraint point *x* of *f* only if *g*(*x*) > 0. Similarly, *h* contains a boundary constraint point *y* of *g* only if *f*(*y*) > 0. Additionally, if a boundary constraint point is eliminated in this process, then the corresponding normal constraint point is also discarded. Figure 6 illustrates the idea.

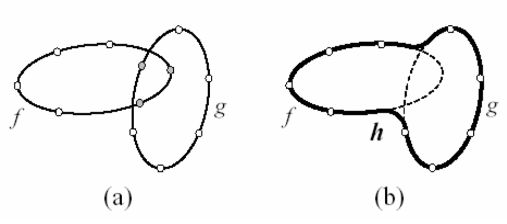


Figure 6. Merging illustration in 2D. (a) Constraint points positioned inside the intersection of the models represented for *f* and *g* are eliminated. (b) The new model represented by function *h* is built with points that remained after the elimination process.

Piercing

Let *f* be the function representing the model to be edited, *C* the 2D closed curve drawn by the user (represented by a simple polygon), and *h* the resulting model from this operation. Then, the piercing algorithm comprises the following steps:

1. Project each vertex C_i of *C* on the front-facing triangles of the polygonized model surface. Let F_i be the corresponding projected point. If the projection of any C_i yields more than one projected point, the piercing algorithm is aborted.

2. Similarly, project the vertices of C on the back-facing triangles of the polygonized model surface and call B_i the resulting projected vertices. As before, abort the algorithm if the more than one projection point is found for any given vertex.
3. Interpolate k evenly spaced points along each line segment F_iB_i. Let us call such points M_j. In our implementation, k = 3, i.e., three points are generated between each pair of vertices F_i and B_i.
4. Create an interpolating function g, which will be built by the creation procedure, but using F_i, B_i and M_j as boundary constraint points. The surface orientation is defined by placing an additional constraint point p placed at the approximate center of the shape and mapped to a negative value (-1 in our implementation). The position of p is estimated by computing the coordinate-wise average of all boundary constraint points. If this point does not lie inside curve C, then the piercing algorithm is aborted.
5. Perform the merging operation on f and g.

Explicit Extrusion

This type of extrusion is defined by two strokes: a base curve drawn directly on the model surface which defines the model area affected by the edition process, and a profile stroke. If f is the input model function, then the explicit extrusion is computed as follows:

1. Project the base curve on the polygonized object using the same rationale described in item 1 of the previous Sub-section. Let us use C to refer to this projected curve.
2. Project the profile curve on the plane that passes through the base curve's barycenter and is parallel to the viewing plane. Let us call P the resulting curve.
3. Create an interpolating function g using the vertices of C and P as boundary constraint points. Additionally, estimate normal constraint points by displacing the vertices of P outward.
4. Apply the merging operation to f and g.

Implicit Extrusion

This operation requires only an extrusion profile [Kar02]. The procedure is the following:

1. Select the silhouette vertices of the model's polygonized mesh vertices. A silhouette vertex is any vertex incident on two triangles whose normals point to opposite sides of the viewer plane. Find S and E, the silhouette vertices which are closest to the initial and end points of the profile curve, respectively.
2. Project the extrusion profile curve on the plane that passes by the middle point of line segment SE and is parallel to the viewing plane.
3. Create an interpolating function g using the vertices of the projected curve computed in the previous step as boundary constraint points. For each of these, add a normal constraint point by displacing it outward with respected to the curve.
4. Apply the merging operation between f and g.

6. IMPLEMENTATION DETAILS

The prototype system was written in the C++ language and the *OpenGL* library was used to render the polygonized models. All example models shown in this paper were built by the prototype system in a PC equipped with a 1.3 GHz AMD-Duron processor and 256 MB of main memory.

The system uses two main data structures: a scene representation and a command list. The scene is the model repository and the command list records the history of a modeling session (Figure 7).

Every time a new modeling operation is issued by the user, a corresponding command is inserted at the end of the command list. Depending on the command type, its execution can insert and/or remove models from the scene. For instance, a command “merge” will insert a new model in the scene, and will remove the input models.

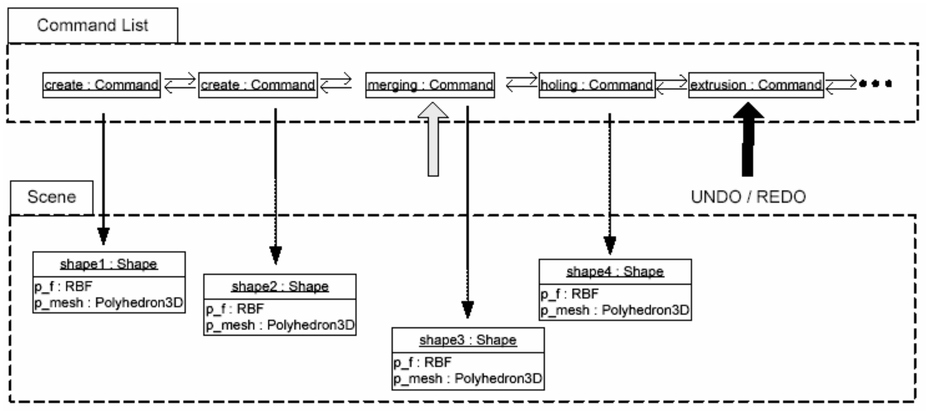


Figure 7. Main data structures of the system.

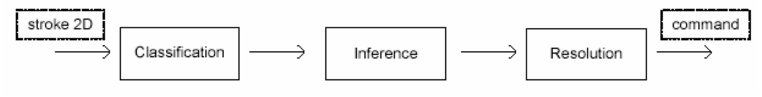


Figure 8. Stages for the command determination to execute starting from a 2D stroke.

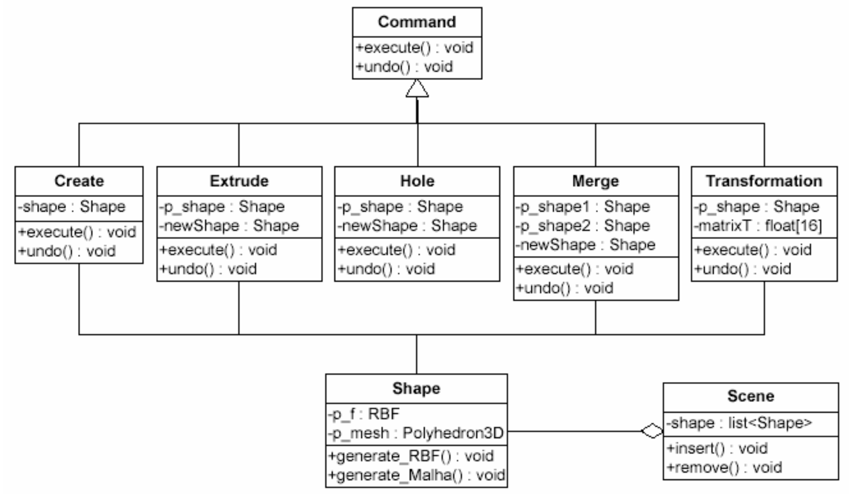


Figure 9. System class hierarchy.

Thus, the *Undo*/*Redo* mechanism works by scanning the command list in both directions replaying or undoing the commands appropriately.

The system determines the command type to be executed in response to the input 2D strokes using the following three-step approach (see Figure 8):

1. The classification stage determines the stroke type, i.e., simple or non-simple, closed or open.
2. Depending on the place where the stroke was drawn and on its type, the inference stage creates the appropriate command.
3. If an ambiguity is detected, the user is prompted to choose the desired outcome. The resolution stage then inserts the command in the list and executes it.

This approach is based on the ambiguities resolution proposal of Alvarado et al. [Alv2001].

A brief description of the system class hierarchy is presented in Figure 9. The class attributes labeled *p_shape* are pointers to models, while the absence of the prefix *p_* means a reference to the model itself.

Superclass **Command** is an abstract class with two methods: *execute()* and *undo()*. Method *execute()* executes the suitable actions for a command, while method *undo()* undoes the actions done by method *execute()*. For instance, in an extrusion operation, *undo()* removes the resulting model from the extrusion operation shape, and inserts the unextruded model *p_shape* again.

Create is a class that implements the model creation process. **Extrude** modifies the model pointed to by *p_shape* generating a resulting model *NewShape*.

The same happens with class **Pierce**. **Merge** produces a new shape *NewShape* starting from models *p_shape1* and *p_shape2*. Rigid motions (rotations and translations) are implemented in class **Transformation**.

Class **Shape** stores object geometry using two representations: *f*, an analytical representation of a RBF-based implicit function, and a triangle mesh generated by applying a polygonization algorithm on *f*.

Finally, class **Scene** contains a (possibly empty) model list that is manipulated by methods *insert()* and *remove()*.

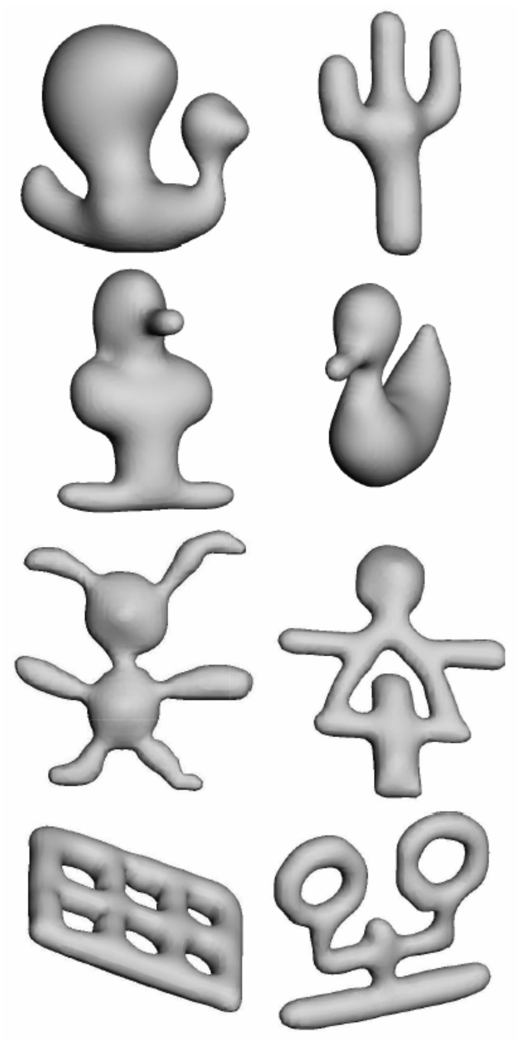


Figure 10. 3D models constructed with the prototype system.

7. RESULTS AND LIMITATIONS

Figure 10 shows some models built with our prototype system. They are smooth surfaces of arbitrary topology and exhibit a loose "look" which is characteristic of free-hand 2D drawings. The interested reader may access http://www.lcg.ufrj.br/Projetos/ffmodelling and download some of these models in OFF format [Ros89].

Due to the nature of the radial basis functions used in the underlying representation of our system, models with creases or sharp features cannot be created. Also, sometimes the result of a modeling operation is unintuitive. This is the case, for instance, when two small objects containing relatively few constraint points are merged (see Figure 11).

Another current limitation of the system lies in the fact that the piercing operation cannot be applied other than on relatively simple local geometries. For instance, if the intended hole would pierce the surface more than once, then the operation fails (see Figure 12(a)). In some other cases, the hole fails to properly pierce the model (Figure 12(b)).

Some modeling operations may incur in a problem known as ***surface leak***. This is due to the constraint points being distributed irregularly. Figure 13 illustrates this problem. We deal with this problem by using a coarse polygonal mesh introduced in the model creation process (see the Creation sub-section).

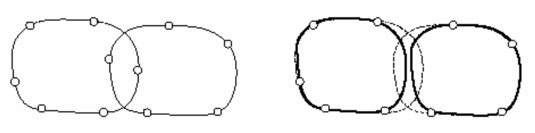


Figure 11. Merging two small models may yield unintuitive results.

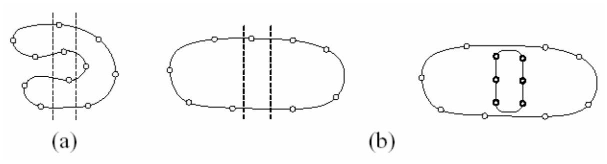


Figure 12. Limitations of the piercing operation. (a) Piercing fails in complicated situations. (b) Hole may fail to pierce the surface completely.

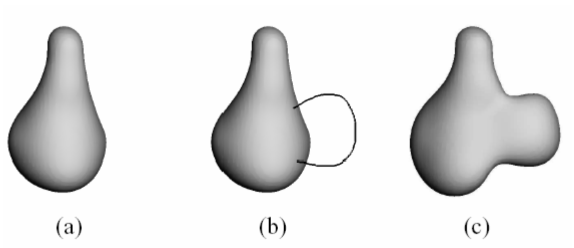


Figure 13. Surface leak problem. (a) Initial interpolation. (b) Extrusion. (c) Leak after extrusion.

8. CONCLUSIONS AND FUTURE WORK

Free-form modeling supported by RBF-based implicits enjoys quite a few advantages over traditional approaches employing parametric surfaces. In particular, the generated models are naturally smooth and many intuitive modeling operations can be implemented with relative ease. Its foremost limitations can be attributed to the time complexity of the scattered point interpolation scheme used.

Therefore, a natural extension of the present work consists of adopting more eficient interpolation schemes such as the FastRBF [Car01], which will enable our system to handle models with increased complexity. This, in turn, will help us cope with the surface leaking problems. Another venue that should be explored is the adoption of a more careful sampling strategy for the constraint points.

The set of modeling operations available in our systems is somewhat limited still. We are working on an enhanced algorithm for the piercing operation, as well as other operations such as cutting and bending. Regarding the visualization process, we are experimenting with a novel polygonization approach with some promising results (access http://www.lcg.ufrj.br/Projetos/ffmodelling for details).

9. ACKNOWLEDGMENTS

We are grateful to CNPq for providing financial support for the first author.

10. REFERENCES

[Alv01] Alvarado, C. and Davis, R. Resolving ambiguities to create a natural computer based sketching environment. In *Proceedings of IJCAI-*2001, pages 1365-1371.

[Ara03] Araujo, B. and Jorge, J. Blobmaker: Free-form modeling with variational implicit surfaces. In 12o *Encontro Portugues de Computação Grafica* (EPCG) - 2003.

[Car01] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of SIGGRAPH 2001*, pages 67-76. ACM Press. 2001.

[Coh00] Cohen, J. M., Hughes, F., and Zeleznik, R. C. Harold: A world made of drawings. In *Proceedings of NPAR 2000*, pages 83-90. ACM.

[Coh99] Cohen, J. M., Markosian, L., Zeleznik, R. C., Hughes, J. F., and Barzel, R. An interface for sketching 3D curves. In *Proceedings of Symposium on Interactive 3D Graphics*, pages 17-21. ACM Press.

[Din02] Dinh, H. Q., Turk, G., and Slabaugh, G. Reconstructing surfaces by volumetric regularization using radial basis functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(10):1358-1371, 2002.

[Iga01] Igarashi, T. and Hughes, J. A suggestive interface for 3D drawing. In *Proceedings of ACM UIST'01*, pages 173-181, 2001. ACM Press.

[Iga99] Igarashi, T., Matsuoka, S., and Tanaka, H. Teddy: A sketching interface for 3D freeform design. In *Proceedings of SIGGRAPH 99*, pages 409-416. ACM Press.

[Kar02] Karpenko, O., Hughes, J. F., and Raskar, R. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum*, 2002.

[Lor87] Lorensen, W. and Cline, H. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163-169, 1987.

[Mar99] Markosian, L., Cohen, J. M., Crulli, T., and Hughes, J. Skin: a constructive approach to modeling free-form shapes. In Proceedings of SIGGRAPH 99, Annual Conference Series, pages 393-400. ACM Press, 1999.

[Owa03] Owada, S., Nielsen, F., Nakazawa, K., and Igarashi, T. A sketching interface for modeling the internal structures of 3D shapes. In *Proceedings of 3rd International Symposium on Smart Graphics*, pages 49-57. Springer, 2003.

[Ros89] Rost, R. J. (1989). A 3D object file format.
http://www.dcs.ed.ac.uk/home/mxr/gfx/3d/off.spec.

[Tai04] Tai, C., Zhang, H., and Fong, C. Prototype modeling from sketched silhouettes based on convolution surfaces. *Computer Graphics Forum*, 23(1):71-83, 2004.

[Tol99] Tolba, O., Dorsey, J., and McMillan, L. Sketching with projective 2D strokes. In *Proceedings of the 12th annual ACM symposium on UIST*, pages 149-157, 1999.

[Tol01] Tolba, O., Dorsey, J., and McMillan, L. A projective drawing system. In *Proceedings of 2001 ACM Symposium on Interactive 3D Graphics*, pages 25-34, 2001.

[Tur99a] Turk, G. and O'Brien, J. Shape transformation using variational implicit functions. In *Proceedings of SIGGRAPH 99*, pages 335-342, 1999.

[Tur99b] Turk, G. and O'Brien, J. Variational implicit surfaces. Technical Report, Georgia Institute of Technology, 1999.

[Tur02] Turk, G. and O'Brien, J. Modelling with implicit surfaces that interpolate. ACM *Transactions on Graphics*, pages 855-873, 2002.

[Zel96] Zeleznik, R. C., Herndon, K. P., and Hughes, J. F. SKETCH: An interface for sketching 3D scenes. In *SIGGRAPH 96 Conference Proceedings*, pages 163-170, 1996

The Adaptive Thin Shell Tetrahedral Mesh

Kenny Erleben
Dept. of Computer Science,
University of Copenhagen,
Denmark
kenny@diku.dk

Henrik Dohlmann
3DLab, School of Dentistry,
University of Copenhagen,
Denmark
henrikd@lab3d.odont.ku.dk

Jon Sporring
Dept. of Computer Science,
University of Copenhagen,
Denmark
sporring@diku.dk

ABSTRACT

Tetrahedral meshes are often used for simulating deformable objects. Unlike engineering disciplines that often focuses on accuracy, computer graphics is biased towards stable, robust, and fast methods. In that spirit we present an approach for building an adaptive inward shell of the surface of an object. The goal is to device a simple and fast algorithm capable of building a topologically consistent tetrahedral mesh. The tetrahedral mesh can be used with several different simulation method, such as the finite element method (FEM), and the main contribution of this paper is a novel tetrahedral mesh generation method based on adaptive surface extrusion.

Keywords
Tetrahedral Mesh, Erosion, Extrusion, Tessellation, Shell, Prism

1 INTRODUCTION

Given a 3D polygonal model created by a 3D artist, it is often a challenge to create a spatial structure for simulating a deformable object. Creating a tetrahedral mesh often results in an enormous number of tetrahedra, hence a more coarse tetrahedral mesh is often sought in order to achieve real-time performance. In this paper a mesh generation method is proposed, that works directly on the surface of a mesh, avoids a constrained Delaunay triangulation, is easy to implement, and yields a tetrahedral count, which is linearly proportional with the number of mesh faces.

Given a watertight twofold boundary representation of an object such as a connected triangular mesh, a prism is generated for each triangle by extruding the triangle inward. The result is a volumetric mesh consisting of connected prisms. These prisms can easily be tessellated into tetrahedra to create the first layer of the thin shell tetrahedral mesh. Succeeding layers can be created by recursively applying this approach. Figure 1 shows examples of thin shells from volumetric meshes.

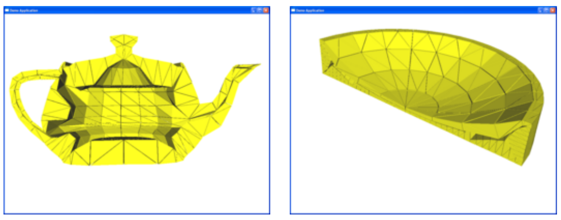

(a) Teapot (b) Bowl

Figure 1: Cut-views showing the shell layers inside the volumetric meshes generated with our method.

Polygonal models are seldom twofold, but often suffers from several kinds of degeneracies. The idea we have illustrated is obviously capable of handling an open boundary, but cases where edges share more than two neighboring faces, or where edges self-loop, will generate prisms, which overlap or degenerate into a zero-volume prisms. In such cases a mesh reconstruction algorithm [NT03] must be applied first.

The suggested prism generation is reminiscent of an erosion operation with a spherical structural element on the polygonal model. The radius of the sphere corresponds to the extrusion length. It is well known that working directly on the boundary representation [Set99] is fast and simple, but topological problems arises easily such as shocks [KTZ95]. The counter-part to shocks are degenerated prisms, that is prisms with less than 6 vertices. These shocks turn out to be the limit on the extrusions lengths.

The Journal of WSCG, Vol.13, ISSN 1213-6964
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency–Science Press

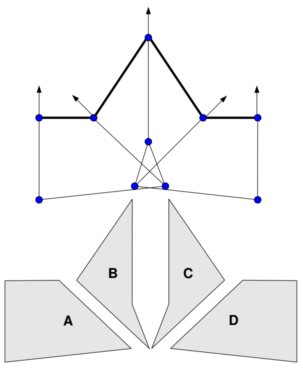


Figure 2: Degenerate prisms results from a too big inward extrusion. This is an example of a swallow tail.

Existing tetrahedral mesh generation methods in the literature typically create an initial, blockified tetrahedral mesh from a voxelization or signed distance map. Afterwards, nodes are iteratively repositioned, while tetrahedra are sub-sampled in-order to improve mesh quality [MT03, PS04, MBTF04]. In contrast to these methods, our is surface-based. An implementation of our method presented is available at [Ope04].

2 MINIMAX INWARD EXTRUSION

The thin shell layer is produced by extruding each triangle in the mesh inwardly, thus producing prisms. We will require the following three properties of the prisms: No two prisms must intersect each other, prisms must have volume larger than zero, and all prisms must be convex. Unfortunately, even for a perfectly connected twofold triangle mesh, too large inward extrusions will cause problems as illustrated in Figure 2. In the figure, the large extrusion length causes prisms B and C to become non-convex. Furthermore, A and D, B and D, C and A, and B and C are overlapping. Fortunately, these degenerate and unwanted prisms can be avoided by reducing the extrusions. Thus, we must seek an upper bound on how far, we can extrude the triangle faces inwardly without causing degenerate prisms.

To make the inward extrusion, we first compute the outward, angle-weighted normals [AB03] for all vertices. Then for a triangle consisting of three vertices $\vec{p}_1$, $\vec{p}_2$, and $\vec{p}_3$, with angle weighted normals $\vec{n}_1$, $\vec{n}_2$, and $\vec{n}_3$, the inward extruded prism is defined by the six points: $\vec{p}_1$, $\vec{p}_2$, and $\vec{p}_3$, and

$$\vec{q}_1(\varepsilon) = \vec{p}_1 - \vec{n}_1\varepsilon, \tag{1}$$
$$\vec{q}_2(\varepsilon) = \vec{p}_2 - \vec{n}_2\varepsilon, \tag{2}$$
$$\vec{q}_3(\varepsilon) = \vec{p}_3 - \vec{n}_3\varepsilon, \tag{3}$$

where $\varepsilon > 0$ is the extrusion length. The notation is illustrated in Figure 3. Clearly ε must be strictly positive, however to further guarantee convexity and positive volume of the prisms, we will use the normal of the extruded faces to generate an upper bound on ε.

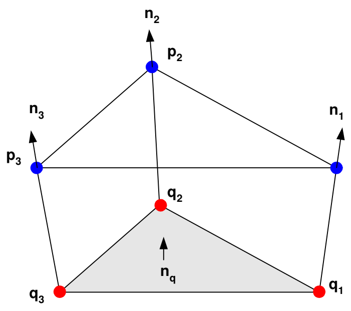


Figure 3: The six points and pseudo normals defining the prism and extrusion direction.

The direction of the normal of the extruded face, $\vec{n}_q$, can be found from $\vec{q}_1$, $\vec{q}_2$, and $\vec{q}_3$, using the cross-product:

$$\vec{n}_q(\varepsilon) = (\vec{q}_2(\varepsilon) - \vec{q}_1(\varepsilon)) \times (\vec{q}_3(\varepsilon) - \vec{q}_1(\varepsilon)). \tag{4}$$

By the distributive property of the cross product, we find a second order polynomial in ε,

$$\begin{aligned}\vec{n}_q(\varepsilon) = &\underbrace{((\vec{p}_2 - \vec{p}_1) \times (\vec{p}_3 - \vec{p}_1))}_{\vec{c}} \\ &+ \underbrace{((\vec{p}_2 - \vec{p}_1) \times (\vec{n}_1 - \vec{n}_3) + (\vec{n}_1 - \vec{n}_2) \times (\vec{p}_3 - \vec{p}_1))}_{\vec{b}}\varepsilon \\ &+ \underbrace{((\vec{n}_1 - \vec{n}_2) \times (\vec{n}_1 - \vec{n}_3))}_{\vec{a}}\varepsilon^2 \\ = &\vec{a}\varepsilon^2 + \vec{b}\varepsilon + \vec{c}. \end{aligned} \tag{5}$$

Observe that $\vec{c} \neq \vec{0}$, since its magnitude is equal to twice the area of the triangle being extruded.

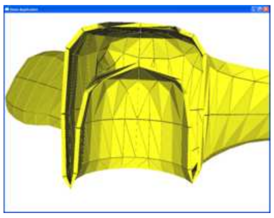
To ensure positive volume and convexity, it is sufficient to ensure positivity of the dot product of the normal of the extruded face, $\vec{n}_q$, with the pseudo normals, $\vec{n}_1$, $\vec{n}_2$, and $\vec{n}_3$. I.e. we must have that,

$$\vec{n}_1 \cdot \vec{n}_q(\varepsilon) > 0, \tag{6}$$
$$\vec{n}_2 \cdot \vec{n}_q(\varepsilon) > 0, \tag{7}$$
$$\vec{n}_3 \cdot \vec{n}_q(\varepsilon) > 0. \tag{8}$$

Using (5), we may formulate the constraints as the following system of inequalities,

$$\begin{bmatrix} \vec{n}_1 \cdot \vec{a} & \vec{n}_1 \cdot \vec{b} & \vec{n}_1 \cdot \vec{c} \\ \vec{n}_2 \cdot \vec{a} & \vec{n}_2 \cdot \vec{b} & \vec{n}_2 \cdot \vec{c} \\ \vec{n}_3 \cdot \vec{a} & \vec{n}_3 \cdot \vec{b} & \vec{n}_3 \cdot \vec{c} \end{bmatrix} \begin{bmatrix} \varepsilon^2 \\ \varepsilon \\ 1 \end{bmatrix} > 0. \tag{9}$$

The largest positive ε fulfilling the system of constraints is the upper bound on the inward extrusion lengths. We find the upper bound by solving for each row the roots of the the second order polynomial in ε. The three rows


(a) Propeller

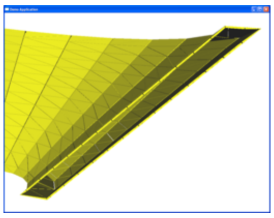

(b) Funnel

Figure 4: Close-up cut-views showing how a global extrusion length causes thin shell layers.

yield a total of 6 roots. If no positive root exist, then $\varepsilon = \infty$, otherwise ε must be less than the smallest positive root.

The three convexity constraints ensure that no neighboring prism will intersect each other, nor will the prism turn its inside out, i.e. flipping the extruded face opposite the original face. A global extrusion length for the entire layer can be found by iterating over each prism. The global extrusion length of the layer is found as

$$\varepsilon = \min\left(\varepsilon^0, \ldots, \varepsilon^{n-1}\right), \tag{10}$$

where ε^i is the extrusion length for the i'th prism. Afterwards, it is a simple matter to compute the actual extrusion and generating the prisms. In some cases using the maximum possible extrusion length of a prism can degenerate it. The degenerated prism will have a zero-area extruded face and is easily detected and treated [ED04]. During triangulation degenerate prisms can be dealt with by insertion of an extra internal corner point.

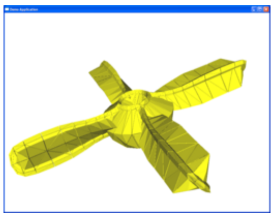
3 ADAPTIVE EXTRUSION

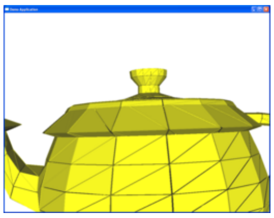
Badly shaped surface triangles can in some cases cause an inefficient small global extrusion length, as illustrated in Figure 4. This is caused by the global optimization in (10), in which a single prism near high curvature will dictate the thickness of the entire layer. Multiple layers will give an impression of a solid or dense object, but will introduce a large number of tetrahedra. The thin shell also causes the tetrahedra to turn into slivers, if the global extrusion length is too small. To overcome such inefficient thin shell layers, we propose to use an adaptive extrusion length.

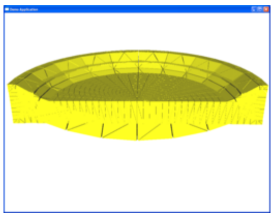
To calculate the adaptive extrusion length, a surface vertex is assigned an extrusion length equal to the minimum extrusion length of the neighboring prisms, of which the vertex is part. Thus for vertex, v

$$\varepsilon_v = \min_{p \in \mathbf{P}(v)} \varepsilon^p, \tag{11}$$

where $\mathbf{P}(v)$ denotes the set of all prisms, of which v is part. Choosing the size of the adaptive extrusion length


(a) Propeller


(b) Teapot


(c) Bowl

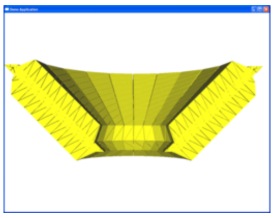

(d) Funnel

Figure 5: Adaptive extrusion length using (11) causes self intersection of the thin shell layers.

in this way does not violate the convexity requirement to the prisms, since given a convex prism, one can always shrink any one of the extrusion lines without destroying convexity.

Nevertheless, this is a local solution, and as is shown in Figure 5, vertices with large extrusion lengths causes self-intersections with opposing faces. Our remedy is to use a root-search method to search for a smaller layer thickness without self-collisions. The idea is to re-cast the problem into a simulation formulation, where the extrusion length is thought of as a time parameter of an evolving surface. Thus we seek the point in time, where the evolving surface touches itself from opposite sides. The simulation loop consist simply of the following two steps:

- Perform collision detection,
- Update the extrusion length values.

These are repeated for a fixed number of iterations.

For each extrusion line we will keep a minimum value of the extrusion length $\varepsilon_{\min}$, a maximum value of the extrusion length $\varepsilon_{\max}$, and a current value of the exturusion length ε. The interval $[\varepsilon_{\min}, \varepsilon_{\max}]$ represents the range of values, where we look for a solution for ε. Initially the value of the minimum, maximum, and current extrusion length are all set equal to (11). During the search for a solution, the minimum and current extrusion length values will be changed, but the maximum length value is left unchanged.

The parameter $\varepsilon_{\min}$ is the largest value that will not destroy the convexity requirement and is therefore always changed to a value that guarantees non-penetration. The parameter ε is the next guess for a non-penetrating extrusion line, and it is set to the half-way point between $\varepsilon_{\min}$ and $\varepsilon_{\max}$.

A spatial grid data structure [TBHPG03] is used to perform collision detection. During a first pass the axis aligned bounding boxes of all the extrusion lines are mapped into the spatial grid, and then in a second pass the axis aligned bounding boxes of the prisms are used to query for overlap with the extrusion lines. Whenever an overlap is found a penetration test is performed between the prism and extrusion line, The brute-force penetration test consist of testing, whether the extrusion line penetrates the five faces of the prism. This can be optimized to perform only penetration testing against the original surface face and the extruded face of the prism. If the extrusion line originates from a vertex shared with the prism, then the penetration test is ignored.

Upon having completed the collision query, a set of colliding extrusion lines and prisms are returned. Now we iterate over all these pairs, and mark each extrusion line, while finding the intersection point with the surface and extruded faces of the prism. If the distance to the intersection point from the originating surface vertex is lower than the minimum extrusion length, then the minimum extrusion length of the vertex is updated to this distance. If a non-penetrating extrusion line is found, then the current extrusion length ε yields a new possible value for the minimum extrusion length $\varepsilon_{\min}$. However instead of simply setting the minimum extrusion length equal to the current extrusion length, our experiments indicate that it is better to down-scale the value of current extrusion length, before assigning it to the minimum extrusion length. This is because, it is likely that the extrusion lines on the opposite side of the shell layer also will increase their minimum extrusion lengths. Down-scaling their values will reduce the chance for the growing extrusion lines to cause a self-collision.

Figure 6 shows a pseudo-code version of the simulation loop, which iteratively adjusts the extrusion lengths.

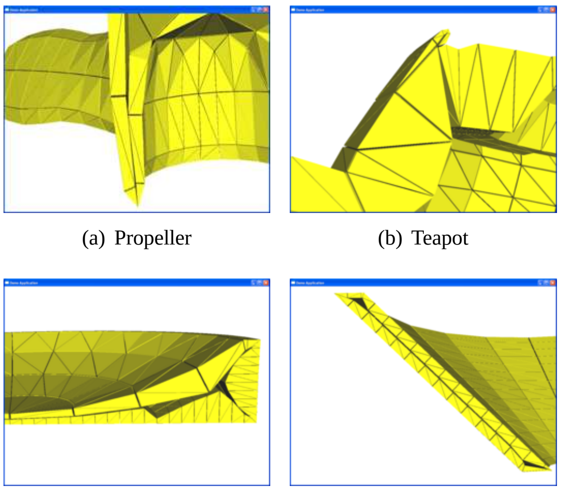
Upon completion of the last iteration of the simulation loop, the value of $\varepsilon_{\min}$ and not ε is used as the extrusion length, since only $\varepsilon_{\min}$ is guaranteed not to cause any self-collisions. Figure 7 shows close-ups of cut-views of volumetric meshes generated using the iteratively adjusted adaptive extrusion length method. Notice that the adaptive extrusion length is small near sharp rigdes, and at flat regions the adaptive extrusion length are increased to the point, where the extruded surface meets with prisms from the opposite of the object.

4 PRISM TESSELLATION

For solid state simulations it is convenient to have objects on tetrahedra form, hence we will tesselate our prisms into tetrahedra. Due to space limitations, we will have to disregard degeneracies, these are however treated in details in [ED04]. For non-degenerate prisms having 6 corners, 3 is the minimum number of tetrahedra we can tesselate the prism into, and this tesselation

```
for i=1 to max iteration do
  Results R = collision(lines,prisms)
  for each (line,prism) in R do
    let p be originating point of line
    let v be intersection point of line
    $\varepsilon_{\min}$(line) = min($\varepsilon_{\min}$(line),dist(p,v))
    mark(line) = true
  next (line,prism)
  for each line in lines do
    if not mark(line) then
      tmp  = $\varepsilon$(line)*0.9;
      if tmp > $\varepsilon_{\min}$(line) then
        $\varepsilon_{\min}$(line) = min(tmp,$\varepsilon_{\max}$(line))
      end if
    end if
    tmp = ($\varepsilon$(line) + $\varepsilon_{\min}$(line))/2
    $\varepsilon$(line) = min($\varepsilon_{\max}$(line),tmp)
    q(line) = p(line) - n(line) * $\varepsilon$(line)
    mark(line) = false
  next line
next i
```

Figure 6: Pseudo-code for iterative adjustment of adaptive extrusion length.


(a) Propeller (b) Teapot

(c) Bowl (d) Funnel

Figure 7: Close-ups of volumetric mesh cut-views, showing the effect of the iteratively adjusted adaptive extrusion length, which gives a thick and non-overlapping layer.

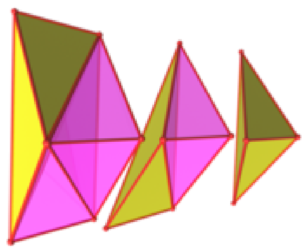


Figure 8: A Prism iteratively tessellated into 3 tetrahedra from left to right. It is helpful to imagine that the rightmost face has been inwardly extruded to produce the leftmost face. The yellow tetrahedra illustrate the iteratively produced tetrahedra.

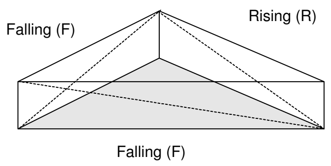


Figure 9: Classification of prism sides as falling (F) or rising (R).

is illustrated in Figure 8. For methods such as Finite Element Modelling (FEM) it is useful for neighbouring prisms to be tesselated such that the generated triangular faces match. We call this tesselation consistency, and it results in a global combinatorial problem.
A prism can be tesselated into three tetrahedra in 6 different ways. In order to classify the 6 types of tesselations, we will mark the rectangular sides of a prism as falling (F) or rising (R). The edge type depends on whether the tesselation edge is falling or rising as we travel along the extruded prism face in counter clock wise manner. This is illustrated in Figure 9. We observe that our tetrahedra tesselation strategy will always have two prism sides of the same type, and the last side of opposite type. Thus we can only have 6 different patterns tabulated in Table 1. The consistency requirement implies that if one side of a prism is marked as F, then

F	R	R
R	F	R
R	R	F
R	F	F
F	R	F
F	F	R

Table 1: The 6 three-tetrahedra tesselation types.

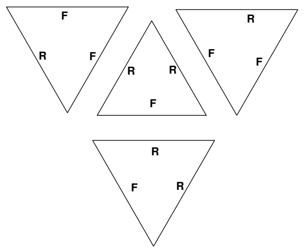


Figure 10: Tesselation example. A simple 3D surface mesh (a tetrahedron) as seen parallel to the surface normal. The letters R/F denotes the choice of Rising/Falling triangulation of the prism sides, which cannot be seen in this projection.

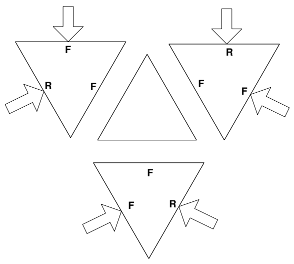


Figure 11: Inconsistent tesselation example. The middle prism will have the same type on all sides, which is not possible.

the neighboring prism will have marked the same side as R. In short, no neighboring prisms will have a side marked with the same type. A simple tesselation example is shown in Figure 10.
Our algorithm for ensuring global consistency is as follows: We start at a single prism and choose one of the 6 tesselation types. Then we visit the neighboring prisms and choose a tesselation, which is consistent with neighboring prisms already tesselated.
With this algorithm, inconsistency may arise as shown in Figure 11. Here, the middle prism is the last prism to be visited. Clearly, it is impossible to assign a tesselation type to the prism, since all three sides should have the same type. This can be repaired by picking one of the neighboring prisms and flipping the type of its shared edge. This action will not change the type of any of the edges marked with arrows in Figure 11. In this case no further inconsistencies are introduced, and the repairing action of the example does not have large scale effect.
Fixing inconsistency locally is attractive, since it limits the computation time, but there is no guarantee that a local solution always can be found. An example of a non-local problem is the dead-lock shown in the top of Figure 12. In this example, none of the edges shared with the inconsistent prism can be flipped without creating an inconsistent neighboring prism, since all the edges marked with arrows are of the same type. The solu-

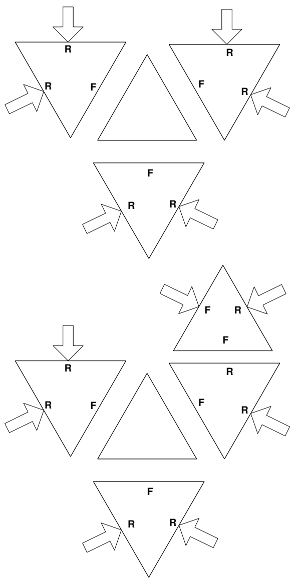


Figure 12: Top picture shows a inconsistent tesselation in a dead-lock. The bottom picture shows that inconsistency problem have been propagated to neighboring prisms further away by extended the region where we search for possible edge flips.

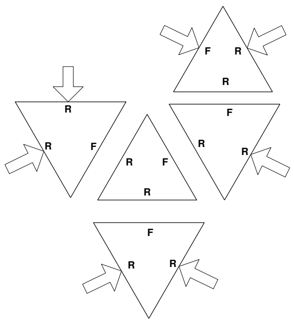


Figure 13: The rippling solution to the dead-locked case shown in Figure 12.

tion to the problem is shown in the bottom of Figure 12. We let the inconsistency ripple as water waves over to neighboring prisms, in a search for a single prism, where an edge flip does not give rise to a new inconsistency. When such a prism is encountered, we track the trajectory of the ripple wave-front back to the originating inconsistent prism and flip all shared edges lying on this path. The result of the rippling for this specific case is shown in Figure 13. Notice that two edges are flipped, which are the edges lying on the path from the unassigned prism to the prism that could be flipped. Also notice that all edges marked with arrows are unaffected by the rippling action. This property ensures that the rippling action will not cause inconsistencies in any prisms elsewhere in the mesh.
A pseudo-code of the tesselation-pattern-finding algorithm is shown in Figure 14. It is our experience

```
algorithm tesselation-pattern()
  Queue $Q$
  Push first prism onto $Q$
  While $Q$ not empty do
    Prism $p = \text{pop}(Q)$
    mark $p$ as visited
    $N =$ neighboring prisms of $p$
    if $N$ is not tesselated then
      pick random pattern of $p$
    else if consistent pattern with $N$
      assign consistent pattern to $p$
    else
      if exist $n \in N$ that can be flipped
        flip type of shared edge with $p$
        assign constent pattern to $p$
      else
        perform-rippling
      end if
    end if
    for all unvisited $n \in N$ do
      $\text{push}(Q, n)$
    next $n$
  End while
End algorithm
```

Figure 14: Pseudo-code for determining tesselation pattern.

that the rippling distance rarely exceeds more than 1–2 faces [ED04]. The tesselation algorithm is thus a computational cheap and fast solution to an unpleasant problem.

5 RESULTS

The adaptive thin tetrahedral shell mesh method has been tested on several surface meshes, 14 of these surface meshes are shown in Figure 15, and number of iterations and total running time for these meshes are listed in Table 2. In [ED04] the time-complexity of the actual tesselation was shown to scale linearly with the number of faces in the original surface mesh. The running time is therefore clearly dependent on the shape of the original surface mesh. As can be seen from the table, surface meshes with small thin structures have the worst running time. This is because in every second iteration of the iterative adjustment of the extrusion length, extrusion lines in these thin regions are increased, leading to penetrations. In the succeding iteration the extrusion lines are shortened to remove the penetration. This oscillating behaviour only slowly converges towards a solution.
Figure 16 shows plots of the number of collisions per iteration. The plots indicate that the iterative adjustment of the adaptive extrusion lengths converges toward a solution, when given enough iterations. The plots however also indicate that the number of collisions is not a strictly monotone decreasing function. It is thus difficult to say anything conclusive about the convergence rate.
Cut-views of the generated tetrahedral meshes can be

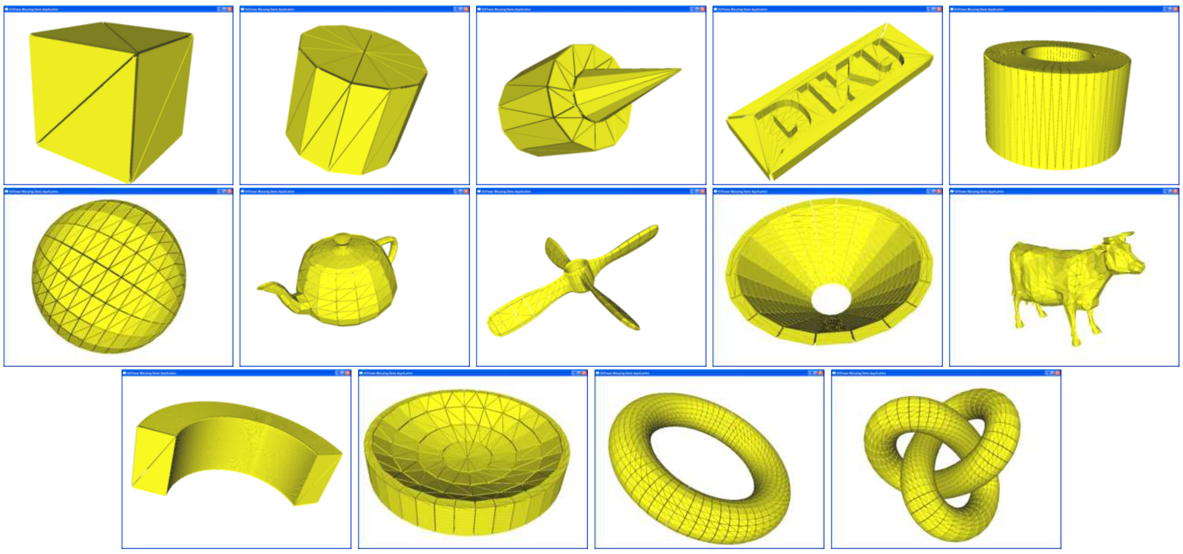


Figure 15: The 14 original surface meshes.

	$\|F\|$	$\|I\|$	time(secs.)
box	12	1	0.01
cylinder	48	1	0.01
pointy	96	100	0.42
diku	288	100	2.95
tube	512	1	0.15
sphere	760	1	0.26
teapot	1056	76	14.05
propeller	1200	72	19.89
funnel	1280	45	13.89
cow	1500	100	37.03
bend	1604	1	0.74
bowl	2680	85	136.75
torus	3072	1	1.49
knot	5760	1	5.94

Table 2: Performance Statistics using Iterative Adjustment. Maximum iteration count was set to 100 in all 14 test cases. The $|F|$-column gives the face count of the meshes, and the $|I|$-column shows the number of iterations.

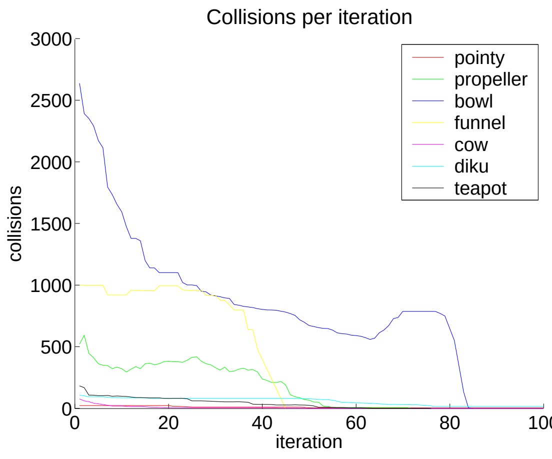


Figure 16: Collisions detected in each iteration. Collision-free test cases are not shown.

seen in Figure 1, 7, and 17. A user specified global maximum extrusion length was used, hence not all tetrahedral meshes fill out the inside void. The figures clearly show that the adaptive method is capable of filling out the inside of a surface mesh much more efficiently than the global extrusion length solution.

6 DISCUSSION

In this paper we have presented results showing that it is possible to generate a thin adaptive shell without topological errors. Our results show that the adaptive thin shell tetrahedral mesh generation method is versatile, robust, simple to implement, and yields useful results.

For the proposed consistent tesselation, we have not yet proven that it is always possible to find a consistent pattern of rising and falling tesselation edges. Neverthe-

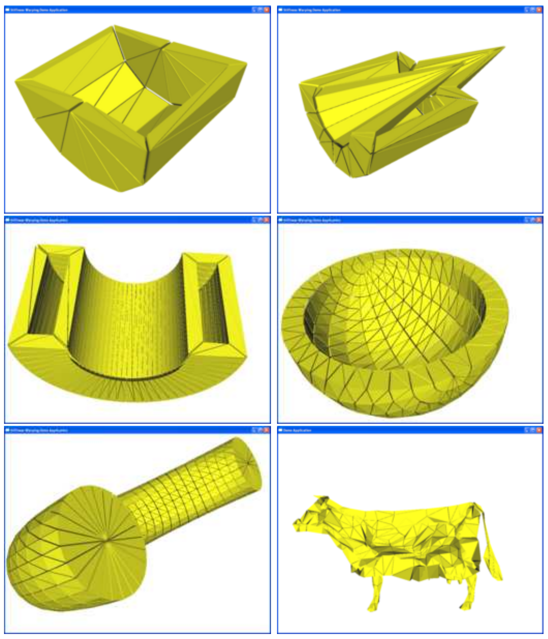


Figure 17: Cut-views of a few selected meshes.

less, we have not yet encountered difficulties with the method, and we believe that the combinatorial problem of finding a consistent tesselation pattern is tractable. We leave the proof for future work.
Lastly the iterative adjustment of the adaptive extrusion lengths can be improved in two ways. Firstly, the convergence rate could be improved to yield faster running times. Secondly, as seen in Figure 1, 7, and 17 some prisms seem to loose the race in increasing their extrusion lengths, before the algorithm terminates. The effect is most noticely seen in case of the bowl mesh, where some prisms could be extruded more to reduce empty space inside the mesh. We leave both these problems for future work.

References

[AB03] Henrik Aanæs and J. Andreas Bærentzen. Pseudo–normals for signed distance computation. In *Proceedings of VISION, MODELING, AND VISUALIZATION*, 2003.

[ED04] Kenny Erleben and Henrik Dohlmann. The thin shell tetrahedral mesh. In Søren Ingvor Olsen, editor, *Proceedings of DSAGM*, pages 94–102, August 2004.

[KTZ95] Benjamin B. Kimia, Allan R. Tannenbaum, and Steven W. Zucker. Shapes, shocks, and deformations I: The components of two-dimensional shape and reaction-diffusion space. *International Journal of Computer Vision*, 15:189–224, 1995.

[MBTF04] N. Molino, R. Bridson, J. Teran, and R. Fedkiw. Adaptive physics based tetrahedral mesh generation using level sets. (in review), 2004.

[MT03] M. Müller and M. Teschner. Volumetric meshes for real-time medical simulations. In *Proc. BVM (Bildverarbeitung für die Medizin)*, pages 279–283, Erlangen Germany, March 2003.

[NT03] Fakir S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, 2003.

[Ope04] OpenTissue, 2004. http://www.opentissue.org.

[PS04] Per-Olof Persson and Gilbert Strang. A simple mesh generator in matlab. *SIAM Review*, 46(2):329–345, June 2004.

[Set99] James A. Sethian. *Level Set Methods and Fast Marching Methods. Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999. Cambridge Monograph on Applied and Computational Mathematics.

[TBHPG03] M. Teschner, M. Müller B. Heidelberger, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling, Visualization*, pages 47–54, Munich, Germany, November 2003.

Direct Volume Rendering of Unstructured Grids in a PC based VR Environment

Paul Benölken
Fraunhofer IWU
Reichenhainer Straße 88
09126 Chemnitz

Holger Graf
Fraunhofer IGD
Fraunhofer Straße 5
64283 Darmstadt

paul.benoelken@t-online.de holger.graf@igd.fhg.de

ABSTRACT

In this paper we present our solution for fast, direct volume rendering of unstructured grids on standard PC workstations. We describe our modification of the incremental slicing approach for achieving high performance as well as the application of geometry-compression methods for minimizing the memory requirements. Furthermore, we show our implementation for the interactive modification of transfer functions (classification) in a virtual reality environment by using 3D interaction widgets. Finally we present and discuss the results, we achieved with our application in a VR environment.

Keywords
Direct Volume Rendering, Unstructured Grids, Geometry Compression, Virtual Reality

1. INTRODUCTION

Direct volume rendering of unstructured 3D scalar fields has been the subject of a number of research activities and publications over the past decade. Many efforts are directed towards improving the rendering performance by parallelization or efficiently using graphics hardware. Although great advances have been reported, many techniques are either restricted to handling fixed cell types and optical models or require massive parallel computing facilities for achieving sufficient frame rates. However, due to the increasing power and wide availability of consumer graphic boards, different volume rendering algorithms have been implemented on standard PC hardware. Nevertheless, the existing solutions still do not fully match the requirements of interactive applications like those required in virtual reality environments.
Beside interactive visualization, the classification of the scalar field by generating transfer functions plays an important role for the volume rendering process.

The Journal of WSCG, Vol.13, ISSN 1213-6964
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency – Science Press

A favourable transfer function exposes the important data structures and hides the non relevant parts of the dataset. Frequently used are opacity functions assigning colors and opacities based on scalar values. Many datasets contain complex structures with overlapping scalar values. Here the application of opacity functions frequently fails. Different techniques have been developed within the last years for visualizing more details by applying multidimensional transfer functions as well as for providing more convenient user interfaces. However the classification of unstructured volume data is not addressed.

2. Related Work

Besides different optimizations and parallelization for improving software solutions like ray castings [Lev90] or splatting techniques [Wes90], several research activities addressed the exploitation of the graphics hardware as in [WE97]. Cell projections like the projected tetrahedra (PT) method [ST90] have become one of the most popular methods for hardware accelerated volume rendering.

Different approaches have been considered for improving the image quality [SBM94], [GRS+02], [RE02] as well as for achieving higher frame rates by accelerating the required cell sorting [Wil92], [SMW98], [CKM+99]. Other hardware accelerated methods like [WKE02] avoid the sorting process by applying an emissive optical model and exploiting the vertex programming facilities of current graphic boards for implementing a view-independent cell projection. Nevertheless, besides tetrahedral elements, unstructured grids resulting from finite element computations frequently contain different polyhedral elements (e.g. prisms, hexahedra etc.). Hence applying the PT algorithm for such grid types requires a tetrahedral decomposition of each cell, which in turn increases the amount of cells to be processed and thus reduces the overall performance. Hence, the recently presented approaches extend the original PT algorithm for processing different kind of polyhedra. Roettger and Ertl [RE03] analyse the maximum performance of PC graphic accelerators like the NVIDIA GeForce series of cards and apply an emissive optical model for achieving fast renderings.

Although several accelerations and improvements were successfully applied as in [Mor04], the performance of these algorithms is still below the interactive frame rates achievable for regular grids. Hence alternative solutions presented by [Wes01] and [WE01] implement different strategies for re-sampling unstructured volume data onto cartesian grids in a preprocessing step. Interactive performance is gained by employing texture based volume rendering on the re-sampled dataset. Nevertheless, these solutions are constrained by the available texture memory on the graphic boards. Larger datasets or higher resolutions are processed by splitting the cartesian grids into multiple blocks (bricks). However, due to the limited bandwidth, data transfer from the CPU main memory to the GPU texture memory results in performance losses.

Another solution, proposed by Max et al. [MWSC03] implements cell projections (including cell sorting) and slicing methods. Both methods are parallelized for improving the performance and supplemented with anti-aliasing techniques for handling small cells.

Further research activities are focused on the definition and manipulations of transfer functions. Typical graphical user interfaces are restricted to opacity functions. Modification of the opacity functions are done by moving control points of the 2D graph. He et al. [HHKP96] apply genetic algorithms for finding good transfer functions. In an iterative process the user selects the desired mapping from an automatically generated population of small thumbnail renderings. Marks et al. [MAB+97] propose a Design Gallery as a visual interface to the space of possible transfer functions. An image difference metric is used for arranging different renderings, from which the user selects the most appealing one. A more data-centric approach was presented by Kindlmann et al. [KD98]. Their semi-automatic method exploits the relationship between data values and their first and second derivatives in direction of the gradient, for detecting material boundaries. Further approaches define the transfer functions based on the principal curvatures reconstructed from the input data [HKG00] or apply dynamic programming for a template based adjustment of transfer functions [RSHSG00]. Kniss et al. [KKC01] use a direct manipulation widget in a desktop environment for specifying multidimensional transfer functions based on data value, gradient magnitude, and the second directional derivative. Botha and Post [BP02] presented a fast method based on slice-based preview for finding an appropriate transfer function.

3. Interactive Direct Volume Rendering

As shown by Chopra and Meyer [CM02] fast visualizations are feasible by using incremental slicing. Performance improvements have been achieved by avoiding redundant computations and replacing the active cell and active edge list in the original algorithm of Yagel et al. [YRLN96] with an active region list. Inspired by the mentioned slicing method of Max et al., our solution finally skips the computation of the active region list which originally was required for each change of the view direction.

Two different modes have been implemented for computing the set of volume slices. In interactive mode for each of the three X, Y, Z axes a set of slices is computed in a preprocessing step with a predefined slice distance. Since artifacts might become visible from intermediate view points, we additionally employ an immediate mode which computes a set of slices from the current view direction when the rotary motion stops.

3.1 Computation of Volume Slices

Given an unstructured grid with N_n node coordinates $n_i \in \{n_0 \ldots n_{Nn-1}\}$ and N_c grid elements $c_i \in \{c_0 \ldots c_{Nc-1}\}$ the geometry and topology of each cell c_i is defined by a list of cell nodes $\{n_{ci}\}$, where each cell node refers to a specific node coordinate n_i.
Starting with a given distance Δ_z of slicing planes and a view direction $\mathbf{x}_{view}$ the number of required slices N_{slices} as well as the start and endpoints, z_{start} and z_{end} in object space are calculated. Afterwards, the slice computation proceeds in the following way:

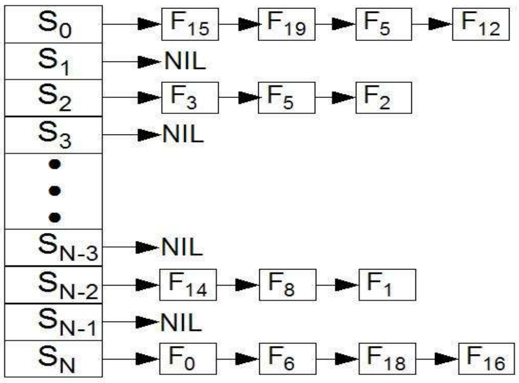
```
1.  for each node $n_i$
        compute and store the node distance $d_{ni}$ to the
        viewplane $\mathbf{p} = z_{end.}$
2.  for each cell $c_i$
        for each cell node $n_{ci}$
        a.) compute the slice index
            $s_i = d_{ni} / (z_{end} - z_{start}) N_{slices}$
        b.) determine the min/max slice index
            $s_{min}$ and $s_{max}$
        c.) for each slice index $s_i \in \{s_{min} \ldots s_{max}\}$
            compute and store the slice polygons $f_{si}$
```

In step 2c we applied a variant of the marching cube method for incrementally slicing the volumetric cells.

Each cell node is classified as “0” or “1” depending on the sign of the node distance d_{nci}. Since in step 1 the viewplane distance d_{ni} was calculated for each grid node, the required computations of the specific slice-point distances d_{nci} for each cell node n_{ci} simplify to

$$d_{nci} = d_{ni} + \Delta_z (s_i + 1).$$

The generated bit sequence is used as an index into a lookup table, which identifies the cell edge for the vertex interpolation of the slice polygon. Different lookup tables are maintained for identifying the specific (e.g. tetra or hexa) polyhedron intersections. Hence a decomposition into tetrahedral elements is not anymore required. Finally each polygon set of a specific slice is stored into an indexed list PL_N as illustrated in figure 1. Therefore the computation of active region lists is skipped.



S_0 → F_{15} → F_{19} → F_5 → F_{12}
S_1 → NIL
S_2 → F_3 → F_5 → F_2
S_3 → NIL
S_{N-3} → NIL
S_{N-2} → F_{14} → F_8 → F_1
S_{N-1} → NIL
S_N → F_0 → F_6 → F_{18} → F_{16}

Figure 1: Structure of the indexed polygon list for storing the slice polygons.

Besides the polygon data, the indexed list structure contains empty slices, which are indicated by a NIL-pointer, so that intermediate slices can be inserted or skipped.

3.2 Semi Adaptive Slicing

The computation of volume slices allows fast direct volume renderings of unstructured grids. However the optimal choice of the slice distance Δ_z is still a problem of such slicing techniques. Small details are missed, if the slice distance was chosen too large, whereas small values of Δ_z result into over sampling. According to Chopra [CM02], no details are missed, if the distance Δ_z is chosen to be neither greater nor equal to the smallest edge e_{min} of the dataset. However, this approach results into extreme large amounts of polygons, which are hardly processed in real-time. Hence Max et al. propose the usage of splatting methods or cell projections for all those cells, which have been missed by the slicing operations.

An alternative approach is the semi adaptive computation of volume slices. For this purpose, a set of equidistant slices is created using our previously described method. In this first step, all elements which have been successfully sliced, are marked as processed. Afterwards the number of slice regions is doubled and the described slicing procedure is applied to the remaining unprocessed cells with the adapted slice distance Δ_{zi}. This step is repeated until either all elements or a predefined percentage has been sliced. Within each iteration step, a separate indexed polygon list PL_{Ni} is built. Afterwards the polygon list of the previous step PL_{Ni-1} is completed with the new generated one. This operation is accomplished by simple pointer operations by exploiting the previously mentioned indexed list structure.

This approach enables the generation of slice stacks with varying resolutions in z-direction. Additional slices may be inserted or skipped in a background process. Hence different levels of detail could be dynamically generated according to the user defined scaling and zooming operations.

3.3 Interactive Classification

An efficient visual analysis of the volume dataset requires an interactive classification and hence direct manipulation of the transfer function. Beside an immediate update of the scalar field visualization, the representation of the transfer functions is another important factor for gaining insight into the dataset and its value distribution.

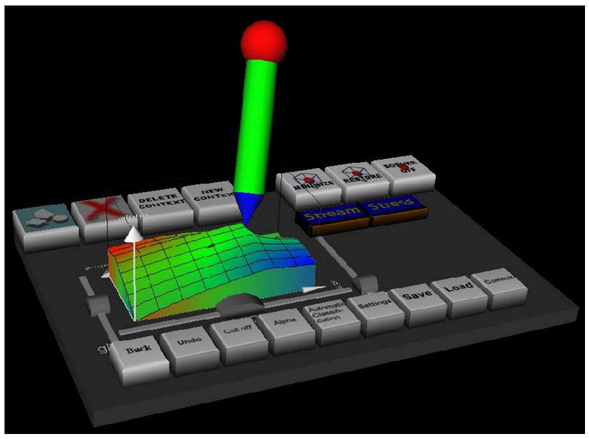


Figure 2: The interaction panel with the 3D classification widget and sliders for interactive manipulation of the transfer function.

Usually multi-dimensional transfer functions offer a higher flexibility for detecting and exposing characteristic structures. Nevertheless the creation of a meaningful representation for more than three independent parameters is not trivial. Therefore our implementation was restricted on 2D transfer-functions which appeared to be a good compromise between the flexibility of multi-dimensional transfer-functions and one-dimensional opacity assignments.

Since one of our aims was the integration into a VR environment, a 3D representation of the transfer function appeared to be a natural extension for 3D interactions and visualizations.

Figure 2 shows the interaction panel with the 3D classification widget and the sliders we implemented for visualizing and manipulating 2D transfer functions. The transfer function is drawn as a height field by mapping scalar values and gradient magnitudes onto the tablet plane, whereas the opacity (alpha) values define the height of the palette.

Direct manipulations are supported by using a 6DOF interaction pen for changing individual alpha values. Since this interaction is strongly influenced by the assignment of alpha and height values, we examined different mappings. Tests with linear mappings showed significant correlations for lower alpha values, whereas changes of higher alpha values are reflected by minor changes in the scalar field visualization. A better reflection of small alpha values was achieved by employing a logarithmic mapping with basis b using the following formula:

$$h = \ln((b-1)\,\alpha + 1) / \ln(b)$$

Where h is the computed height value and α indicates the corresponding transparency. The inverse function is used for obtaining the transparency from the user defined height value, which is:

$$\alpha = (b^h-1) / (b-1).$$

Indirect manipulations of the transfer function are accomplished by moving one of the 3D sliders on the interaction panel. Different operations have been implemented for changing the global transparency as well as for highlighting edges and material boundaries by adjusting gradient weights. Another operation supports the selection of specific regions by specifying mean and range of the gradient-value domain. Further interaction buttons have been integrated, for storing the current transfer function, for loading the settings from a previous session, as well as for performing undo and redo operations on the current transfer function. A list of transfer functions is used for tracking the changes within a session.
The Classification is accomplished by storing the transfer functions as 2D texture maps. The scalar values and gradient values are transferred to the implemented fragment program which applies texture mapping by taking the scalar values and gradient magnitudes as texture coordinates for determining the fragment color and finally performs the lighting calculations.

4. Our Solution

This chapter describes our realization of an interactive direct volume rendering system. It gives an overview of the architecture and some optimization strategies which are used in order to optimize the data handling on common PCs.

4.1 Architectural View

Figure 3 shows the client-server architecture of our application. Memory and CPU intensive tasks like the slicing of the grid and the computation of gradient values are performed by the server.

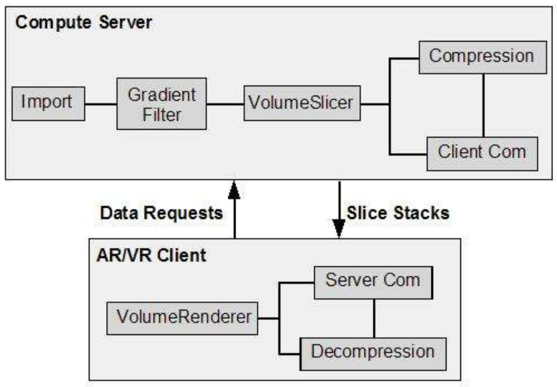


Figure 3. Client-Server architecture for direct volume rendering

The rendering client is in charge of requesting the processed data from the server and rendering the generated stack of polygon slices (stack processing) as well as for performing per fragment operations as lightning or texture reads using its Graphics Processing Unit or GPU. Furthermore, the rendering client handles the user interactions like changes in the view direction and manipulation of the transfer function.

An optional compression and decompression of the slice stacks can be applied for limiting the memory requirements and obtaining an efficient usage of the available bandwidth.

4.2 Geometry Compression

One of the requirements for the above mentioned method to interactively render unstructured grids is the storage of the computed polygon sets. Depending on the underlying grid resolution and the selected slice distance, the required storage space might easily exceed the capacities of common workstations.
Hence for limiting the memory requirements of our method, we integrated vector quantization and scalar normalization operations. In our implementation we used 16 Bit shorts for storing vertex coordinates and 8 Bit unsigned char values for each normal component and scalar values.

Thus the required memory was reduced by a factor of 2 compared to the original floating point size. Since the coding of the vertex positions consists of a translation and a scaling operation, the subsequent decoding is easily accomplished by the graphics hardware.

As already pointed out by Deering [Dee95], the resulting accuracy after decoding the compressed data, is usually sufficient for display purposes.

An additional reduction of the data size is achieved by transferring the individual slice polygons into indexed face sets, which are efficiently displayed using OpenGL vertex arrays. With this elimination of redundant vertex information and the mentioned quantization operation we achieved a reduction of the polygon slices by 25% of the original data size.

Further reductions have been achieved by integrating a compression scheme, which we successfully applied for arbitrary polygon sets from CAD models [BS02]. The algorithm encodes a polygonal mesh into a byte sequence which is finally compressed using Huffman encodings. We applied this method to the computed polygon slices and achieved a reduction of below 10 percent of the original size, on different slice sets.

5. Results

Our implementation was tested in our VR environment with different datasets and PC workstations. Figure 4 shows the interactive direct volume rendering of the space shuttle flow field in our VR environment.

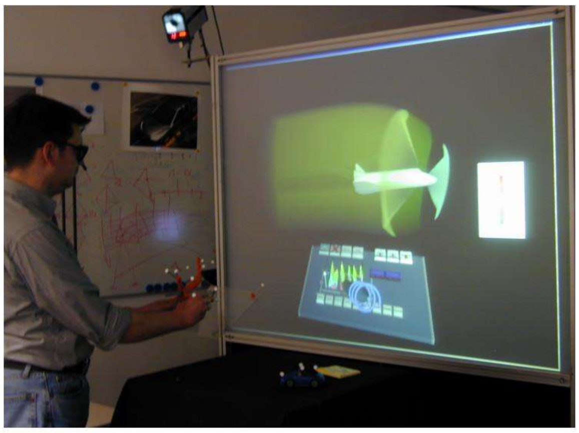


Figure 4: Interactive direct volume rendering the space shuttle flow field in a VR environment.

The classification of the volume data and hence the manipulation of the underlying transfer function is done by directly changing the 3D classification widget with the 6DOF interaction pen or by using the 3D menu elements on the interaction panel. The position and orientation of the dataset is controlled by the user by placing the tracked artifact object with its tracking sensors in the VR environment.

The evaluation of our method was done using different datasets with different sizes and cell elements. The shuttle dataset shown in figure 4 consists of 226800 nodes and 215512 hexaedron-elements. Figure 5 shows the result from a simulation of a polymere injection of a single nozzle, which was displayed using our direct volume rendering method. .

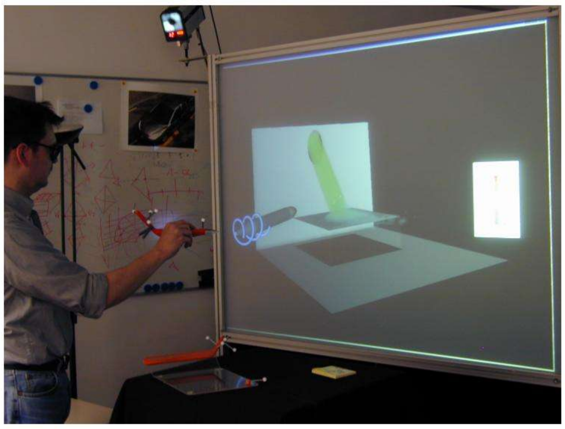


Figure 5: Direct volume rendering of a polymer injection from a single nozzle

The datasets consists of 87754 nodes, 223936 tetra elements and 26900 hexahedrons. Figure 6 shows the result of our direct volume rendering method with the bluntfin dataset. The dataset consists of 40960 nodes and 37479 hexaedron elements.

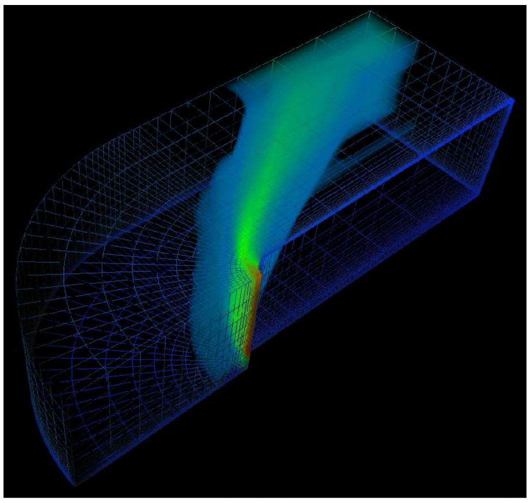


Figure 6: Direct volume rendering of the bluntfin dataset.

Some benchmarks have been driven in order to validate our approach. Figure 7 shows the timings of the slicing and quantization operation, which we achieved for the used datasets with different slice numbers.

The measurements were performed on a mobile PC equipped with a 3,06 GHz Intel CPU, 512MB main memory and a Windows XP operating system.

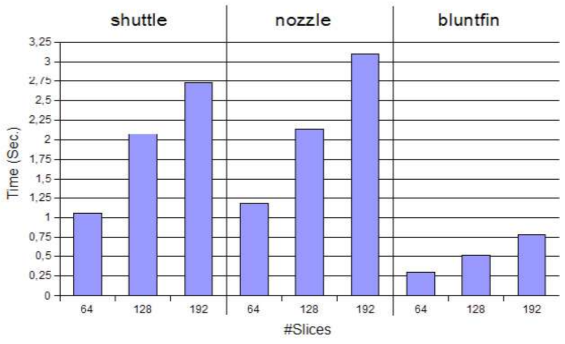


Figure 7: Timings for slicing the datasets with different slice numbers.

The performance of the slicing and quantization operation is strongly related to the size of the dataset and the number volume slices, as shown in the diagram below. The computation of detailed slice sets is done very quickly for small datasets whereas the calculation for large datasets requires a delay of up to three seconds. However, after the slice computation is completed, the display and blending of the slice stacks is performed by the graphics hardware.

Figure 8 summarizes the framerates we achieved for rendering and lighting the mentioned datasets using axis aligned slices with different slice distances and hence polygon numbers. The measurements have been carried out on a windows PC equipped with a 2,8 GHz Intel CPU, 1GB main memory and an NVIDIA GeForce FX 5900 graphics board with 128MB video memory.

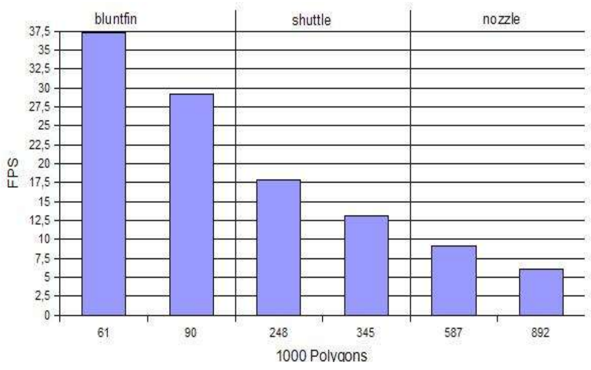


Figure 8: Framerates for rendering and lighting axis aligned slices with different polygon numbers using a view port of 1024 × 768.

The diagram shows the strong dependency of the achieved display performance from the number of polygons which have to be processed by the graphics hardware. As expected, small polygon sets are displayed with high framerates, wheras the display time slows down with the increasing number of polygons such that the limits of these graphicscards will be at approx. 1 million polygons.

Hence the performance of the presented approach is predominantly defined by the rastering and blending capabilities of the graphics hardware.

6. Conclusion

Our solution allows fast direct volume renderings of unstructured grids on a standard PC platform. The slicing algorithm quickly generates polygon slices for small datasets. Larger datasets can be processed by generating axis aligned slice sets in a preprocessing step. Alternatively slice sets with a reduced z-resolution might be used for previewing during the user interaction, while additional slices are computed in a background process.

As shown before the methods processes unstructured grids with mixed tetra- and hexaedral elements and is easily extended for handling further elements. A tetrahedral decomposition is not anymore required so that the amount of cell elements to be processed is strongly reduced which in turn leads to faster computations. The implemented compression methods additionally reduce the memory requirements so that the available bandwidth is more efficiently used.

Moreover, the method allows an improved balancing of the workload between GPU and CPU and hence an improved overall performance. In contrast to this the load of view-independent tetra-projections is predominantly on the GPU. Furthermore the semi-adaptive slicing approach offers an efficient alternative for displaying small grid cells without generating inadequate amounts of polygons.
Finally the created modules support the interactive visualization and classification of volume data in a VR environment. The developed 3D interaction facilities enable a direct and convenient manipulation of the transfer function as well as an intuitive navigation in virtual environments.

7. Outlook

The presented work currently allows the presentation of static volume data. Further work should concentrate on the inclusion of dynamic and time variant simulation data. With some modifications this method can be used for colourised and opacity animated renderings of scalar fields, which values change over time. Using a time invariant geometry the interpolation weights resulting from the slicing operations as well as the indices of affected grid nodes can be stored in addition to the vertex data. During the following time steps only the indexed nodes and the interpolation weights have to be transferred to the GPU which eventually performs the interpolation via a vertex program.

REFERENCES

[BS02] P. Benölken and A. Stork. Geometry compression for collaborative CAD applications. In Ralph H. u.a. Stelzer, editor, *CAD 2002.* Proceedings : Corporate Engineering Research, pages 121-129.

[BP02] C.P. Botha and F.H. Post. New technique for transfer function speci-fication in direct volume rendering using real-time visual feedback. In Proc. of the SPIE Int. Symposium on Medical Imaging, volume 4681, 2002.

[CKM+99] J. Comba, J. T. Klosowski, N.L. Max, J.S.B. Mitchell, C.T. Silva, and P.L. Williams. Fast polyhedral cell sorting interactive rendering of unstructured grids. In Computer Graphics Forum (Proc. Eurographics '99), volume 18, pages 369–376, 1999.

[CM02] Prashant Chopra and Joerg Meyer. Incremental slicing revisited: Accelerated volume rendering of unstructured meshes. In Proceedings of IASTED Visualization, Imaging, and Image Processing 2002, pages 533–538, Sept. 9-12 2002.

[Dee95] Deering, M 1995. Geometry Compression Proceedings SIGGRAPH, 1995.

[GRS+02] Stefan Guthe, Stefan Roettger, Andreas Schieber, Wolfgang Straßer, and Thomas Ertl. High-quality unstructured volume rendering on the pc platform. In ACM Siggraph/Eurographics Hardware Workshop, 2002.

[HHKP96] T. He, L. Hong, A. Kaufman, and H. Pfister. Generation of transfer functions with stochastic search techniques. In Proceedings IEEE Visualiztaion, pages 227–234. IEEE, 1996.

[HKG00] J. Hladuvka, A. König, and E. Gröller. Curvature-based transfer functions for direct volume rendering. In Spring Conference on Computer Graphics (SCCG 2000), pages 58–65, 2000.

[KD98] G. Kindlmann and J.W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In IEEE Symposium On Volume Visualization, pages 79–86. IEEE, 1998.

[KKC01] J. Kniss, G. Kindlmann, and C.Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In Proceedings IEEE Visualization 2001, pages 255–262. IEEE, 2001.

[Lev90] M. Levoy. E±cient ray tracing of volume data. *ACM Transactions on Graphics*, 9 (3): 245-261, July 1990.

[MAB+97] J. Marks, B. Andalman, P.A. Beardsley, H. Pfister, et al. Design galleries: A general approach to setting parameters for computer graphics and animation. In ACM Computer Graphics (SIGGRAPH '97 Proceedings), pages 389–400. ACM, August 1997.

[Mor04] Kenneth Dean Moreland. Fast High Accuracy Volume Rendering. Dissertation, The University of New Mexico, 2004.

[MWSC03] Nelson Max, Peter Williams, Claudio Silva, and Richard Cook. Volume rendering for curvilinear and unstructured grids. In Computer Graphics International, 2003.

[RE02] S. Roettger and T. Ertl. A two-step approach for interactive preintegrated volume rendering of unstructured grids. In Proceedings of IEEE Volume Visualization and Graphics Symposium 2002, pages 23–28, October 2002.

[RE03] S. Roettger and T. Ertl. Cell projection of convex polyhedra. In Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics, page 103108. ACM, 2003.

[RSHSG00] C. Rezk-Salama, P. Hastreiter, J. Scherer, and G. Greiner. Automatic adjustment of transfer functions for 3d volume visualization. In Proc. Vision, Modelling, and Visualization (VMV), pages 357–364, 2000.

[SBM94] C. M. Stein, B. G. Becker, and N. L. Max. Sorting and hardware assisted rendering rendering for volume visualization. In Symposium on Volume Visualization, pages 83–89, 1994.

[SMW98] C. Silva, J. Mitchell, and P. Williams. An interactive time visibility ordering algorithm for cell complexes. In ACM / IEEE Symposium on Volume Visualization, pages 15–22, 1998.

[ST90] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. In Computer Graphics (San Diego Workshop on Volume Visualization), volume 24, pages 63–70, 1990.

[WE97] R. Westermann and T. Ertl. Rendering and re-sampling unstructured volume data by polygon drawing. Technical Report 17, Universität Erlangen-Nürnberg, 1997.

[WE01] Manfred Weiler and Thomas Ertl. Hardware-software-balanced resampling for the interactive visualization of unstructured grids. In Proceedings of the conference on Visualization '01, pages 199 – 206. IEEE Computer Society, 2001.

[Wes90] L. Westover. Footprint evaluation for volume rendering. In *Computer Graphics*, volume 24, page 367-376, 1990.

[Wes01] R. Westermann. The rendering of unstructured grids revisited. In Eurographics/ IEEE Symposium on Visualization 2001, pages 65–74, 2001.

[Wil92] P. Williams. Visibility ordering meshed polyhedra. In ACM Transaction on Graphics, volume 11, pages 103–126, 1992.

[WKE02] Manfred Weiler, Martin Kraus, and Thomas Ertl. Hardware-based view-independent cell projection. In *IEEE Symposium on Volume Visualization*, pages 13 - 22, 2002.

[YRLN96] R. Yagel, D. Reead, P. Law, A. Shihh, and Shareef N. Hardware assisted volume rendering of unsructured grids by incremental slicing. In *1996 Volume Visualization Symposium*, pages 55-62. IEEE Computer Society Press, 1996.

Denoising 2-D Vector Fields by Vector Wavelet Thresholding

Michel A. Westenberg and Thomas Ertl
Institute for Visualization and Interactive Systems, University of Stuttgart
Universitätsstr. 38
70569 Stuttgart, Germany
{westenberg, ertl}@vis.uni-stuttgart.de

ABSTRACT

Noise reduction is an important preprocessing step for many visualization techniques that make use of feature extraction. We propose a method for denoising 2-D vector fields that are corrupted by additive noise. The method is based on the vector wavelet transform, which transforms a vector input signal to wavelet coefficients that are also vectors. We introduce modifications to scalar wavelet coefficient thresholding for dealing with vector-valued coefficients. We compare our wavelet-based denoising method with Gaussian filtering, and test the effect of these methods on the signal-to-noise ratio (SNR) of the vector fields before and after denoising. We also compare our method with component-wise scalar wavelet thresholding. Furthermore, we use a vortex measure to study the performances of the methods for retaining relevant details for visualization. The results show that for very low SNR, Gaussian filtering with large kernels has a slightly better performance than the wavelet-based method in terms of SNR. For larger SNR, the wavelet-based method outperforms Gaussian filtering, because Gaussian filtering removes small details that are preserved by the wavelet-based method. Component-wise denoising has a lower performance than our method.

Keywords

Denoising, flow visualization, multiwavelets, wavelets.

1. INTRODUCTION

Data acquired by physical measurements are often corrupted by noise. In fluid mechanics, such data may be obtained by, for instance, particle image velocimetry (PIV). This is a technique that provides global velocity measurements by recording the position over time of small tracer particles inserted into the flow [Pra00]. Noise in the recorded images is a source of errors in PIV measurements, and it can result in spurious vectors or global noise in the reconstructed vector field. The spurious vectors can be repaired by averaging or median filtering, however, the global noise requires a different removal method.

The Journal of WSCG, Vol.13, ISSN 1213-6964
WSCG 2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency Science Press

The process of removing noise is called denoising, and its goal is to suppress the noise while retaining the relevant details. A commonly used denoising method is smoothing by Gaussian filtering. However, this does not only affect the noise, but also may destroy small features in the data.

Better performance is usually obtained by a smoothing technique that is edge-preserving, such as anisotropic diffusion [Per90]. This technique has been extended for smoothing orientation fields [Per98], but it has not been tested in a practical application, and has not been evaluated on directional fields. Another successful iterative method for image denoising is based on minimizing the total variation of the image subject to constraints that involve the noise statistics [Rud92]. This approach has been extended to vector-valued functions, and has been used for denoising color images [Blo98]. In a recent paper, this method was used for the reconstruction of flow velocity images acquired by magnetic resonance velocity imaging [Ng03]. Such images are used in the study of cardiovascular function by analyzing the blood flow patterns and their interaction with cardiovascular structure. Noise has detrimental effects on this analysis, and it is very important that features in the data are retained by the denoising method.

Another class of denoising methods is based on thresholding of wavelet coefficients, an idea introduced about one decade ago by Donoho [Don95]. Since then, much work has been done in this area, and many wavelet-based denoising methods have been proposed for scalar signals [Str01], natural images [Cha00, Sim96], and medical images [Piž03, Win04], to name a few.

The purpose of this paper is to report on work in progress on denoising 2-D vector data that are corrupted by additive noise. Our method performs thresholding on wavelet coefficients that are obtained by a so-called vector wavelet transform [Xia96]. This is an extension of the scalar wavelet transform that deals with vector data, and it maps vector data to wavelet coefficients that are also vectors. It is important to note that the vector wavelet transform is different from a component-wise scalar wavelet transform, and that the mathematical foundation is based on multiwavelets. We introduce extensions to the scalar wavelet-based denoising technique, in order to be able to deal with the vector-valued coefficients.

The organization of this paper is as follows. Section 2 discusses the mathematical background of vector wavelets, and describes the algorithm to compute the vector wavelet transform efficiently. In Section 3 we briefly describe wavelet-based denoising of scalar data, and we introduce our modifications for dealing with vector data. Section 4 compares the results of vector wavelet-based denoising and Gaussian smoothing, and we perform an experiment with component-based scalar wavelet denoising. Finally, we draw conclusions in Section 5 and discuss future work.

2. VECTOR WAVELETS

The concept of a vector wavelet transform has existed for about a decade, and the theory follows scalar wavelet theory closely [Xia96]. Vector wavelet transforms are based on so-called multiwavelets, which expand a scalar function by several scaling functions and wavelet functions rather than by a single pair. In the following, we briefly describe multiwavelets, and we refer the readers to the papers [Tan99] and [Xia96] for full details.

2.1 Multiwavelets

A biorthogonal multiwavelet basis consists of a multiscaling function vector $\Phi(t) := [\phi_1(t), \ldots, \phi_r(t)]^\mathrm{T}$ and its dual $\widetilde{\Phi}(t) := [\widetilde{\phi}_1(t), \ldots, \widetilde{\phi}_r(t)]^\mathrm{T}$, with r an integer, and x^T denoting the transpose of x. Typically, $r = 2$ or $r = 3$ in practical applications with 2-D and 3-D vector fields, respectively. These multiscaling functions satisfy the two-scale dilation equations

$$\begin{aligned}\Phi(t) &= \sqrt{2}\sum_n H_n \Phi(2t-n),\\ \widetilde{\Phi}(t) &= \sqrt{2}\sum_n \widetilde{H}_n \widetilde{\Phi}(2t-n),\end{aligned} \tag{1}$$

in which H_n and $\widetilde{H}_n$ are real-valued $r \times r$ matrix sequences. The multiwavelet functions $\Psi(t)$ and $\widetilde{\Psi}(t)$ are associated with the multiscaling functions by the two-scale wavelet equations

$$\begin{aligned}\Psi(t) &= \sqrt{2}\sum_n G_n \Phi(2t-n),\\ \widetilde{\Psi}(t) &= \sqrt{2}\sum_n \widetilde{G}_n \widetilde{\Phi}(2t-n),\end{aligned} \tag{2}$$

in which G_n and $\widetilde{G}_n$ are also real-valued $r \times r$ matrix sequences.

The expansion of an input vector signal $f^\mathrm{T}(t)$ on a biorthogonal multiwavelet basis is given by

$$\begin{aligned}f^\mathrm{T}(t) = \sum_k (c_k^M)^\mathrm{T} \Phi_{M,k}(t) &+ \sum_{j=1}^{M}\sum_k (d_k^j)^\mathrm{T} \Psi_{j,k}(t),\\ \Phi_{j,k}(t) &= 2^{-j/2}\Phi(2^{-j}t-k),\\ \Psi_{j,k}(t) &= 2^{-j/2}\Psi(2^{-j}t-k),\end{aligned} \tag{3}$$

where M denotes the depth of the decomposition. The coefficients c_k^M and d_k^j are called approximation coefficients and detail coefficients, respectively, as in the scalar case. Note that these coefficients are now $r \times 1$ column vectors.

2.2 Fast vector wavelet transform

Given coefficient sequences H_n, G_n, $\widetilde{H}_n$, and $\widetilde{G}_n$ that are $r \times r$ matrices, and which satisfy the perfect reconstruction conditions, we can compute the 1-D discrete vector wavelet transform of the input sequence c^0 by the pyramid algorithm of Mallat. The main difference with the scalar algorithm is that scalar multiplications are replaced by matrix-vector multiplications. The M-level wavelet decomposition computes the coefficients c_k^j and d_k^j as

$$c_k^j = \sum_n \widetilde{H}_{n-2k} c_n^{j-1} \qquad d_k^j = \sum_n \widetilde{G}_{n-2k} c_n^{j-1}. \tag{4}$$

Reconstruction is computed as

$$c_k^{j-1} = \sum_n H_{k-2n}^\mathrm{T} c_n^j + \sum_n G_{k-2n}^\mathrm{T} d_n^j. \tag{5}$$

The extension to a 2-D transform is done in the standard way by applying the 1-D transform to the rows

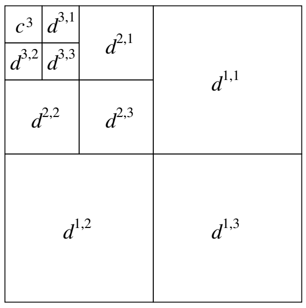


Figure 1. Coefficients of a three level 2-D (vector) wavelet transform.

and columns. The wavelet transform for M levels then results in approximation coefficients $c^M_{k,l}$ and three sets of detail coefficients $d^{j,\tau}_{k,l}$, $j = 1,\ldots,M$, $\tau = \{1,2,3\}$. The coefficients are ordered as shown in Fig. 1.

2.3 Filter coefficients

In principle, the filter coefficients of the multiwavelets available from the literature could be used for computing the vector wavelet transform. However, it turns out that the performance for vector signal processing applications is poor [Fow02]. The source of the problem lies in the fact that constant input signals are not preserved when performing a reconstruction from wavelet approximation coefficients only. Constant in the context of vector fields means that all vectors point in the same direction. Intuitively, one would expect a constant signal, however, most multiwavelets result in an oscillatory distortion. This means that the coefficients $c^M_{k,l}$ do not consist of a low resolution approximation of the original data. This is rather disturbing, as most denoising and compression schemes preserve the approximation coefficients and discard detail coefficients.

Fowler and Hua [Fow02] have proposed a scheme to design filter coefficients that define a multiwavelet basis that does not suffer from the problem mentioned above. The resulting wavelets are known by the name omnidirectionally balanced symmetric-antisymmetric (OBSA); part of this name refers to the constraints formulated for the construction process. In the remainder of this paper, we will use the OBSA 5-3 and OBSA 7-5 filters. The numbers denote the lengths of the coefficient sequences H_n and $\widetilde{H}_n$, respectively.

3. WAVELET-BASED DENOISING

We assume that the noise is *additive*, and has a normal distribution with zero mean and variance σ_n^2, denoted as $N(0,\sigma_n^2)$. Wavelet-based denoising methods in the 1-D scalar case then work in three steps. (1) Compute an M-level wavelet transform. (2) Modify the detail coefficients d^j_k, $j = 1,\ldots,M$, by a threshold function. The approximation coefficients c^M_k are not modified. (3) Compute the inverse wavelet transform. The extension to higher dimensions is straightforward.

There are two popular threshold functions in use: hard and soft thresholding. Both set the coefficients below the threshold T to zero. Hard thresholding retains the coefficients above the threshold unaltered. Soft thresholding, also called shrinkage, reduces the amplitude of the coefficients above T as follows

$$\eta_T(x) = \operatorname{sgn}(x)\cdot\max(|x|-T,0). \qquad (6)$$

For image denoising, soft thresholding generally yields more visually pleasing results than hard thresholding, and it is therefore the preferred choice.

Many methods have been proposed to select a good threshold T, a number of which are contained in the WaveLab software [Buc95]. In this paper, we use a method called BayesShrink [Cha00], which computes a data-driven estimate of T for each set of detail coefficients $d^{j,\tau}_{k,l}$, $\tau = \{1,2,3\}$ independently. This method was proposed for image denoising, and it is based on the observation that the detail coefficients in a subband of a natural image can be characterized by a generalized Gaussian distribution (GGD) [Mal89, Sim96]. The probability density function is given by

$$p(x) = \left[\frac{\nu\eta(\nu,\sigma)}{2\Gamma(1/\nu)}\right] e^{-[\eta(\nu,\sigma)|x|]^\nu}, \qquad (7)$$

with

$$\eta(\nu,\sigma) = \frac{1}{\sigma}\sqrt{\frac{\Gamma(3/\nu)}{\Gamma(1/\nu)}}, \qquad (8)$$

where $\Gamma(x)$ denotes the gamma function. The shape parameter ν controls the exponential rate of decay. A Gaussian distribution is obtained by $\nu = 2$. The parameter σ is the standard deviation.

We have observed that the individual components of the vector detail coefficients also follow a GGD. Figure 2 shows parts of the histograms of the second-level vector detail coefficients $d^{2,1}_{k,l}$, $d^{2,2}_{k,l}$, and $d^{2,3}_{k,l}$ of a slice of a hurricane data set as an example. The top row shows the histograms of the first components of the vectors, and the bottom row shows the histograms of the second components. All these histograms can be qualitatively described by a GGD. It is therefore valid to use the BayesShrink method.

We can now describe our modifications to the scalar wavelet-based denoising scheme for dealing with vector data. Calculations that involve the absolute value of a scalar coefficient now use the vector magnitude of

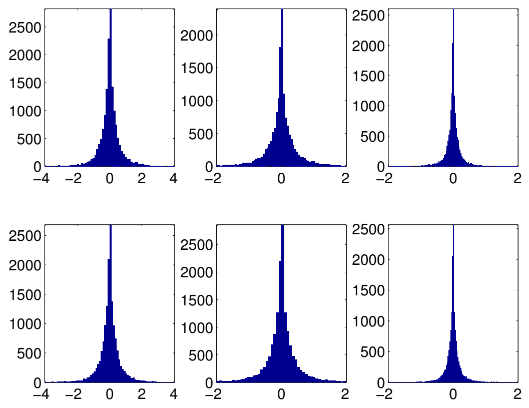


Figure 2. Histograms of vector wavelet detail coefficients at level 2 of an example data set. Histograms of the first vector components are on the top row and of the second components on the bottom row. From left to right are $d^{2,1}_{k,l}$, $d^{2,2}_{k,l}$, and $d^{2,3}_{k,l}$, respectively. All histograms can be described qualitatively by a generalized Gaussian distribution.

that coefficient. Furthermore, we define the variance σ^2 of an $N \times N$ vector field $v_{k,l}$ as

$$\sigma^2 = \frac{1}{rN^2} \sum_{k=1}^{N} \sum_{l=1}^{N} ||v_{k,l} - \bar{v}||^2, \qquad (9)$$

where the average $\bar{v}$ is a vector that contains the component-wise averages of $v_{k,l}$, and $|| \cdot ||$ denotes the Euclidian norm. This definition includes a division by r, the number of components of the vectors, for the following reason. If a vector field contains only noise, i.e., each component contains noise distributed as normal $N(0, \sigma^2)$, the equation above will yield precisely σ^2.

The threshold is dependent on the variance $\hat{\sigma}_d^2$ of the coefficients $d^{j,\tau}_{k,l}$ under consideration and the global noise variance σ^2. If the noise characteristics of the data acquisition process are known, it may be possible to determine the global noise variance from that information. Alternatively, the global noise variance can be estimated from the detail coefficients $d^{1,3}_{k,l}$ by the robust median estimator [Cha00]:

$$\hat{\sigma} = \frac{\text{median}(|d^{1,3}_{k,l}|)}{0.6745}. \qquad (10)$$

Finally, the threshold T is computed as

$$T = \frac{\hat{\sigma}^2}{\sqrt{\max(\hat{\sigma}_d^2 - \hat{\sigma}^2, 0)}}. \qquad (11)$$

If the denominator in this equation becomes equal to zero, the threshold T becomes ∞, and all coefficients are assigned the zero vector.

For our method, we adapted the soft thresholding method such that it shrinks the vector magnitudes. We define the modified soft thresholding $\vec{\eta}_T(x)$ for a vector x as

$$\vec{\eta}_T(x) = x \cdot \frac{\max(|x| - T, 0)}{|x|}. \qquad (12)$$

When $|x| = 0$, we set $\vec{\eta}_T(x) = [0, 0]^{\mathrm{T}}$.

4. RESULTS

We conducted a series of experiments in which noise of known standard deviation was added to a slice (490×490) of a hurricane data set, consisting of 2-component velocity vectors, see Fig. 3(a). The resulting noisy vector fields had signal-to-noise ratios (SNR) of $\{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$. An example rendering of the vector magnitudes of a noisy vector field with SNR $= 10$ is shown in Fig. 3(b). The SNR is expressed in dB and computed from the standard deviations σ (data) and σ_n (noise) as

$$\text{SNR} = 20 \log_{10} \frac{\sigma}{\sigma_n}.$$

To provide some intuition, an SNR around 40 dB is considered acceptable in image processing.

We applied our wavelet-based denoising method to the resulting noisy vector fields, using the biorthogonal OBSA 5-3 and OBSA 7-5 multiwavelets. The depth of the wavelet decomposition was fixed to three. We also performed filtering with Gaussian kernels of various widths. The width of the Gaussian kernel is described by its width in pixels at half of the maximum of the height of the Gaussian, a measure called Full Width at Half Maximum (FWHM). For example, a Gaussian filter with FWHM $= 5$ contains 13 pixels when sampled between -3σ and 3σ. Filter values beyond 3σ are negligibly small, and are therefore not used.

Example renderings of the vector magnitudes of the results of both Gaussian filtering and our method are shown in Fig. 3(c) and Fig. 3(d), respectively. The noisy input vector data had SNR $= 10$ (Fig. 3(b)), a high noise level at which the standard deviation of the noise is about one-third the standard deviation of the data. Qualitatively, both output images look similar, although the Gaussian filtered data appears to be more smooth, due to the large filter kernel used. The performance of the methods is comparable, as they both yield similar output signal-to-noise ratios.

Figure 4 shows the output SNR plotted against the input SNR. The plot shows that Gaussian filtering with large kernels performs slightly better than the wavelet-based method for very low SNRs. For an SNR between 15 and 20 dB, both methods show similar performance. For larger SNRs, the Gaussian filtering

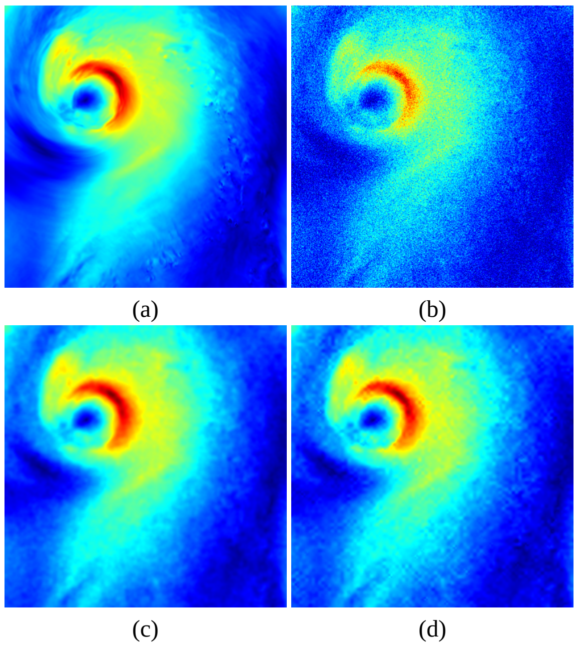


Figure 3. The images show color-encoded vector magnitudes; red corresponds to high velocities and dark blue to low velocities. (a) Noise-free test data. (b) Noisy test data with SNR $= 10$**. (c) Result after denoising by Gaussian filtering with FWHM 5. The filtered data has SNR** $= 23.4$**. (d) Result after denoising by wavelet coefficient thresholding. The resulting data has SNR** $= 22.3$**.**

method smooths to strongly, and for SNRs above 30 dB, the output SNR is actually lower than the input SNR. The wavelet-based method does not have this problem, and the output SNR is in the worst case equal to the input SNR. We also performed the experiment (results not included) with the OBSA 5-3 wavelet, and its performance is similar to the performance of the OBSA 7-5 wavelet. However, the performance for low SNR is worse, which can be explained by the fact that the OBSA 5-3 wavelet is not as smooth as the OBSA 7-5 wavelet.

For comparison, we implemented component-based scalar wavelet-based denoising, i.e. we treated each component of the vector field as a scalar data set, and applied scalar denoising. We used a fourth-order B-spline wavelet [Chu92] as a basic wavelet. It is clear that component-wise denoising has a consistently lower performance than a vector-based approach for the wavelets we tested, see Fig. 4. We presume this is due to possibility of changing the orientation of a vector when its components are thresholded independently. A more extensive investigation is necessary to see if this is indeed the cause of the lower performance.

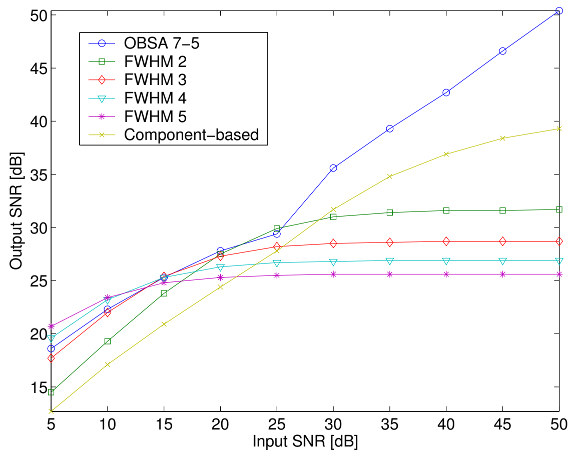


Figure 4. The output SNR plotted against the input SNR of wavelet-based denoising (OBSA 7-5) and Gaussian filtering (FWHM) with filters of increasing width. Also plotted is the performance of scalar wavelet-based denoising of the individual vector components independently of each other.

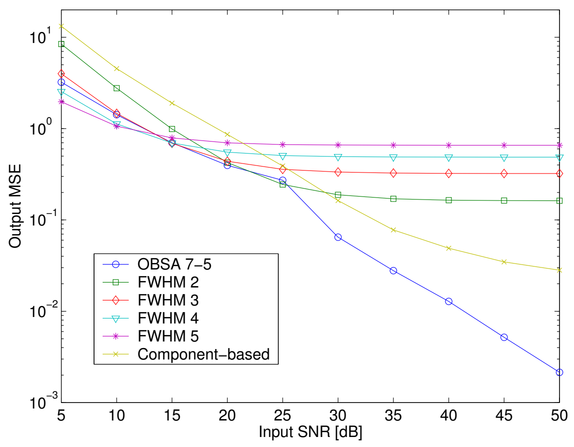


Figure 5. Output MSE plotted against the input SNR of wavelet-based denoising (OBSA 7-5) and Gaussian filtering (FWHM) with filters of increasing width. Also plotted is the performance of scalar wavelet-based denoising of the individual vector components independently of each other.

We also computed the mean square errors (MSE) between the original data and the denoised data, and the results are shown in Fig. 5. The vertical axis is on a logarithmic scale. The plot confirms that Gaussian filtering smooths too much when the noise level is low, which results in an MSE that is almost two orders of magnitude larger in comparison with our method.

Although the SNR is a good measure for the overall performance, it is not suitable to measure how well local features are retained. A problem, however, is that

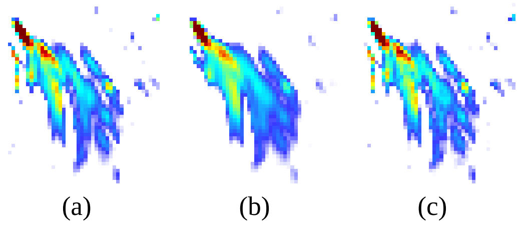


Figure 7. Detail images of a larger coherent feature in the data, selected from the larger structures in the upper left quadrants of the images in the third row of Fig. 6. (a) Noise-free data. (b) Gaussian filtering. (c) Wavelet-based denoising. Note how the small vertical structure on the left disappears with Gaussian filtering.

we do not actually have suitable quantitative measures, therefore, we render an image of the feature of interest, and make a visual assessment of the performance. Our feature of interest is a measure of vorticity, commonly referred to as the λ_2-definition [Jeo95]. The method computes the eigenvalues λ_1, λ_2, and λ_3, $\lambda_1 \geq \lambda_2 \geq \lambda_3$, of the matrix

$$M = \left[\frac{J+J^{\mathrm{T}}}{2}\right]^2 + \left[\frac{J-J^{\mathrm{T}}}{2}\right]^2. \qquad (13)$$

Here, J is the velocity gradient tensor. Vortex cores are defined as the points where λ_2 is negative.

Figure 6 shows color-encoded (blue to red) λ_2 values in a selected range for some of the generated noisy vector fields (left column), and the results of denoising these data sets by Gaussian filtering and our method. The middle column shows the best results obtained by Gaussian filtering, and the right column shows the results of our method using the OBSA 7-5 multiwavelets. The SNR is displayed below each image, as well as the filter size of the Gaussian kernel, and the percentage of wavelet coefficients that remain after thresholding. These percentages are indicative of the power of wavelets to capture relevant features with only a small number of coefficients.

For the high SNR input (almost noise free), Gaussian filtering misses details, especially in the areas with fine detail. An example of loss of detail is shown in Fig. 7, in which a small vertical structure is visible in the original data (Fig. 7(a)), which is lost by Gaussian filtering (Fig. 7(b)), but retained by our wavelet-based method (Fig. 7(c)).

We have seen that for high noise levels, Gaussian filtering performs better, because stronger low-pass filtering is needed. However, this is also possible to perform with our method. It appears that the threshold selection process underestimates the noise level,

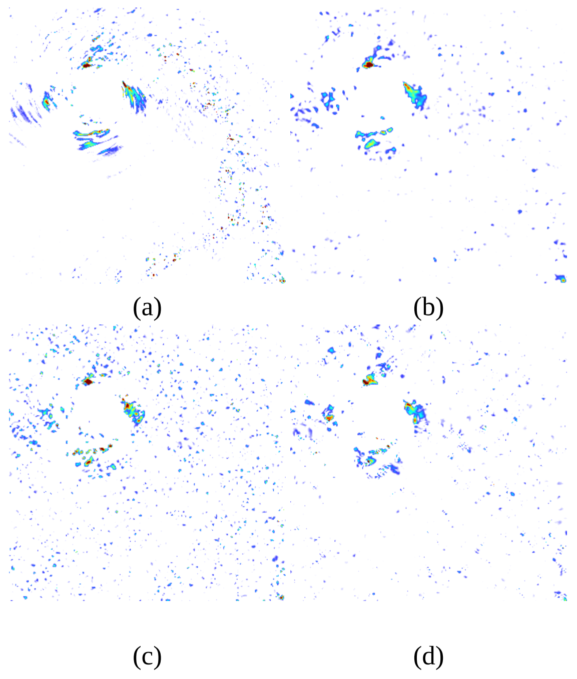


Figure 8. Denoising of the noisy vector data with SNR $=10$. All images show color-encoded λ_2 values in a selected range. (a) Rendering of the noise-free data. (b) Result of Gaussian filtering with FWHM 5. (c) Wavelet-based denoising with automatic threshold selection. The threshold is such that 5% of the largest detail coefficients remain after thresholding. (d) Wavelet-based denoising with the threshold lowered to a value such that only 2% of the largest detail coefficients are retained.

and that a lower threshold value is necessary. We performed a simple experiment with the noisy vector data with SNR $=10$ to see if it is possible to improve the output of our method, and the results are shown in Fig. 8. The λ_2 values of the noise-free data are shown in Fig. 8(a). We repeat the results of Gaussian filtering and our method in Fig. 8(b) and Fig. 8(c), respectively. Our method retains about 5% of the largest detail coefficients. When we lower the threshold such that only 2% of the largest coefficients are retained, we obtain the image shown in Fig. 8(d). The SNR improves only slightly to SNR $=22.5$, but the visual appearance of the features is much improved, and we also see a reduction of artifacts, i.e., features introduced that were not in the original noise-free data. Although this shows that it is possible to obtain a more ‘smooth’ result with our method, the problem is that this approach introduces a parameter (the number of coefficients to retain) in the method.

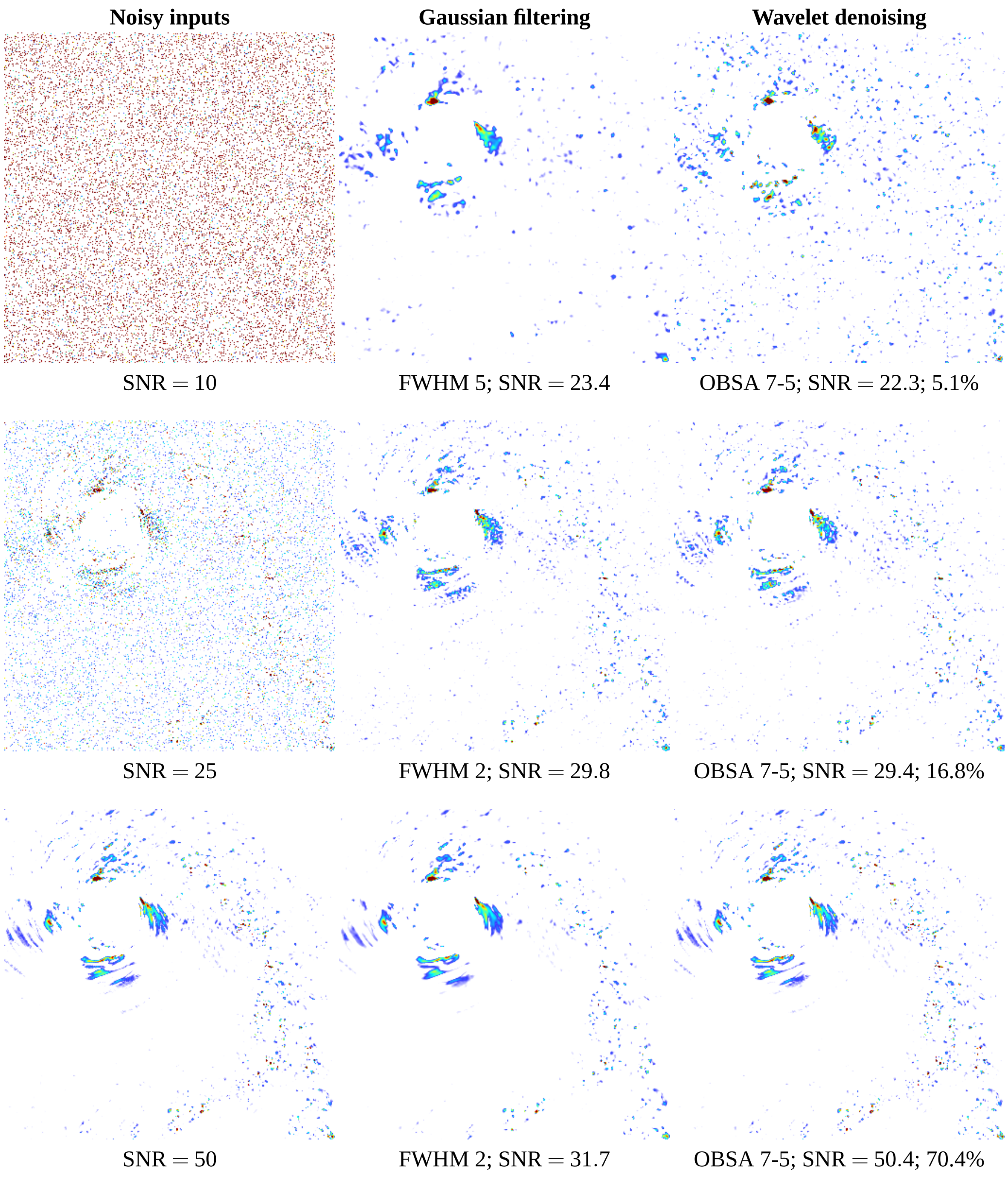


Figure 6. Results of denoising using Gaussian filtering and wavelet-based denoising. All images show color-encoded λ_2 values in a selected range. Left column: noisy input data of various signal-to-noise ratios. Middle column: results of Gaussian filtering using the filter with the best performance. Right column: wavelet-based denoising with the OBSA 7-5 multiwavelets. The depth of the wavelet decomposition was fixed at three levels. The resulting SNR after denoising is shown below the images. Additionally, the right column shows the percentage of remaining wavelet detail coefficients.

5. DISCUSSION

We have proposed a denoising method for 2-D vector fields that are corrupted by additive noise. The method is an extension of scalar wavelet-based denoising techniques to vector data, and makes use of a vector wavelet transform.

We have shown that the proposed method outperforms Gaussian smoothing for low to moderate noise levels. For very high noise levels, the wavelet threshold selection appears to underestimate the noise level, and in such case, Gaussian filtering performs better. However, by adapting the threshold, we have demonstrated that the result can be improved. This should be investigated in a more systematic way, and it would be interesting to see if other wavelet coefficient threshold selection schemes produce better results.

We have also performed a simple experiment in which we used scalar denoising applied to the vector components independently. The result of this experiment shows that it is necessary to treat the vector components in a coupled way. It would be possible to use a component-wise scalar wavelet transform combined with our proposed vector coefficient thresholding. We expect, however, that the performance will still be lower, since the vector wavelet transform already considers the coupling of the vector components during the decomposition phase.

Currently, we are working on an extension to vectors with three components. This is challenging, since most research has focussed on multiwavelet design for vectors of only two components. This extension would open up the possibility of denoising 3-D vector fields, and could also result in a promising denoising method for diffusion-tensor MRI volumetric data. It may also be useful for the study of cardiovascular function, and a comparison with the method proposed by Ng [Ng03], should be made. Finally, it is necessary to evaluate the method on PVI data sets, which is ongoing work.

6. ACKNOWLEDGEMENTS

This research was funded by the project SFB 382 of the German Research Foundation (DFG), and by the Alexander von Humboldt Foundation with a Humboldt Research Fellowship for the first author.

7. REFERENCES

[Blo98] Blomgren, P. and Chan, T. F. Color TV: total variation methods for restoration of vector-valued images. *IEEE Trans. Image Processing*, 7(3):304–309, 1998.

[Buc95] Buckheit, J. B. and Donoho, D. L. WaveLab and reproducible research. Technical Report 474, Dept. of Statistics, Stanford University, 1995.

[Cha00] Chang, S. G., Yu, B., and Vetterli, M. Adaptive wavelet thresholding for image denoising and compression. *IEEE Trans. Image Processing*, 9(9):1532–1546, 2000.

[Chu92] Chui, C. K. *An Introduction to Wavelets*. Academic Press, 1992.

[Don95] Donoho, D. L. De-noising by soft thresholding. *IEEE Trans. Information Theory*, 41:613–627, May 1995.

[Fow02] Fowler, J. E. and Hua, L. Wavelet transforms for vector fields using omnidirectionally balanced multiwavelets. *IEEE Trans. Signal Processing*, 50:3018–3027, 2002.

[Jeo95] Jeong, J. and Hussain, F. On the identification of a vortex. *J. Fluid Mechanics*, 285:69–94, 1995.

[Mal89] Mallat, S. G. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

[Ng03] Ng, Y.-H. P. and Yang, G.-Z. Vector-valued image restoration with applications to magnetic resonance velocity imaging. *J. WSCG*, 11(2):338–345, 2003.

[Per90] Perona, P. and Malik, J. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[Per98] Perona, P. Orientation diffusions. *IEEE Trans. Image Processing*, 7(3):457–467, 1998.

[Piž03] Pižurica, A., Philips, W., Lemahieu, I., and Acheroy, M. A versatile wavelet domain noise filtration technique for medical imaging. *IEEE Trans. Medical Imaging*, 22(3):323–331, 2003.

[Pra00] Prasad, A. K. Particle image velocimetry. *Current Science*, 79(1):51–60, 2000.

[Rud92] Rudin, L. I., Osher, S., and Fatemi, E. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[Sim96] Simoncelli, E. P. and Adelson, E. H. Noise removal via Bayesian wavelet coring. In *Proc. IEEE Int. Conf. Image Processing*, volume 1, pages 379–382, Lausanne, Switzerland, September 16–19 1996.

[Str01] Strela, V. and Walden, A. T. Signal and image denoising via wavelet thresholding: Orthogonal and biorthogonal, scalar and multiple wavelet transforms. In Fitzgerald, W. F., Smith, R. L., Walden, A. T., and Young, P. C., editors, *Nonlinear and Nonstationary Signal Processing*. Cambridge University Press, 2001.

[Tan99] Tan, H. H., Shen, L.-X., and Tham, J. Y. New biorthogonal multiwavelets for image compression. *Signal Processing*, 79(1):45–65, 1999.

[Win04] Wink, A. M. and Roerdink, J. B. T. M. Denoising functional MR images: a comparison of wavelet denoising and Gaussian smoothing. *IEEE Trans. Medical Imaging*, 23(3):374–387, 2004.

[Xia96] Xia, X.-G. and Suter, B. W. Vector-valued wavelets and vector filter banks. *IEEE Trans. Signal Processing*, 44(3):508–518, 1996.

Fast and Accurate Hausdorff Distance Calculation between Meshes

Michael Guthe   Pavel Borodin   Reinhard Klein

University of Bonn
Institute of Computer Science II
Römerstraße 164
D-53117, Bonn, Germany

guthe@cs.uni-bonn.de   borodin@cs.uni-bonn.de   rk@cs.uni-bonn.de

ABSTRACT

Complex models generated e.g. with a laser range scanner often consist of several thousand or million triangles. For efficient rendering this high number of primitives has to be reduced. An important property of mesh reduction – or simplification – algorithms used for rendering is the control over the introduced geometric error. In general, the better this control is, the slower the simplification algorithm becomes. This is especially a problem for out-of-core simplification, since the processing time quickly reaches several hours for high-quality simplification.

In this paper we present a new efficient algorithm to measure the Hausdorff distance between two meshes by sampling the meshes only in regions of high distance. In addition to comparing two arbitrary meshes, this algorithm can also be applied to check the Hausdorff error between the simplified and original meshes during simplification. By using this information to accept or reject a simplification operation, this method allows fast simplification while guaranteeing a user-specified geometric error.

Keywords

Mesh comparison, Hausdorff error measurement, mesh simplification.

1. INTRODUCTION

Today, polygonal meshes have become ubiquitous as three-dimensional geometric representation of objects in computer graphics and some engineering applications. They are used for rendering of objects in a broad range of disciplines like medical imaging, scientific visualization, computer aided design (CAD), movie industry, etc. New acquisition techniques allow the generation of highly detailed objects with a permanently increasing polygon count. The handling of huge scenes composed of these high-resolution models rapidly approaches the computational capabilities of any graphics hardware. Therefore, level-of-detail techniques become inevitable. In order to build such level-of-detail representations many simplification algorithms exist that produce high-quality approximations of complex models with a reasonable amount of polygons.

The Journal of WSCG, Vol.13, ISSN 1213-6964
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency – Science Press

However, for many applications it is very important to have precise control over the geometric error introduced by simplification. The common way to provide an accurate error control, which can be used to calculate image space errors during visualization, is to measure the Hausdorff distance between the simplified and original meshes. However, this distance can only be approximated by sampling, and therefore, the better the accuracy is, the slower the measurement algorithm becomes. When used to steer simplification, the performance of the simplification algorithm is reduced accordingly.

The main contribution of this work is an efficient algorithm to measure and update the Hausdorff distance between a simplified mesh and the original model. The superior speed of our approach is mainly due to its ability to quickly determine regions of high geometric distance (or during simplification, regions where the distance is above the desired value) and adapt sampling there.

2. PREVIOUS WORK

Since mesh simplification is one of the fundamental techniques used for polygonal meshes, there is an extensive amount of different methods. Since there are detailed reviews of simplification algorithms (e.g. [Lue01]), we give only a short overview of the most related methods.

Rossignac and Borrel [Ros93] introduced the family of *vertex clustering* methods. Although very fast, their algorithm and its derivative methods (e.g. [Low97]) allow almost no control over the error (it is bound by the cell size), and the reduction rate is quite low in flat parts of the model.

Cohen et al. [Coh96] developed *simplification envelopes* to guarantee fidelity bounds while enforcing local and global topology. The simplification envelopes consist of two offset surfaces at some distance ε from the original surface. Since these envelopes are not allowed to self-intersect, ε is decreased at high curvature regions. By keeping the simplified surface inside these envelopes, the algorithm can guarantee a geometric deviation of at most ε, and additionally it checks that the surface does not self-intersect. While this algorithm has the advantage to guarantee a geometric error bound, it is quite slow and requires an orientable manifold for the construction of the offset surfaces. Zelinka and Garland [Zel02] modified this approach by using *permission grids* – spatial occupancy grids, where an operation is only performed if all cells that are intersected by the new triangles are allowed to be occupied. Although the algorithm is much faster than [Coh96] and doesn't need an orientable manifold mesh, the simplified model often contains much more triangles due to the discrete grid and the fact that the Manhattan distance is used instead of the Euclidean.

The *vertex pair contraction* operation introduced at the same time by Popović and Hoppe [Pop97] and Garland and Heckbert [Gar97] has become the most common operation and is used in many simplification methods. In conjunction with the quadric error metric introduced in that work, it offers flexible control over the quality, still at very high reduction speed. However, the quadric metric mostly overestimates the real geometric error which results in non-optimal reduction rates and the need to measure the exact error after simplification.

Klein et al. [Kle96] first used the Hausdorff distance between the original and simplified mesh to control the simplification error, although with significant computational effort. In [Bor03a] Borodin et al. have produced high-quality results by combining *generalized pair contractions* – an extension of the vertex pair contraction – with the control of the distance between the original and simplified models during the whole simplification process.

In the area of mesh comparison, Cignoni et al. [Cig98] introduced the first method dedicated exclusively to measurement of errors on simplified surfaces, which allows to compare quality of different simplification methods. Another method, presented by Aspert et al. [Asp02], is more efficient in terms of speed at the cost of higher memory use. Both algorithms are based on sampling of the geometry of the two models to be compared, where the sampling density depends on the desired accuracy. In order to double the accuracy the number of samples needs to be multiplied by four. Therefore, these algorithms quickly become slow for higher accuracy.

3. TERMINOLOGY

First we define the distance $d(p,S')$ between a point p on a surface S and another surface S' as:

$$d(p,S') = \min_{p'\in S'} d(p,p'),$$

where $d(p,p')$ is the Euclidian distance between two points in E^3.

The geometric distance – also called one-sided or single-sided Hausdorff distance – between two surfaces S and S' is then defined as:

$$d(S,S') = \max_{p\in S} d(p,S')$$

Note, that this distance is not symmetric in general, i.e. $d(S,S') \neq d(S',S)$. The symmetrical Hausdorff distance is defined as:

$$d_s(S,S') = \max\left(d(S,S'),d(S',S)\right)$$

This value gives more accurate measure of the distance between two surfaces by preventing the possible underestimation, which can appear if using only one-sided distances.

4. MESH COMPARISON

The main idea of our new mesh comparison algorithm is to adapt the sampling density used for distance calculation to the actual geometric deviation in the corresponding area. Hereby, the main goal is to draw samples only in those regions where the maximum distance between both objects is expected.

To achieve this, we first make two observations:

- Since the Hausdorff distance is defined as the maximum of the distances of all points on both meshes to the other mesh, we should avoid sampling in areas, where they are closer to each other than the actual – yet unknown – Hausdorff distance. This can be achieved by comparing coarse voxelizations of the two objects, considering triangles within voxels of high distance first, and stopping comparison, when the already found distance is larger than the highest possible distance between remaining voxels.

- When processing triangles inside a voxel cell, we only need to subsample a triangle, if its geometric distance can be larger than the already found maximum. This can only happen, if any of its vertices is farther away from the other mesh than one of its interior points, or if any of these distances exceeds the maximum. Therefore, a tight upper bound of a triangle-to-mesh distance is required.

Data Structures

To quickly determine the regions of high geometric distance we sort the triangles of both meshes into two voxel grids respectively. Note, that later on in our algorithms – similarly to [Cig98] and [Asp02] – this grid is also used to quickly find the closest point on one of the meshes for a given sample point.

The grid dimensions depend on the objects' bounding boxes and the number of triangles. We aim to have 10 triangles per occupied cell in average. This can be achieved approximately by calculating the number of required cells for a cube tessellated with the same number of triangles as is in the larger mesh. This leads to a resolution $r = \sqrt{\frac{\#triangles}{10 \cdot 6}}$. To avoid memory problems we restrict ourselves to resolutions of 256^3.

To speed up finding voxels of high distances between both voxelizations we use an octree structure for each of them, build upon the entries within the grids. In order to get full octrees we allow only resolutions of $2^n \times 2^n \times 2^n$.

Main Algorithm

Initially, we set the current Hausdorff distance to zero. We start traversing the octree structures of both meshes simultaneously, measuring the distance of each cell to all other cells on the same level in order to find the closest one in the other mesh. If for the current cell the closest other cell is found, we can calculate the minimum and maximum distances between two points inside these cells. If the minimum distance is larger than the current Hausdorff distance, we update the Hausdorff distance accordingly. If the maximum distance is less than or equal to the current Hausdorff distance, traversal of the subtree is skipped.

In order to consider cells containing triangles with larger distance first, the octree traversal is steered using a priority queue. This queue contains the already processed octree cells sorted by their maximum geometric distance.

When a leaf cell is reached during traversal, we collect all contained triangles and insert them into the same priority queue as the cells, again according to their maximum possible geometric distance. Depending on their minimum distance we again update the Hausdorff distance. To prevent multiple insertions of the same triangle into the priority queue, we mark triangles and process only those yet unmarked. The traversal and therefore the whole algorithm stops if either the queue becomes empty (e.g. when both meshes are identical) or the maximum possible distance of all remaining cells and triangles is less than the already found Hausdorff distance. The main algorithm to calculate the Hausdorff distance is shown in Fig. 1.

```
MinError=0
AddToQueue(RootCellA)
AddToQueue(RootCellB)
while(QueueNotEmpty)
   GetCellWithHighestMaxDistance
   UpdateMinError
   if(LeafNode)
      InsertTrianglesIntoQueue
   else
      InsertChildrenIntoQueue
return minError
```

Figure 1. Main algorithm to calculate the Hausdorff distance.

Cell-Based Distance

To quickly find the closest cell, when traversing the octree from a node to its children, we store all indices of occupied cells, for which the minimum distance was less than the maximum distance to the closest cell. Then we need to check only the children of these cells when calculating the distances of the cells' child nodes. Note, that for the root nodes calculating the closest cells and the distances is trivial.

To simplify the distance calculation, we use the bounding box of the union of both meshes to construct the grid. Furthermore, we restrict ourselves to cubic grid cells, which further simplifies the distance calculation to calculations based on the cell coordinates.

Distance of a Triangle

To calculate lower and upper bounds for the geometric distance between a triangle and the other mesh, we first need to calculate the distances of its vertices. If a vertex is inside the currently processed grid cell, we can use its stored closest cells to find candidate triangles for the next surface point in the other mesh. If it is outside the current cell, we descend the hierarchy again to find the occupied cells closest to the current vertex. Then we calculate distances to all triangles starting with those contained in the closest cell. When the distance to the closest point found so far is closer than the distance to the remaining cells, the distance of the currently processed vertex is found. To prevent multiple distance calculations for the same triangle, we store the indices of triangles and collect only the unprocessed triangles from each cell.

After the distances for the three vertices of the current triangle are calculated, we know that the minimum geometric distance of the triangle is the maximum of the vertex distances $\|V_i - P_i\|$, and the maximum geometric distance is at most the maximum of the vertex distances and the distances of the triangle barycentre B to the three vertex base points P_i (see Fig. 2).

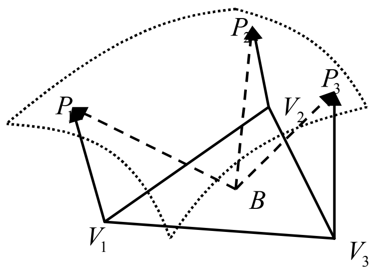


Figure 2. Minimum and maximum geometric distances of a triangle.

Therefore, we can determine the possible interval of the geometric distance d as:

$$d \geq \max_{i=1}^{3}\left(\|V_i - P_i\|\right) = d_{min}$$

$$d \leq \max\left(H_{min}, \max_{i=1}^{3}\left(\|B - P_i\|\right)\right).$$

Additionally, no point on the triangle can be farther away from the other mesh than its vertices from any of the base points, and thus

$$d \leq \min_{i=1}^{3}\left(\max_{j=1}^{3}\left(\|V_i - P_j\|\right)\right).$$

If the closest points of all three vertices lie on the same triangle (see Fig. 3), the maximum vertex distance is already the geometric distance of the current triangle. Otherwise, the triangle is inserted into the priority queue. Note, that we have to take care about the fact that the closest point may lie on several triangles (if it falls onto an edge or into a vertex).

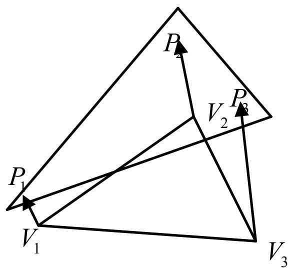


Figure 3. Exact geometric distance of a triangle.

When a triangle from the queue is processed, it is subdivided and the distances for its children are calculated. To prevent repeated calculation of the closest point/triangle for the same vertex, we calculate them for the three new vertices during subdivision. Then we only need to calculate the minimum and maximum possible distances before eventually storing the child triangles in the priority queue. The subdivision algorithm is shown in Fig. 4.

```
CalculateSubdivisionBasePoints
for(allChildTriangles)
   minDistance=max(vertexDistances)
   if(AllBasePointsOnSameTriangle)
      maxDistance=minDistance
   else
      maxDistance=max(barycenterDistances)
   InsertIntoQueue
```

Figure 4. Subdivision sampling algorithm.

Note, that calculating the base points and checking if they all lie on the same triangle is also necessary, when a leaf cell is processed in order to add all contained triangles to the queue.

5. APPLICATION TO SIMPLIFICATION

To control the Hausdorff error during simplification, only the part of the mesh affected by the current operation needs to be considered. Therefore, the affected triangles of the simplified mesh are directly inserted into the queue, and the error measurement for the original model is restricted to the region around these triangles using their common bounding box. Since the error of neighbouring triangles in the original model may also be affected, we need to extend this bounding box by the current Hausdorff error.

Furthermore, it is not necessary to calculate the exact geometric error, but only to check if it is below a user-specified threshold. Therefore, we do not need to insert cells or triangles, for which the maximum possible distance is below this threshold, into the queue, and thus refine sampling only in regions, where the error may be above this value. Analogously, if the minimum error found so far is above this threshold, we can immediately stop the calculation and reject the simplification operation. When calculating the geometric error of a triangle, we can also immediately stop searching for the base points P_i as soon as we found one that is closer than the desired error minus the maximum length of the two edges adjacent to the current vertex (according to the triangle inequation no vertex can be farther from a point than the distance of any vertex to this point plus the distance to this vertex).

The fact that only an accept/reject decision is required to decide, if a simplification operation will be performed, allows for some additional simple tests to quickly find an answer in most cases.

The simplification algorithm delivering the best trade-off between speed and quality of the simplified model is the one based on the quadric error metric [Gar97]. Choosing this simplification algorithm as base for our method, we get the additional advantage: the error quadric gives an (admittedly sometimes largely overestimated) upper bound for the Hausdorff error and can thus be used as a criterion to accept an operation without further tests.

Then two additional simple tests are possible to quickly reject an operation. First, the distance of the new vertex to the simplified mesh before the current edge collapse operation is calculated. If this exceeds twice the desired Hausdorff error ε, the operation can be rejected. Note, that exceeding of 2ε is required due to possible configurations similar to the one shown in Fig. 5.

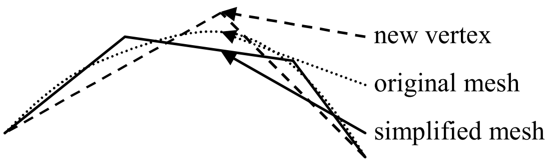


Figure 5. Quick reject tests.

If the operation passed this test, the distance from the new vertex to the original mesh is calculated. If this exceeds the specified threshold, the operation is also rejected. These two tests have the advantage that they quickly reject many operations and no update of the grid is required for their calculation.

When an operation passed these two tests without being rejected, the grid and octree of the simplified model are updated. If the operation has not been accepted by the quadric test, the Hausdorff distance between the updated meshes is calculated. When the operation is rejected by the Hausdorff error check, the vertex is split again, updating the grid and octree of the simplified mesh, and the operation is removed from the simplification queue. The overall pipeline of the error-checking algorithm is shown in Fig. 6.

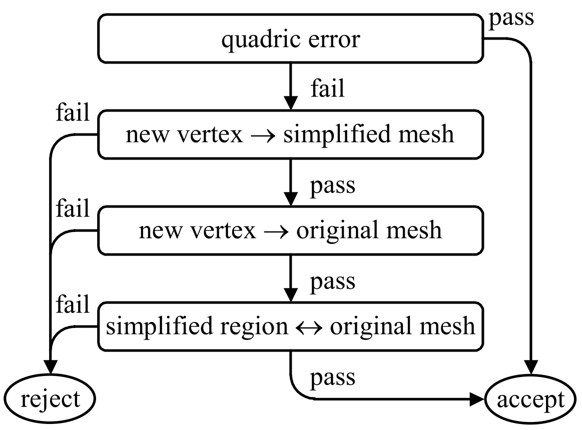

quadric error
pass
fail
fail
new vertex → simplified mesh
pass
fail
new vertex → original mesh
pass
fail
simplified region ↔ original mesh
pass
reject
accept

Figure 6. Error testing pipeline.

If the simplification queue is empty, all possible collapse operations that do not exceed the specified Hausdorff error have been performed.

6. RESULTS

Since our algorithm is applicable to both, measuring distances between meshes and controlling the introduced Hausdorff error during simplification, we compare it to previous approaches in both fields. We ran all tests on a PC with an Athlon 3000+ and 2 GB of main memory.

Mesh Comparison

To demonstrate the advantages of our algorithm, we compare its computation time with the two standard tools for measuring the Hausdorff distance: Metro [Cig98] (version 4.0) and MESH [Asp02] (version 1.12). The models used for evaluation are shown in

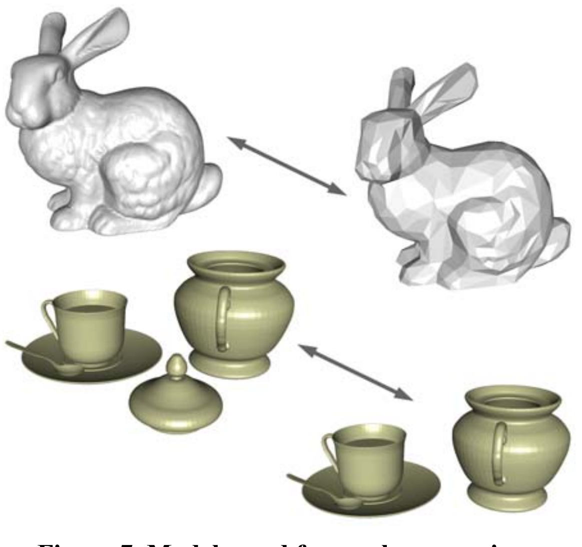


Figure 7. Models used for mesh comparison.

Fig. 7; the numbers of their vertices and triangles are listed in Tab. 1.

Model	# triangles	# vertices
Bunny (orig.)	69,451	34,834
Bunny (simpl.)	1,001	553
Coffee set	69,696	34,860
Without lid	60,936	30,480

Table 1. Models used for mesh comparison.

Tab. 2 shows the comparison in computation time of the three algorithms with an accuracy of 0.01% of the model diameter.

	Metro	MESH	Our alg.
Bunny	1,406 sec	395 sec	2.7 sec
Coffee set	13,008 sec	1,396 sec	2.1 sec

Table 2. Computation times of error-measuring algorithms.

At this accuracy our algorithm is several orders of magnitude faster than Metro and MESH, since we sample the mesh surface densely in regions of high geometric distance only. This is especially visible, when comparing the coffee set with and without lid, as shown in Fig. 8, where only samples in the region of the highest Hausdorff distance were taken.

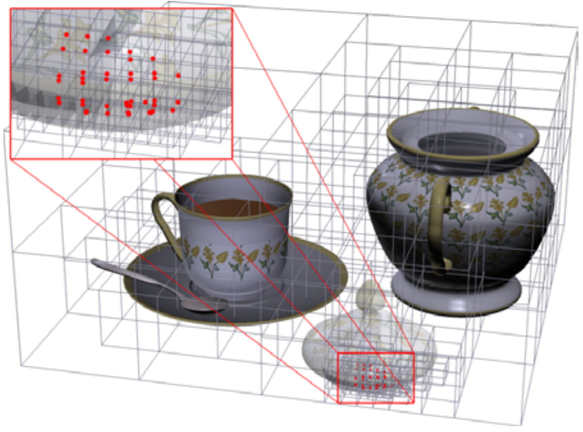


Figure 8. Visited octree cells and taken samples for coffee set scene with and without lid.

Fig. 9 shows a detailed plot of the computation times of the three algorithms, when comparing the simplified bunny with the original model, using different accuracies ranging from 1% of the bounding box diameter (practically useless) to 0.001% (very accurate).

It is clearly visible, that in contrast to both Metro and MESH, the computation time of our algorithm depends only very little on the desired accuracy. Note, that comparing the meshes with accuracy higher than 0.01% was not possible using MESH,

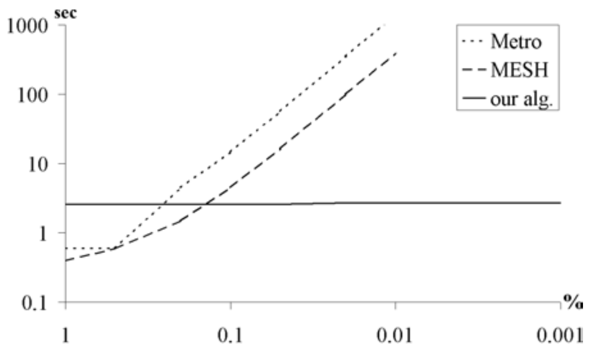


Figure 9. Computation times of error-measuring algorithms.

since it ran out of memory and Metro needs more than a day to compare the simplified and original bunny at 0.001%.

Error Control

In the field of error control during simplification, we compare our method with two simplification algorithms that guarantee a user-specified geometric error: *simplification envelopes* [Coh96] and *high-quality simplification* [Bor03a] (using the out-of-core simplification [Bor03b], when necessary). For comparison, we use different scanned objects from the Stanford 3D Scanning Repository [Sta3D] and the Digital Michelangelo Project [DigMi] shown in Fig. 10 and Tab. 3.

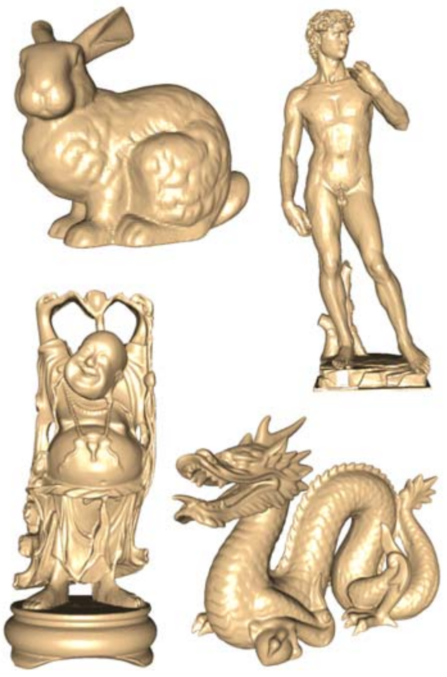


Figure 10. Models used for simplification.

Tab. 4 compares the computation times of the two mentioned simplification algorithms with our approach. For all models and algorithms the same simplification errors (1% and 0.1% of the model diameter) were used. The Hausdorff distance of 1%

Model	# triangles	# vertices
Bunny	69,451	34,834
Dragon	871,414	437,645
Buddha	1,087,474	543,652
David 2mm	7,227,031	3,614,098

Table 3. Models used for simplification.

is especially interesting for out-of-core simplification using hierarchical partitioning (e.g. [Bor03b]), since it is close to the resolution of $\frac{\varepsilon}{128}$ used for each octree cell.

	[Coh96]	[Bor03a]	our alg.
$\varepsilon = 1\%$			
Bunny	1:12	1:25	0:52
Dragon	n.a.	27:58	6:48
Buddha	n.a.	25:27[1]	12:37
David 2mm	n.a.	3:01:43[1]	1:06:22
$\varepsilon = 0.1\%$			
Bunny	0:46	0:46	1:28
Dragon	n.a.	15:37	14:59
Buddha	n.a.	24:08[1]	21:13
David 2mm	n.a.	3:00:03[1]	1:51:56

Table 4. Computation times of simplification algorithms.

Note, that the simplification envelopes restricts only the geometric error from the simplified model to the original, which is sufficient for rendering, but may cause inaccuracies for other applications like collision detection. Similarly, the high-quality simplification guarantees an upper bound for the geometric error from the original to the simplified model only, and thus may close large holes in the model, which is not always desired. Additionally, the accuracy is low, since only samples at vertex positions are taken. If out-of-core simplification is used, the error is only guaranteed to lie between $\frac{4}{5}\varepsilon$ and ε. This means that a more aggressive simplification would be possible without exceeding the threshold.

The computation time of the simplification envelopes is similar to the one of the high-quality simplification, but the algorithm requires orientable manifold meshes, and therefore worked only for the bunny model. Although our algorithm guarantees the Hausdorff distance to be below a specified threshold, the performance is even better than the simplification envelopes and the high-quality simplification for larger models and/or simplification errors.

[1] Out-of-core simplification [Bor03b].

7. CONCLUSION

We have presented an efficient algorithm to measure the geometric distances and the Hausdorff distance between two meshes. Our approach is much faster than existing algorithms for reasonable accuracies (i.e. less than 0.01% of the model diameter), since it needs to refine sampling only in regions of high distance and thus hardly depends on the required accuracy. This is accomplished by using a bi-hierarchical search algorithm to quickly find regions of possibly high geometric distances.

Furthermore, we have shown that our algorithm can also be applied to increase performance, efficiency, and accuracy of error-bounded simplification by using a chain of simple accept/reject tests to quickly determine, if exact evaluation of the Hausdorff distance is necessary. Instead of measuring the distance, we can stop traversing the hierarchy, when the minimum possible error is above the desired threshold, or the maximum possible is below. Using this technique, our approach is up to four times as fast as comparable algorithms when drastically simplifying the model.

8. ACKNOWLEDGEMENTS

We thank the Stanford 3D Scanning Repository and the Digital Michelangelo Project for providing us with the models. The coffee set model is courtesy of Renzo Del Fabbro.

9. REFERENCES

[Asp02] Aspert, N. Santa-Cruz, D., and Ebrahimi T. MESH: measuring errors between surfaces using the Hausdorff distance. Proc. of the IEEE International Conference on Multimedia and Expo, pp. 705-708, 2002.

[Bor03a] Borodin, P., Gumhold, S., Guthe, M., and Klein, R. High-quality simplification with generalized pair contractions. Proc. of GraphiCon '03, pp. 147-154, 2003.

[Bor03b] Borodin, P., Guthe, M., and Klein, R. Out-of-core simplification with guaranteed error tolerance. Proc. of Vision, Modeling and Visualisation '03, pp. 309-316, 2003.

[Cig98] Cignoni, P., Rocchini, C., and Scopigno, R. Metro: measuring error on simplified surfaces. Computer Graphics Forum, vol. 17, no. 2, pp. 167-174, 1998.

[Coh96] Cohen, J., Varshney, A., Manocha, D., Turk, G., Weber, H., Agarwal, P., Brooks, F., and Wright, W. Simplification envelopes. Computer

Graphics (Proc. of SIGGRAPH '96) 30, pp. 119-128, 1996.

[DigMi] The Digital Michelangelo Project. http://www-graphics.stanford.edu/projects/mich.

[Gar97] Garland, M. and Heckbert, P. S. Surface simplification using quadric error metrics. Computer Graphics (Proc. of SIGGRAPH '97) 31, pp. 209-216, 1997.

[Kle96] Klein, R., Liebich, G., and Straßer, W. Mesh reduction with error control. Proc. of IEEE Visualization '96, pp. 311-318, 1996.

[Low97] Low, K.-L. and Tan, T.-S. Model simplification using vertex-clustering. Proc. of Symposium on Interactive 3D Graphics, pp. 75-81, 1997.

[Lue01] Luebke, D. A Developer's Survey of Polygonal Simplification Algorithms. IEEE Computer Graphics and Applications, 21(3), pp. 24-35. 2001.

[Pop96] Popović, J. and Hoppe, H. Progressive simplicial complexes. Computer Graphics (Proc. of SIGGRAPH '97) 31, pp. 217-224, 1997.

[Ros93] Rossignac, J. and Borrel, P. Multi-resolution approximations for rendering. Modeling in Computer Graphics, pp. 455-465, 1993.

[Sta3D] The Stanford 3D Scanning Repository. http://www-graphics.stanford.edu/data/3dscanrep.

[Zel02] Zelinka, S. and Garland, M. Permission grids: practical, error-bounded simplification. ACM Transactions on Graphics, 21(2), pp. 1-25, 2002

Go with the Winners Strategy in Path Tracing [1]

László Szirmay-Kalos, György Antal

Dept. of Control Eng. and Information Tech.
TU Budapest, Budapest
Magyar Tudósok krt. 2., H-1117, Hungary
szirmay@iit.bme.hu, gy_antal@yahoo.com

Mateu Sbert

Inst. of Applied Math. and Informatics
University of Girona, Girona
Campus Montilivi, E-17071, Spain
mateu@ima.udg.es

ABSTRACT

This paper proposes a new random walk strategy that minimizes the variance of the estimate using statistical estimations of local and global features of the scene. Based on the local and global properties, the algorithm decides at each point whether a Russian-roulette like random termination is worth performing, or on the contrary, we should split the path into several child paths. In this sense the algorithm is similar to the go-with-the-winners strategy invented in general Monte Carlo context. However, instead of establishing thresholds to make decisions, we compute the number of child paths on a continuous level and show that Russian roulette can be interpreted as a kind of splitting using fractional number of children. The new method is built into a path tracing algorithm, and a minimum cost heuristic is proposed for choosing the number of reflected rays. Comparing it with the classical path tracing approach we concluded that the new method reduced the variance significantly.

Keywords: Global illumination, random walk, Monte Carlo methods.

1 Introduction

Random walk global illumination algorithms evaluate an infinite sequence of integrals of the following form:

$$L^r = \int_{\Omega_1} w_1 \cdot \left(L^e + \int_{\Omega_2} w_2 \cdot (L^e + \ldots)\; d\omega_2 \right) d\omega_1 \tag{1}$$

where L^r is the reflected radiance, L^e is the emission, w is the scattering density, usually expressed as the product of the BRDF and the cosine of the orientation angle, and Ω_i is the set of directions of possible illumination.

When the first outer integral is estimated by Monte Carlo techniques n_1 random directions are obtained with a probability density p_1, and the following quadrature is computed

$$\hat{L}^r = \frac{1}{n_1} \cdot \sum_{i=1}^{n_1} \frac{w_1(\omega_1^i)}{p_1(\omega_1^i)} \cdot L_i^{\text{in}} = \sum_{i=1}^{n_1} W_1^i \cdot L_i^{\text{in}}.$$

where $L_i^{\text{in}} = L_i^e + L_i^r$ is the sum of the emission and the reflected radiance at the hit point of the traced ray. If $n_1 > 1$, then the random path is split into n_1 paths at this point. When $n_1 = 1$ *splitting* does not happen. Term $\frac{w_1(\omega_1^i)}{p_1(\omega_1^i)} \cdot L^e$ can be immediately added to the estimate, but the computation of reflected radiance L_i^r poses a similar integration problem, which can be solved by repeating the same procedure. Each step l we add n_l number of $W_l \cdot L^e$ terms to the quadrature, where *potential* W_l can be expressed in a product form

$$W_l = W_{l-1} \cdot \frac{1}{n_l} \cdot \frac{w_l(\omega_l^i)}{p_l(\omega_l^i)}. \tag{2}$$

Sampling a random direction is not the only way to estimate L^r in a point if a random approximation of the radiance function is available in the scene. Taking this random approximation as the real radiance and computing its reflection by deterministic connections also lead to the estimate of all remaining terms of the infinite Neumann series. This corresponds to *joining* the path with

[1] Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. The Journal of WSCG, Vol.13, ISSN 1213-6964. WSCG'2005, January 31-February 4, 2005 Plzen, Czech Republic. Copyright UNION Agency - Science Press

the results of other paths obtained earlier. Finally, we might decide not to continue the computation of the path. This is called *termination.*

As we walk along the random path, we make a decision at in each step. Should we spawn new random rays, or should we estimate the reflected radiance directly? If random rays are sampled, what is their optimal number? Each of these decisions results in a term in the integral quadrature and also an error in the estimate. The complete rendering algorithm will evaluate many recursive integrals with a lot of random paths, thus we make a lot of decisions that affect both the error of the integral quadratures and the total computation time. In this paper we propose an approach that minimizes this total computation error and keeps the computational time low.

Section 2 reviews the previous work, and particularly the go-with-the-winners strategy. In section 3 we present a theoretical analysis of the simultaneous application of Russian roulette, splitting and joining, and extend the concept of the go-with-the-winners strategy to use continuous scale. Then in section 4 the cost-variance optimization is discussed. Finally, in section 5 we present a minimum cost heuristic for choosing the number of reflected rays in stochastic ray tracing.

2 Previous work

Path splitting, joining and termination have been intuitively and partially applied in several random walk global illumination algorithms.

Russian-roulette[AK90] terminates the walk randomly. When the path is terminated, no deterministic estimation takes place, and the illumination of this point is supposed to be zero. The probability of the random termination is the albedo of the visited point, or the luminance of the albedo in case of spectral rendering. In order to compensate the not computed terms, when the integrand is really computed, it is divided by the continuation probability. There are several problems of classical Russian-roulette. It increases the variance inversely proportional to the continuation probability. On the other hand, spectral rendering poses another problem to Russian roulette, where the contributions are transferred on different wavelengths simultaneously, but the continuation probability should obviously remain a scalar value. If this scalar is the luminance of the albedo, then the estimation can be very poor if the spectrum of the reflection does not coincide with the transferred potential. For example, when a path visits first a red, then a green surface, then the contribution will be zero, but this is not recognized by Russian-roulette. These problems have been pointed out in [SSKK03].

The variance introduced by Russian roulette can also be reduced by setting the termination probability globally and not locally. It means that the continuation probability is the average albedo of the whole scene, and not the local albedo. Such approach was used by Keller [Kel97], when the continuation probability has been determined separately, and also in ray-based stochastic iteration algorithms, where the contraction ratio of the integral operator has been determined on the fly [SK99]. However, global termination probability may also cause infinite variance.

In random walk algorithms that reuse light paths, we also have a random estimate of the incoming illumination, which can be obtained without continuing the random walk. The acquisition of this estimate may require data-structure searches (*photon-map* [JC95, Chr00], *irradiance caching* [WRC88], *discontinuity buffer* [WKB$^+$02]) or tracing deterministic shadow rays (*bi-directional path tracing* [LW93, VG95], *virtual light sources algorithm* [Kel97, WKB$^+$02], and *path reuse* [BSH02]).

The benefits of path termination, splitting and joining can also be combined. In path reuse methods, paths are terminated by Russian-roulette, and its visited points are joined with other paths. Since such methods may generate a complete path in many different ways a clever weighting scheme should be applied, as proposed by *multiple importance sampling* [Vea97].

Considering these, we can conclude that termination, splitting and joining have already shown up in many different random walk algorithms, and even their intuitive optimal combination has been emerged. On the other hand, Bolin and Meyer [BM97] analyzed the variance of Russian-roulette and splitting.

In this paper we follow this direction of the previous work in order to find optimal termination/splitting/joining, which results in the smallest error. This work has been inspired by a general Monte Carlo strategy called *go with the winners* [AV94, Gra01] that can include many approaches dealing with termination and splitting [Kah56]. In this method, the decision is made according to the accumulated potential W, which is compared with two predefined constants W^- and W^+ ($W^- < W^+$). If $W < W^-$, then Russian roulette is executed with probability W/W^-. If $W^- \leq W < W^+$, then the path is extended by a single ray. If $W^+ \leq W$, then the random path splits to n subpaths (say $n = 10$), and the potential is divided by n.

3 Random walks with termination, splitting and joining

Suppose that $l-1$ steps of the random walk have already been computed and we are facing the decision of what to do having potential W_{l-1}. If the walk is continued, then W_l needs to be found, and n_l estimates of reflected radiance $\hat{L}_l^r$ are added to the quadrature. If the walk is not continued, then the estimate should cover all $l, l+1, \ldots$ steps, which can also be added to the quadrature. The random termination can also be imagined similarly to splitting, but now we use $n_l \leq 1$ number of random directions in average. The average value comes from the fact that sometimes the path is not continued at all.

At a given point of the random walk, parameter n_l must be determined to minimize the error. Each sample contributes to the square error of the integral quadrature proportionally to its own square error, which equals to the variance in the unbiased case, and to the sum of the variance and the square of the bias in the biased case. Thus the decision should be made to minimize the introduced error.

The variance computation is discussed for splitting and random termination separately.

3.1 Splitting: $n_l \geq 1$

When n_l random directions are used, the estimator of the contribution of paths of length l is

$$\hat{L}_l^r = W_{l-1} \cdot \frac{1}{n_l} \cdot \sum_{i=1}^{n_l} \frac{w_l(\omega_l^i)}{p_l(\omega_l^i)} \cdot L_i^{\text{in}},$$

where ω_l^i is the ith random sample of integrand variable ω_l, and $p_l(\omega_l^i)$ is the probability density of obtaining this sample. This means breaking the paths to n_l children, where child i has

$$W_l^i = W_{l-1} \cdot \frac{1}{n_l} \cdot \frac{w_l(\omega_l^i)}{p_l(\omega_l^i)}$$

potential, and $W_l^i \cdot L_i^{\text{in}}$ is the contribution of this path. When using this formula in practical algorithms, we can usually assume that p_l mimics w_l, i.e. where w_l is non-zero, their ratio $w_l/p_l = a_l$ is — at least approximately — constant. This constant is the probability that the light is not absorbed, and is called the *albedo*.

The variance of this contribution is:

$$\frac{W_{l-1}^2}{n_l^2} \cdot D^2 \left[\frac{w_l}{p_l} \cdot L^{\text{in}}\right] = \frac{W_{l-1}^2}{n_l^2} \cdot a_l^2 \cdot D^2 \left[L^{\text{in}}\right].$$

The total variance of the family of n_l samples obtained by splitting the path is n_l times this variance since the children can be assumed to be independently generated. Thus the total variance of the family of paths is:

$$\frac{W_{l-1}^2}{n_l} \cdot a_l^2 \cdot D^2 \left[L^{\text{in}}\right]. \qquad (3)$$

3.2 Random termination: $n_l < 1$

In this case the average number of samples to continue a path is less than 1. It corresponds to the case when the probability of path continuation is n_l. When no random sample is taken, the result of all remaining steps — i.e. the contribution of paths of length $l, l+1, \ldots$ — is estimated by a known constant value, for example by 0 as suggested by Russian-roulette. To be general, let us assume that we have a random estimate $\hat{L}$, which is available without spawning random rays. The expected value of estimate $\hat{L}$ may or may not be equal to the exact integral value L^r, which can be expressed by bias ΔL in this estimation:

$$E[\hat{L}] = L^r + \Delta L, \quad L^r = E\left[\frac{w_l}{p_l} \cdot L^{\text{in}}\right].$$

When the walk is decided to be terminated, we use available estimate $\hat{L}$. If the walk is continued, then a linear combination of actually computed radiance $W_{l-1} \cdot w_l/p_l \cdot L^{\text{in}}$ and estimate $W_{l-1} \cdot \hat{L}$ is inserted in the estimator, that is, we use

$$W_{l-1} \cdot \left(\alpha \cdot \frac{w_l}{p_l} \cdot L^{\text{in}} + \beta \cdot \hat{L}\right).$$

The α and β values of this linear combination can be determined from the requirement that the expected value of this estimator should be correct:

$$n_l \cdot W_{l-1} \cdot \left(\alpha \cdot E\left[\frac{w_l}{p_l} \cdot L^{\text{in}}\right] + \beta \cdot E\left[\hat{L}\right]\right) +$$

$$(1 - n_l) \cdot W_{l-1} \cdot E\left[\hat{L}\right] =$$

$$W_{l-1} \cdot L^r \cdot (1 - (1 - \alpha - \beta) \cdot n_l) +$$

$$W_{l-1} \cdot \Delta L \cdot (1-(1-\beta)\cdot n_l).$$

Note that the cases of continuation and termination have been weighted with n_l and $1-n_l$, respectively, since these are their probabilities. To make this estimate unbiased, it should be equal to $W_{l-1}\cdot L^r$, thus $\alpha+\beta=1$ should hold, and the following term should be zero

$$W_{l-1} \cdot \Delta L \cdot (1-\alpha \cdot n_l).$$

Even if $\hat{L}$ is biased (i.e. ΔL is not zero), the bias of the random walk estimate can be made zero by setting $\alpha = 1/n_l$. Using this assumption, the variance of the estimate is

$$n_l \cdot W_{l-1}^2 \cdot E\left[\left(\frac{w_l/p_l \cdot L^{\text{in}}}{n_l} - \frac{(1-n_l)\cdot \hat{L}}{n_l}\right)^2\right] +$$

$$(1-n_l)\cdot W_{l-1}^2 \cdot E\left[\hat{L}\right]^2 - W_{l-1}^2 \cdot (L^r)^2 =$$

$$\left(\frac{1}{n_l}-1\right)\cdot W_{l-1}^2 \cdot E\left[\left(\frac{w_l}{p_l}\cdot L^{\text{in}} - \hat{L}\right)^2\right] +$$

$$W_{l-1}^2 \cdot D^2\left[\frac{w_l}{p_l}\cdot L^{\text{in}}\right].$$

This formula can be used to obtain the variance for a given n_l. Note that if $\hat{L}$ is not far from an unbiased estimator, i.e. $\hat{L} \approx L^r = E\left[\frac{w_l}{p_l}\cdot L^{\text{in}}\right]$, then

$$E\left[\left(\frac{w_l}{p_l}\cdot L^{\text{in}} - \hat{L}\right)^2\right] \approx E\left[\left(\frac{w_l}{p_l}\cdot L^{\text{in}} - L^r\right)^2\right],$$

which equals to $D^2\left[\frac{w_l}{p_l}\cdot L^{\text{in}}\right]$, and thus the variance is approximately

$$\frac{W_{l-1}^2}{n_l}\cdot D^2\left[\frac{w_l}{p_l}\cdot L^{\text{in}}\right] = \frac{W_{l-1}^2}{n_l}\cdot a_l^2 \cdot D^2\left[L^{\text{in}}\right].$$

Note that the variance has the same formula as derived for the case of splitting (equation 3).

3.3 Estimation of $D^2\left[L^{\text{in}}\right]$

We face the problem that incoming radiance L^{in} is a random variable and is not known. The variance of L^{in} can come from two different sources. On the one hand, for fixed ω, the incoming radiance is estimated by continuing the random walk, which obtains the estimate by random simulation. On the other hand, even if we exactly knew the conditional expectation $\tilde{L}^{\text{in}}(\omega)$ of $L^{\text{in}}(\omega)$ for fixed incoming direction ω, then the variation of this expectation for different incoming directions would be another source of the error. Formally, we can write

$$D^2\left[L^{\text{in}}\right] = E\left[\left(L^{\text{in}} - E\left[L^{\text{in}}\right]\right)^2\right] =$$

$$\int_\Omega E\left[\left(L^{\text{in}} - E\left[L^{\text{in}}\right]\right)^2 \mid \omega\right]\cdot p_l(\omega)d\omega =$$

$$\int_\Omega E\left[\left(L^{\text{in}}(\omega) - \tilde{L}^{\text{in}}(\omega)\right)^2\right]\cdot p_l(\omega)\,d\omega +$$

$$\int_\Omega \left(\tilde{L}^{\text{in}}(\omega) - E\left[L^{\text{in}}\right]\right)^2 \cdot p_l(\omega)\,d\omega.$$

The first term in this sum describes how well the algorithm can estimate the radiance of a single point, and is approximated by a global constant V_R. The second term, on the other hand, represents how quickly the incoming radiance changes in the domain of the random directions, which is prescribed by the local BRDF. For instance, if the examined point is an ideal mirror, then BRDF sampling samples just a single direction, and the second term is zero. Generally, the second term gets bigger as the size of the set of possible directions grows. As can be shown the dependence is quadratic, that is, the second term is proportional to the square of the size of the directional domain. Let us consider a simple, Phong-like BRDF with shininess parameter s. Diffuse and mirror like materials can be imagined as special cases of $s=0$ and $s=\infty$, respectively. The size of the domain of a Phong-like BRDF is $2\pi/(s+1)$ [LW94], thus the second term is approximated by $V_V/(s+1)^2$, where V_V is a general global constant.

Summarizing, the total variance of the children of a single parent is approximated as

$$\frac{W_{l-1}^2}{n_l}\cdot a_l^2 \cdot \left(V_R + \frac{V_V}{(s_l+1)^2}\right).$$

4 Variance-cost optimization

In the previous section we determined the variance associated with splitting and random termination with incoming radiance estimation. The variance is inversely proportional to value n, which stands for the average number of continued path at this point. On the other hand, if ray tracing is responsible for a major part of the computation time, then the cost is proportional to n. The goal is to obtain the most accurate result paying the lowest cost, that is, to minimize the total variance of the result with a constraint on the total number of rays. Formally, the optimization goal has the form

$$\sum_k \sum_l \sigma_{k,l}^2 / n_{k,l},$$

where k considers each light path and l each ray of a path, and

$$\sigma_{k,l}^2 = D^2 \left[W_{k,l-1} \cdot \frac{w_{k,l}}{p_{k,l}} \cdot L_k^{\text{in}} \right] \approx$$

$$W_{k,l-1}^2 \cdot a_{k,l}^2 \cdot \left(V_R + \frac{V_V}{(s_{k,l}+1)^2} \right),$$

with constraint $\sum_k \sum_l n_{k,l} = N$, where $n_{k,l}$ is the average number of paths leaving the lth sample point of path k, and N is the total number of rays used to compute the whole image. Using the Lagrange multiplier method, we have to find the minimum of

$$\sum_k \sum_l \frac{\sigma_{k,l}^2}{n_{k,l}} + \lambda \cdot \left(\sum_k \sum_l n_{k,l} - N \right).$$

Making the partial derivatives equal to zero, we obtain

$$n_{k,l} = N \cdot \frac{\sigma_{k,l}}{\sum_{k'} \sum_{l'} \sigma_{k',l'}}.$$

It means that at each visited point number of child rays n_l should be proportional to

$$W_{k,l-1} \cdot a_{k,l} \cdot \sqrt{V_R + \frac{V_V}{(s_{k,l}+1)^2}}.$$

We could establish only a requirement of proportionality, and parameters V_R and V_V are left free. These parameters depend on the scene properties and may also be subjects for statistical estimation. On the other hand, we can follow a simple intuition. Assume that the accumulated potential and the albedo are maximum, that is $W_{l-1} \cdot a_l = 1$. If the surface is an ideal mirror, i.e. $s_l = \infty$, then a reasonable way to continue the path randomly with exactly one child. On the other hand, if the surface is purely diffuse, i.e. $s_l = 0$, and we may require the maximum number of children equal to $n_{\max}$. The optimal selection of $n_{\max}$ depends also on the properties of the scene. For example, if the illumination in the scene is homogeneous, i.e. a point receives similar illumination from all directions, then $n_{\max}$ is 1. As the illumination gets more and more heterogeneous, $n_{\max}$ is worth increasing. We used value 10 in the implementation, which seems to be a good choice for practical scenes. From these two requirements, V_R and V_V can be obtained, and the general formula for the number of children is

$$n_{k,l} = W_{k,l-1} \cdot a_{k,l} \cdot \sqrt{1 + \frac{n_{\max}^2 - 1}{(s_{k,l}+1)^2}}. \qquad (4)$$

If the material model consists of several different elementary materials (e.g. diffuse + specular), then the number of children should be computed separately using the albedo of the elementary BRDFs, and then the results should be added.

5 Variance based Go with the Winners Strategy

We propose a path tracing algorithm that is driven by the theoretical results of previous sections. Note that if path tracing used only BRDF sampling, then the probability of hitting small light sources would be very small. In order to avoid this problem, the illumination of small light sources is directly estimated at each point of the random walk. This technique, which is called *next event estimation* or *direct light source computation*, is also incorporated into both the reference and the new algorithm.

At each visited point number of child rays n_l is computed according to equation 4. If the computed n_l turns out to be less than 1, then the child ray is traced only with probability n_l. If we decide not to trace the child ray, then estimate $\hat{L}$ is used instead. If according to the random decision, we have to trace a child ray, then $(1 - n_l)/n_l \cdot \hat{L}$ is subtracted from the result. On the other hand, if the computed n_l is greater than 1, we find the nearest integer and spawn n_l child rays from this point. The potential passed with a child ray is divided by n_l.

The first problem that needs to be solved is to find an approximation of radiance $\hat{L}$. We could use, for example, a photon map, or a statistical estimation gained during the computation of previous paths. In the implementation we made a direct estimation in the following way [SSKK03].

Suppose that the scene is closed. In this case, we can approximate the average reflected radiance in the scene, which can be regarded as an estimate for $\hat{L}$. Note that we use the reflected radiance here, since the direct illumination is computed separately by next event simulation. The total emitted power of the light sources is

$$\Phi^e = \int_S \int_\Omega L^e(\vec{x}, \omega) \cdot \cos\theta \, d\vec{x} d\omega$$

where S is the set of all surface points, L^e is the emitted radiance and θ is the angle between the direction of the emission and the surface normal. This emitted power will be multiplied by the albedo at each reflection. Suppose that the average albedo in the scene is $\tilde{a}$. The reflected power in the scene is the sum of the single reflection, double reflection, etc., that is:

$$\Phi^r \approx \Phi^e \cdot (\tilde{a} + \tilde{a}^2 + \ldots) = \frac{\tilde{a}\Phi^e}{1 - \tilde{a}}.$$

From the average power, we can obtain the average radiance:

$$\hat{L}(\vec{x}, \omega) \approx \frac{1}{\pi S} \cdot \frac{\tilde{a}\Phi^e}{1 - \tilde{a}}.$$

Formula 4 contains the accumulated potential of the path, W_{l-1}. The computation of W_{l-1} poses no particular problem, as we increase the length of the path, the potential is updated according to equation 2. However, we have to take into account that in the global illumination problem the potential is not scalar, but a vector whose elements correspond to the wavelengths on which the computation is carried out. These vectors are multiplied as *diadic products*, that is, the result is also a vector of the same dimension, whose elements are the products of the respective elements in the two operands.

The albedo showing up in equation 4 is available as a local material property, as well as shininess parameter s_l. Note that the albedo also depends on the wavelength, thus diadic product is applied when it is multiplied with the potential.

The modified versions of equations 4 and 2 for the spectral case, denoting the diadic product by $\circ$ and the luminance of a spectrum by $\mathcal{L}$, is:

$$n_{k,l} = \mathcal{L}(W_{k,l-1} \circ a_{k,l}) \cdot \sqrt{1 + \frac{n_{\max}^2 - 1}{(s_{k,l} + 1)^2}},$$

$$W_l = \frac{W_{l-1} \circ w_l(\omega_l^i)}{n_l \cdot p_l(\omega_l^i)}.$$

6 Simulation results

The proposed variance based go with the winner strategy has been implemented in a path tracing algorithm. The results are compared with the classical path tracing applying Russian roulette. The termination probability was set equal to the local albedo. In both algorithms we included direct light source computation (next event simulation) to handle small light sources.

To make the comparison fair, we allowed the two algorithms to use the same number of rays to compute the image. The new method distributed the available rays differently for pixels and for the different levels of recursion, aiming at the goal to place more rays at higher variance domains. We were surprised that when the two methods traced the same number of rays, the go-with-the-winner solution was about 20% faster. A possible explanation is that the new method applies much less recursive calls to generate child rays, and the rays resulted from splitting are much more coherent, thus the new method automatically provides better cache utilization. The rendering times were measured in the open source RenderX.NET [Ant04] global illumination framework, that is a software package written completely in C# targeting the .NET platform.

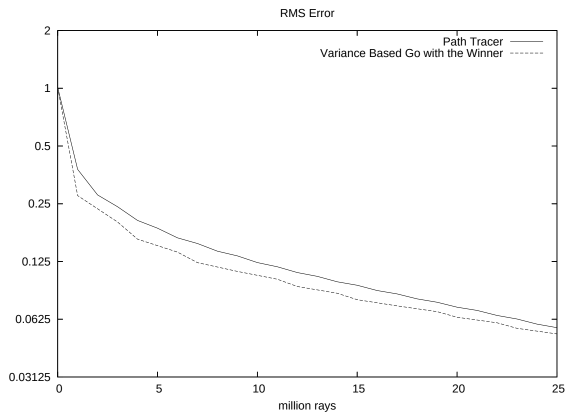


Figure 1: *Relative error curves obtained with the original path tracing algorithm and the proposed method for the Cornell Girl scene.*

The computed images are shown by figures 2 and 3, which demonstrate the superior performance of the new method. On the one hand, examining the error curves (figure 1) we can conclude that the new method can provide the same error level using about 30-50% less rays. The improved image quality is due to several features. The new method distributes the variance evenly in the pixels of the image, and does not devote unnecessary amount of computation to simpler parts. On the other hand, splitting allows to reuse path segments, which also saves time and make the saved time available to generate additional paths.

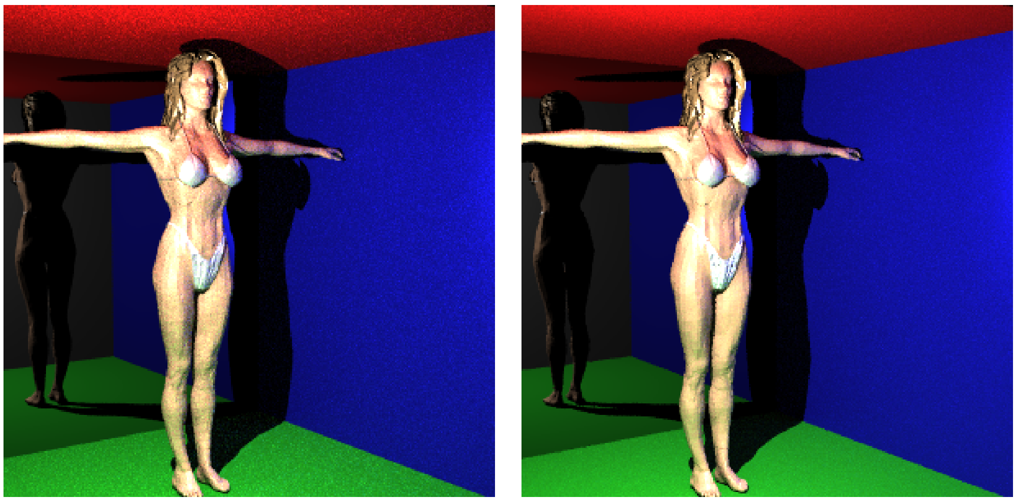


Figure 2: Comparison of classical path tracing with Russian-roulette and path tracing using the go with the winner strategy for a Cornell Girl scene. Both images have been obtained by casting 9 million rays. The image resolution is 300 × 300.

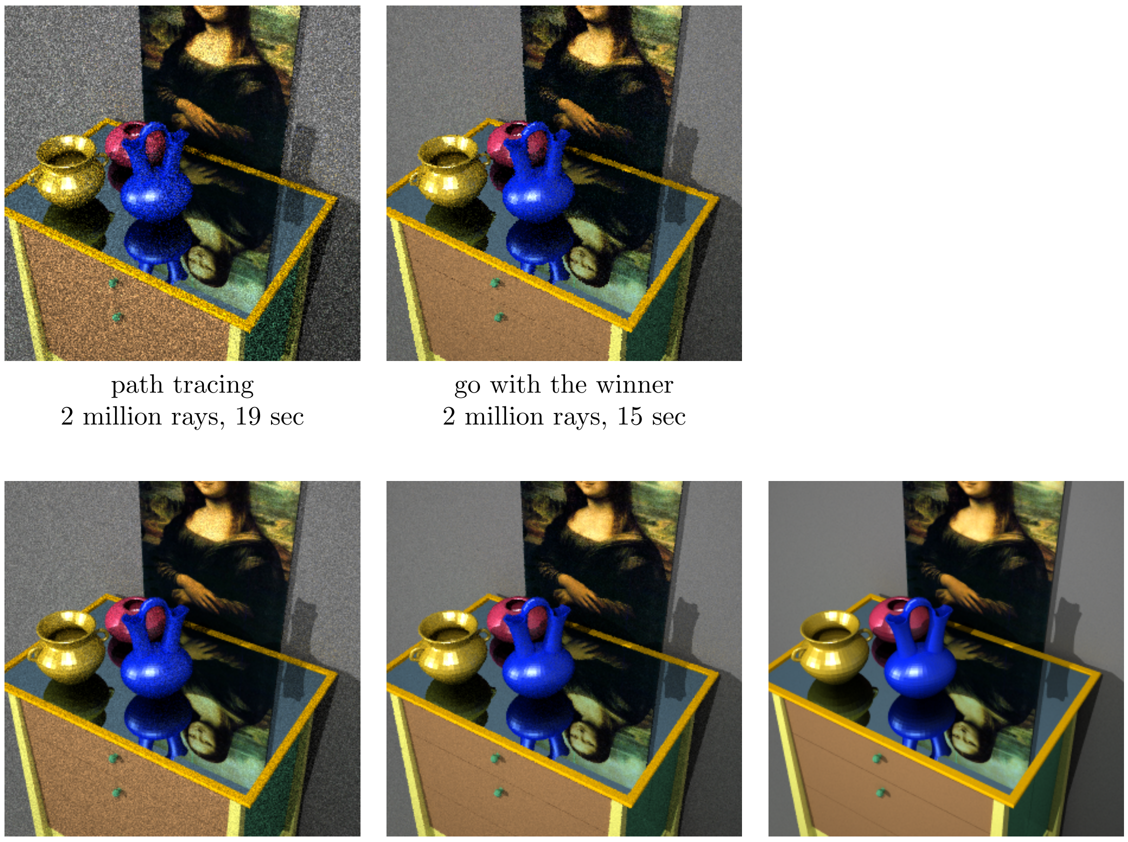


Figure 3: Comparison of classical path tracing with Russian-roulette and path tracing using the go with the winner strategy for the “Table with vases” scene. The image resolution is 300 × 300.

7 Conclusions

This paper proposed an extended go with the winner strategy to improve random walk global illumination algorithms. The basic idea is that at each visited point the variance caused by tracing the next random ray is estimated, and we split or randomly terminate the path to maintain a roughly constant variance in all steps. The variance estimation seems complicated at the first glance, but the implementation of the method is still straightforward. Having a random walk global illumination program, the required modifications are trivial to implement. The simple formula of equation 4 should be included, and based on the result several random rays should be generated, or if it is smaller than 1, this value will be the continuation probability of Russian roulette.

According to our measurements, this simple change can speed up the calculation by about 30-50% due to the better distribution of rays, and other 20% speed up is due to reducing the number of recursive calls and making the rays more coherent.

8 Acknowledgements

This work has been supported by the GameTools FP6-004363 EU project, OTKA ref. No.: T042735, by TIN 2004-07451-C03-01, and by the Spanish-Hungarian Action Fund. The scenes have been modeled by Maya that was generously donated by AliasWavefront.

REFERENCES

[AK90] J. Arvo and D. Kirk. Particle transport and image synthesis. In *SIGGRAPH '90 Proceedings*, pages 63–66, 1990.

[Ant04] Gy. Antal. RenderX.NET. 2004. http://www.sourceforge.com/projects/renderx-net.

[AV94] D. Aldous and U. Vazirani. Go with the winners algorithms. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, 1994.

[BM97] M. R. Bolin and G. W. Meyer. An error metric for Monte Carlo ray tracing. In *Rendering Techniques '97*, pages 57–68, 1997.

[BSH02] P. Bekaert, M. Sbert, and J. Halton. Accelerating path tracing by re-using paths. In *Proceedings of Workshop on Rendering*, pages 125–134, 2002.

[Chr00] P. Christensen. Faster photon map global illumination. *Journal of Graphics Tools*, 4(3):1–10, 2000.

[Gra01] P. Grassberger. Go with the winners: a general monte carlo strategy. In *Proceedings der CCP2001)*, 2001.

[JC95] H. W. Jensen and N. J. Christensen. Photon maps in bidirectional Monte Carlo ray tracing of complex objects. *Computers and Graphics*, 19(2):215–224, 1995.

[Kah56] H. Kahn. Use of different monte carlo sampling techniques. In *Symposium on Monte Carlo Method*, 1956.

[Kel97] A. Keller. Instant radiosity. *SIGGRAPH '97 Proceedings*, pages 49–55, 1997.

[LW93] E. Lafortune and Y. D. Willems. Bi-directional path-tracing. In *Compugraphics '93*, pages 145–153, Alvor, 1993.

[LW94] E. Lafortune and Y. D. Willems. Using the modified Phong reflectance model for physically based rendering. Technical Report RP-CW-197, Department of Computing Science, K.U. Leuven, 1994.

[SK99] L. Szirmay-Kalos. Stochastic iteration for non-diffuse global illumination. *Computer Graphics Forum (Eurographics'99)*, 18(3):233–244, 1999.

[SSKK03] L. Szécsi, L. Szirmay-Kalos, and C. Kelemen. Variance reduction for russian-roulette. *Journal of WSCG*, 11, 2003.

[Vea97] E. Veach. *Robust Monte Carlo Methods for Light Transport Simulation.* PhD thesis, Stanford University, 1997.

[VG95] E. Veach and L. Guibas. Optimally combining sampling techniques for Monte Carlo rendering. In *ACM SIGGRAPH '95 Proceedings*, pages 419–428, 1995.

[WKB$^+$02] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slussalek. Interactive global illumination using fast ray tracing. In *13th Eurographics Workshop on Rendering*, 2002.

[WRC88] G. J. Ward, F. M. Rubinstein, and R. D. Clear. A ray-tracing solution for diffuse interreflection. *Computer Graphics*, 22(4):85–92, 1988.

Statistical Shape Analysis for Computer Aided Spine Deformity Detection

Gerhard H. Bendels   Reinhard Klein   Mandana Samimi   Alfred Schmitz

University of Bonn
Institute of Computer Science II
Computer Graphics
Römerstraße 164
D-53117 Bonn, Germany

University of Bonn
Department of Orthopaedics
Sigmund-Freud Straße 25
D-53105 Bonn, Germany

ABSTRACT

In this paper we describe a medical application where we exploit surface properties (measured in form of 3D-Range scans of the human back) to derive a-priori unknown additional properties of the proband, that otherwise can only be acquired using multiple x-ray recordings or volumetric scans as CT or MRI. On the basis of 274 data sets, we perform classification using statistical shape analysis methods. Consistent parameterization and alignment is achieved on the basis of only few anatomic landmarks. As our choice of landmarks is easy to detect on the human body, our approach is feasible for screening applications that can be expected to have much impact on the early detection and later treatment of spine deformities, in particular scoliosis.

Keywords Statistical Shape Analysis, PCA, Medical Assistance, Scoliosis

1 Introduction and Previous Work

Anthropometric investigations offer interesting approaches to determine etiologic factors of trunk deformities in children. Idiopathic scoliosis is the common spine deformity in prepuberal children [AD85]. Some anthropometric parameters are known as risk factors for developing scoliosis or for scoliosis progressing [HKHDL94, LLFP98, NHSP93, NSL+85]. Early detection of these risk factors could help to prevent developing or progressing of scoliosis by early onset of therapy. Therefore there is a need for screening investigations. In previous studies, anthropometric data was collected mostly by manual measurements [LLFP98, NHSP93]. That means that anthropometric studies are time-consuming and require high personnel expenditures. In screening programs we need an efficient perception and evaluation of anthropometrical data without high personnel costs. Due to its non-invasiveness, accuracy and acquisition speed, recording range-images with laser range scanners seems appropriate for such screening applications [SGWS02].

The Journal of WSCG, Vol.13, ISSN 1213-6964
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency–Science Press

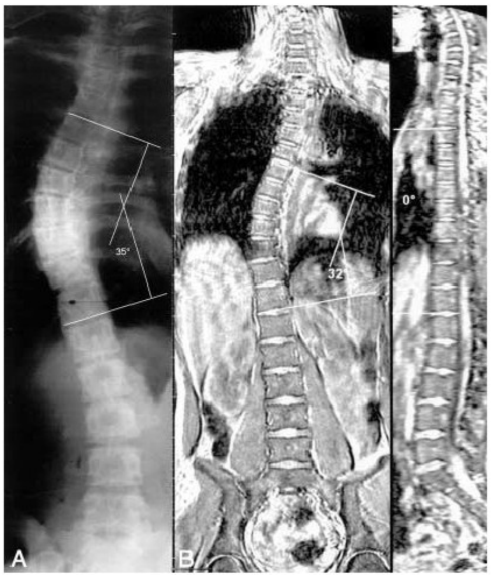


Figure 1: Conventional radiograph (A) and Magnetic resonance (MR) total spine imaging (B,C), exhibiting the flattening effect of probands being in a supine position during recording.[SJK+01]

The non-invasiveness is of particular importance, since X-ray studies to verify clinical findings in patients with scoliosis and other deformities of the spine are associated with considerable radiation exposure as well as a variety of other problems, particularly as regards assessing disease progression. As close monitoring of the scoliosis is required when the greatest growth of the spine occurs, around puberty and early adolescence, there are obvious concerns that repeated radiographs result in an excessive radiation burden, especially to the developing breast tissue in girls. Nash et al. [NGBP79] estimated that 22 radiographic examinations are performed in the course of scoliosis management.

Therefore, there is a necessity for techniques to reduce the frequency in which x-ray recordings have to be made – if not render them unnecessary. In medical applications, there has consequently been an increasing effort replacing the x-ray examination by other techniques. Inter alia, researchers have investigated MRI-techniques [SJK+01] to assess, visualize, and monitor scoliotic spine deformities. Nevertheless these techniques are not always suitable: Due to the expensiveness and time-intensity of the data acquisition procedure this method is not feasible in screening applications. Moreover, during the CT- or MRT-data acquisition process the proband is in a supine position (see figure 1). This way, e.g. leg length discrepancy, a potent cause for postural scoliosis, is not easily detected, whereas apparent if the proband is in an upright position.

Hence, in the course of the past few years a number of alternative, supplementary spinal diagnostic procedures have been developed which are based on analysis of the surface of the back: Photogrammetry/raster stereometry [LHH+98, DH94], opTRImetric system, ISIS system, video raster stereometry (formetrics), ultrasound-guided spine analysis (Zebris) and ultrasound topometry [AMVK00, RS85]. In particular, [DH94] has used structured light to reconstruct the surface of the proband's back, and produced promising results in assessing the degree of scoliosis, although – lacking anatomical landmarks by which the data sets can be robustly aligned – with yet large error margins.

Not only in medical applications, also in the area of computer graphics, creating computable models of the human body or parts thereof has fascinated researchers over the past decades. As the human eye is especially sensitive in detecting unrealistically modelled human bodies, modelling particularly faces from scratch is an almost infeasible task. Therefore, anthropometric data acquired on or from real human beings has been used for modelling. In [DMS98] statistical distribution of a collection of predetermined facial measurements is used to determine the likelihood of a modelled face, thereby effectively restricting the range of allowable models to constraints derived from a set of input faces. Also focussing on faces, Blanz and Vetter introduced the much celebrated morphable face model [BV99]. Key contribution of their approach is deriving a full correspondence between dense polygonal mesh approximations to the faces using texture information and optical flow techniques. With the face meshes in full correspondence, they perform a principal component analysis identifying correlation and the amount of variation contained in the set of input prototype faces. Although faces seem to be of particular interest to the research community, also the whole body has been subject to research [SMT03, ACP03]. Allen et al. [ACP03] present a human body model that was generated using full body scans acquired in the CAESAR project. The main challenge here was to derive the full correspondence between the body scans. To this end, markers were attached to the probands before scanning. Consistent parametrization was then achieved by fitting a predetermined template mesh to the body scan, where the objective function to be minimized during fitting evaluated the misalignment of the given marker point positions as well as the misalignment of automatically detected geometric features.

Our approach is similar to [BV99] and [ACP03] in the sense, that we aim at deriving a model of the human back such that important information concerning the spine deformity can be won from the 3D-surface information only. Nevertheless, focussing on this application field, our approach is conceptually simpler and very easy to implement. Moreover, our approach relies only on the use of few anatomic landmarks to derive both a robust correspondence between surface points and a robust alignment method. A further important aspect is that we, in contrast to previous approaches, exploit machine learning techniques for classification.

The rest of the paper is organized as follows: We will describe the data acquisition process in section 2. The alignment process required to normalize the data before it can be statistically analysed (section 4) is described in detail in section 3. After the presentation of the results achieved with our approach (section 5), the paper is concluded with final remarks and some hints at future directions of research in section 6.

2 Data Acquisition

Our data basis consists of 3D-scans taken from 109 patients, part of which undergoing scoliosis treatment, others only monitoring. Additionally, in a medical screening cooperation with a local school, we have scanned 165 pupils with no known spine deformity (as they have not been undergoing orthopedic examination beforehand).

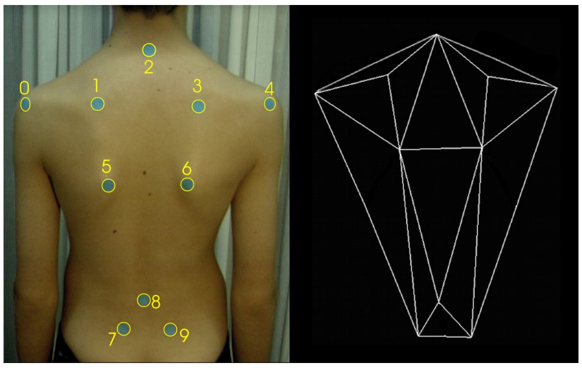


Figure 2: Anatomic landmarks are labelled by an orthopedist. Geometric positions of the landmarks allow consistent coarse mesh generation.

Before scanning, every proband was examined by an orthopedist specialized on spine deformity, who also labelled anatomic landmarks with adhesive markers. These anatomic landmarks (see also figure 2) were chosen for anatomical expressivity and robust detection:

- The spinous process of C7 (2)
- The acromial angle (0,4)
- The superior angle of the scapula (1,3)
- The inferior angle of the scapula (5,6)
- The spinous process of L4 (8)
- The posterior superior iliac spine (7,9)

Note that despite recent advances in 3D-Feature detection the placement of a few marker points to label anatomic landmarks cannot be replaced by automatic feature detection mechanisms as some anatomic landmarks (especially the posterior superior iliac spine and the spinous process of L4) are often covered by soft tissue and are hence not visible in the surface data. This is of particular hindrance in the case of corpulent probands. On the other hand, labelling can be performed not only by specialized physicians but also by trained personnel, such as teachers in schools – a fact that is vital if our system is to be applied in screening applications. During the data acquisition we let physicians do the labelling in order to be able to use their classification statement in the statistical learning stage.
The anatomic landmarks themselves form the vertices for a coarse mesh approximation of the back surface recorded in the range scans. In order to capture the geometric variability contained in the back surface, we construct additional landmarks for our mesh. Following the nomenclature from [DM98], we call these *Pseudo Landmarks*. In order to produce consistently parameterized meshes for the whole set of range images needed for the statistic analysis, we perform semi-uniform subdivision on the coarse mesh (see figure 3), updating the geometry information with information from the range images. Please note that other

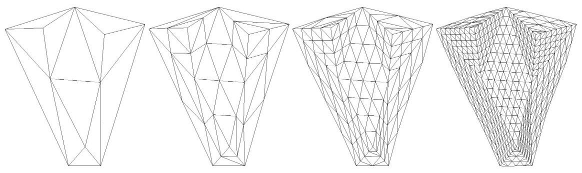


Figure 3: The coarse mesh is semi-uniformly subdivided to produce additional Pseudo Landmarks for the statistical analysis, thereby constructing a consistently parameterized surface approximation.

approaches for mesh re-parametrization as suggested e.g. in [PSS01],[KS04] or [SAPH04] are also feasible at this stage of our algorithm. But, benefitting from the basically planar geometry of the human back, we found this very simple approach of semi-uniform subdivision to be sufficient for our the ensuing application, the statistical analysis. For more complex geometries, e.g. if consistent meshes have to be derived for the entire torso, other strategies will have to be applied. Of course, it is also possible to fit an appropriate template mesh to the range images, as was suggested in [ACP03].

Notation

Suppose we have m data sets (*shapes*). In each data set, we have k corresponding feature points (*landmarks*) in 3-space. Each shape can therefore be represented as an $(k \times 3)$-*shape configuration matrix* $\mathbf{X}_i$, $i = 1, ..., m$, where the j-th row $\mathbf{x}_j^i$, $j = 1, ..., k$ denotes the position of the j-th landmark. The respective components of the landmark vector $\mathbf{x}_j^i$ are denoted by x_j^i, y_j^i, and z_j^i. We suppress the shape index i in case the meaning is clear from the context.

3 Shape Alignment

In order to be able to perform statistical analysis on the shape represented by the landmark coordinates, we need to somehow separate shape variability, that we want to detect, from other sources of variation in the data, e.g. scaling or position in space, that are meaningless for our application. Therefore the input data sets have to be aligned and normalized to make them invariant with respect to the corresponding set of transformations. Although in general this transformation set can be chosen arbitrarily [RDRD04], we choose as invariance set the set of Euclidean similarity transformations, since, according to the shape definition of

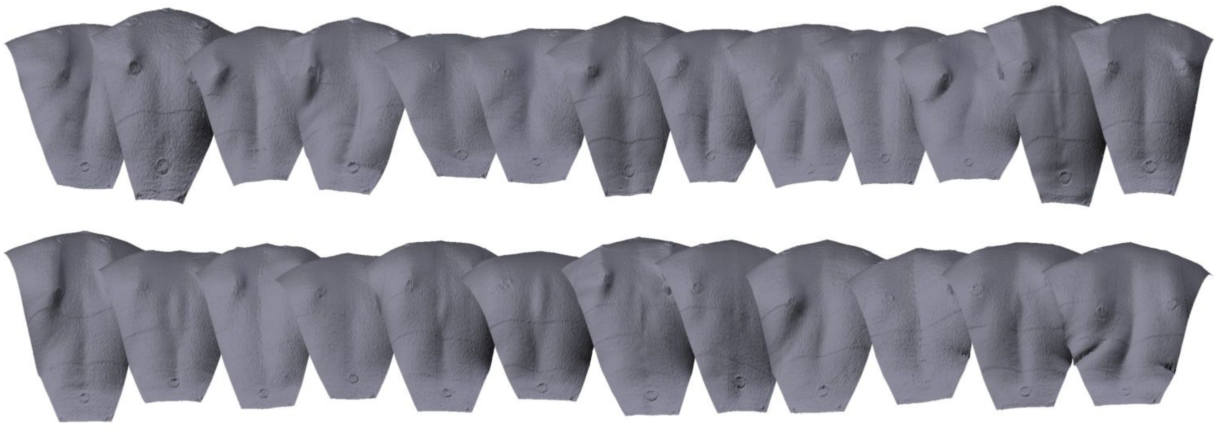


Figure 4: Illustration of the shape space after the reconstruction stage: 25 random examples of the overall 274 reconstructed consistent meshes

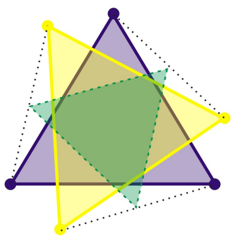


Figure 5: Two identical shapes only differing w.r.t. their rotation (blue and yellow, solid). Without alignment, the mean shape (green, dashed) defined by the arithmetic mean of the respective landmarks would be considerably smaller in size – and even degenerate to a point, had the rotation been about 180 degrees.

Kendall [Ken77], *a shape is all the geometrical information that remains when location, scale, and rotational effects are filtered out*. This means that for each shape $\mathbf{X}$, we have to find an appropriate *scale* $s(\mathbf{X})$, *translation* $\mathbf{d}(\mathbf{X})$, and *rotation* $\mathbf{R}(\mathbf{X})$.
In our algorithm, we will use an alignment approach that combines ideas of two classic alignment approaches, both of which we will shortly describe in the following. For a more thorough covering of alignment approaches, the reader is referred to the extensive literature in this field, e.g. [DM98, Boo86, Sch66], and [Goo91]. A nice introduction is also given in [SG02].

According to Bookstein [Boo84, Boo86] invariance with respect to the Euclidean similarity transformations can be achieved for planar shapes by translating, rotating and scaling each shape such that a pair of landmarks (the so-called baseline) is mapped to predetermined positions. The major drawback of this approach is that it is very sensitive to errors in the baseline landmarks and also, if these are determined automatically, e.g. as points of maximal curvature or as having the maximum distance, to misidentification.

Therefore, a more robust alignment approach has become popular under the name Procrustean Analysis [Sch66]. The basic idea in Procrustean analysis is to find the required similarity transformations through objective function minimization. This objective function can be defined choosing an appropriate shape distance measure and an appropriate reference shape, with respect to which the distance measure is evaluated. One popular choice for the reference shape is the mean shape

$$\overline{\mathbf{X}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{X}_i,$$

where on the right hand side, the $\mathbf{X}_i$ have to be aligned in order to be able to compute the ”true” mean shape. To solve this hen-and-egg problem, defining the reference shape and aligning the shape configurations is usually understood as an iterative process of aligning all data sets to an *estimated* mean shape $\mathbf{Z}$, updating the mean $\overline{\mathbf{X}}$ and iterating:

```
findMean($\mathbf{X}_1, \ldots, \mathbf{X}_m$,Z)
while Z changes do
    for all $i = 1, \ldots, m$ do
        align $\mathbf{X}_i$ with $\mathbf{Z}$;
    end for
    update $\mathbf{Z}$;
end while
```

An obvious choice for the shape distance measure, required to qualify the optimality of a transformation, is the sum of the squared distances between the corre-

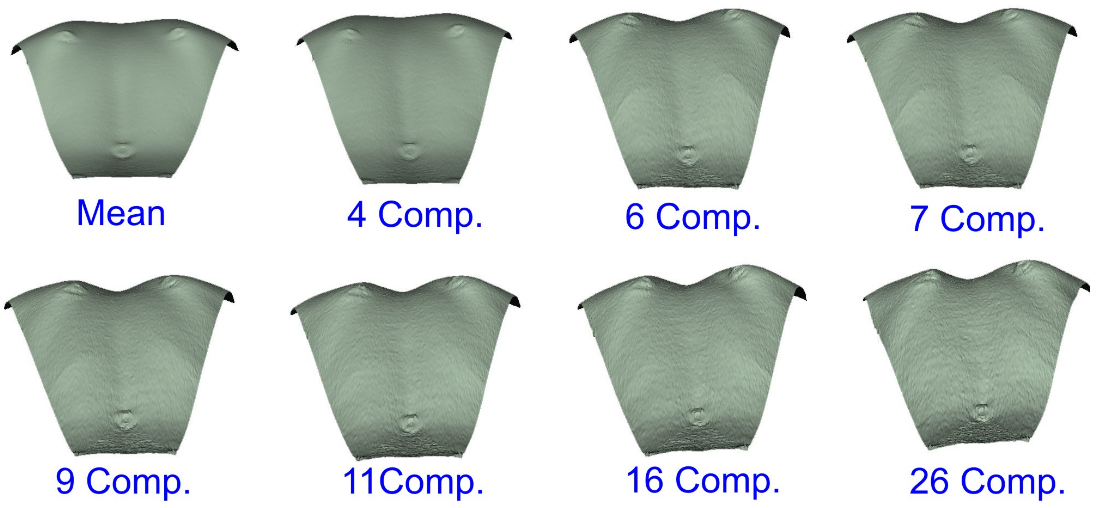


Figure 6: The mean value $\overline{\mathbf{X}}$ and the reconstruction of an example configuration using the denoted numbers of components.

sponding landmarks:

$$D(\mathbf{X}, \mathbf{Y})^2 = \sum_{j=1}^{k} (\mathbf{x}_j - \mathbf{y}_j)^2$$

As this is the same distance measure that is used in the registration of two point sets using the original iterative closest pairs (ICP)-Algorithm, this distance measure leads to a method that is just as susceptive to run into local minima for all but good initial positions. Hence, using this distance measure, shapes have to be roughly pre-aligned, to avoid misalignments. In addition to that, this approach is especially suitable if applied to data sets with uniform landmark confidence, whereas in our case, especially landmarks 2 and 8 (see 2) are of higher confidence compared to the remaining landmarks.
Hence we propose a hybrid approach to compute the similarity transformations given by (s,$\mathbf{d}$,$\mathbf{R}$):

Translation invariance is achieved by moving the centre of gravity to the origin, i.e. for a configuration $\mathbf{X}$ we compute the centroid

$$\mathbf{d}(\mathbf{X}) = \frac{1}{k} \sum_{j=1}^{k} \mathbf{x}_j .$$

This transformation can conveniently be performed by pre-multiplying $\mathbf{X}$ by the $k \times k$-centring matrix $\mathbf{C} = \mathbf{I}_k - \frac{1}{k}\mathbf{1}_k\mathbf{1}_k^T$, where $\mathbf{I}_k$ is the $k \times k$ identity matrix and $\mathbf{1}_k$ is the k-vector of ones.

Gaining **rotation invariance** is a two-stage procedure in our approach: First, each shape is rotated such that the best-fitting plane of the landmarks in three-space (in a least-squares sense) is rotated to the plane defined by $z = 0$. Since the first stage does not yet determine a unique rotation, a second rotation (around the z-axis) is determined for the second stage. Accounting for the varying confidence in the landmarks, we define a generalized bookstein baseline as the best fitting line to the set of points given by

$$\left\{ p_2,\ p_8,\ \frac{1}{2}(p_7 + p_9),\ \frac{1}{6}(p_0 + p_1 + p_3 + p_4 + p_5 + p_6) \right\}$$

(see figure 2). This special baseline selection was motivated by the fact that the landmarks 2 and 8 (spinous process of C7 and L4), and to a lesser extent landmarks 7 and 9 (posterior superior iliac spine) can be detected very confidently and more robustly than the others.
In the second stage, we therefore rotate each shape such that the projection of this baseline to the plane $z = 0$ is rotated to be parallel to the y-axis.
Please note, that the parameters for the described similarity transformations can very conveniently computed by applying a principal component analysis to the set of anatomic landmarks (for the first stage) or to the set of points described above (for the second stage).
Scale invariance is simply obtained by setting the Euclidean distance between landmarks 2 and 8 to be of unit length.

4 Statistical Analysis

After the shape alignment, the set of shape configurations, consisting of the coordinates of the anatomic and the pseudo landmarks, is fit to be analysed by standard statistical analysis methods. In the following, the shapes will be represented as $(3k)$-dimensional column vectors, which are for simplicity also denoted by $\mathbf{X}_i$, $i = 1, \ldots, m$, as they contain exactly the same information as the $(k \times 3)$-configuration matrices.

In order to reduce dimensionality of the data set for ensuing classification steps we perform a principal component analysis (PCA) on the set of configurations.
As a result from the PCA, we get a set of vectors $\mathbf{e}_1, \ldots, \mathbf{e}_{3k}$ with $||\mathbf{e}_i|| = 1$, $\forall i = 1, \ldots, 3k$, and scalars $\lambda_1, \ldots, \lambda_{3k}$ with $\lambda_i \geq \lambda_{i-1}$, $\forall i = 2, \ldots, 3k$ as the eigenvectors and eigenvalues of the corresponding covariance matrix

$$\mathbf{S} = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{X}_i - \overline{\mathbf{X}})(\mathbf{X}_i - \overline{\mathbf{X}})^T,$$

where $\overline{\mathbf{X}}$ is the mean shape (see section 3). The principal components $\mathbf{e}_i$ form a basis of the shape space spanned by the input configurations, and hence we have for any shape configuration $\mathbf{X}$ and a suitable weight vector $\mathbf{w} = \mathbf{w}(\mathbf{X}) \in \mathbb{R}^m$

$$\mathbf{X} = \overline{\mathbf{X}} + \sum_{i=1}^{m} w_i \mathbf{e}_i,$$

leading to $\mathbf{w}(\mathbf{X})$ being an alternative representation of $\mathbf{X}$ in the PCA-space.

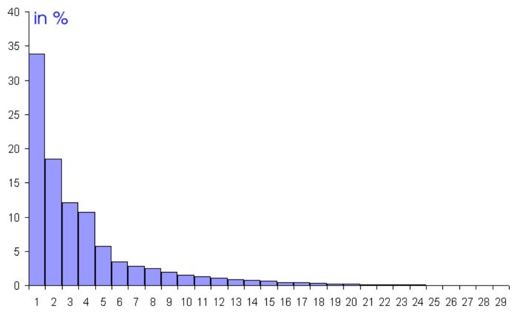


Figure 7: The first 30 main components contribute to over 99 % of the variation in the input data

As can be seen from figure 7 the first 30 components represent already 99 percent of the variation contained in the respective sets (see also figure 6). Therefore, we truncate the weight vectors $\mathbf{w}$ after the 30th component, neglecting the contribution of the principal components $\mathbf{e}_{31}$ to $\mathbf{e}_{3k}$.

Support Vector Machine Approach

Having dramatically reduced the dimension of the data vectors, we are now ready to apply Support Vector Machine classification to our data.

The concept of support vector machines, introduced in [BGV92] and [Vap98] is to find separating planes in high-dimensional vector spaces of labelled sample data. In our setting, the data vectors $(\mathbf{w}, \ell)$ consist of the PCA weight vectors $\mathbf{w}_i$, $i = 1, \ldots, m$ of the back (called *instances*) and appropriate labels $\ell \in -1, 1$ declaring if the corresponding proband was "affected by spine deformity" or "no abnormality detected" (NAD). The basic idea is then, that the *classification function*

$$f : \mathbb{R}^{3k} \mapsto \{-1, 1\}$$

is known for a certain set of instances, called the *learning set*, and unknown otherwise. In our setting, we use linear discriminants a.k.a. *perceptron* as classification function:

$$f(\mathbf{w}) = \langle \mathbf{u}, \mathbf{w} \rangle + b,$$

where $\mathbf{u}$ and b are the parameters that have to be learned from the training examples in the learning set. In addition to that, we have also investigated the effect of decision functions non-linear in $\mathbf{w}$, i.e.

$$f(\mathbf{w}) = \sum_{\nu=1}^{N} \alpha_\nu K(\mathbf{w}_\nu, \mathbf{w}) + b,$$

where N is the number of instances in the learning set and K the radial basis function

$$K(\mathbf{w}_\nu, \mathbf{w}) = \exp(-\gamma ||\mathbf{w}_\nu - \mathbf{w}||^2)$$

with $\gamma > 0$. The decision rule is defined to be $sgn(f)$.
As stated before, we investigated the statistical coherence of an overall set of single shot scans of 274 probands, 109 of which were attending scoliosis consultations, the remaining 165 with no a-priori known spine deformity. All probands have been examined and the data sets have correspondingly been labelled "affected" or "NAD". On the basis of this data, we have performed a cross-validation test [CST03], with a preceding grid search for appropriate parameters, as suggested in [HCL04]. For a detailed description of the maximum margin training algorithm, see [BGV92].

5 Results and Conclusions

In this paper, we have described a medical application in which we exploited range images of the human back to derive a computer aided spine deformity detection system. To this end, we recorded an extensive set of range scans of probands with a small set of marked feature points. These feature point markers represent landmarks that cannot be detected by automatic 3d feature detection, as they are often covered by soft tissue, esp. for corpulent probands, but are easy to be found on the real human body. Using these landmarks for consistent parameterization of the polygonal mesh approximations and for aligning the shapes prior to the statistical analysis, we achieved the good results given in table 1, which is in the order of precision a specialized physician would achieve in a screening application and constitutes an improvement over the current

# Folds	∅Precision	
	linear	rbf
2	92,4812	92,8571
5	92,1053	92,4812
10	92,1053	92,4812
20	92,1053	92,8571
50	92,1053	92,8571

Table 1: Results of the cross validation test using linear or radial basis function-based decision functions. ”#Folds” denotes the number of subsets the set of all instances is divided into. (#Folds-1) of these subsets are used for learning, the remaining 1 for testing. ”∅Precision” gives the average percentage of correctly classified instances.

state-of-the-art. This stresses the feasibility of our approach for screening applications, as the markers can easily be applied by trained personnel (e.g. teachers in school) whereas traditional medical classification has to be performed by specialized physicians.
To separate shape variability from variation in pose or scale, the consistently parameterized data sets are normalized in our approach using a novel alignment procedure that is, while benefitting from ideas both of the so-called Procrustean analysis and the alignment using Bookstein-coordinates, simple in concept and easy to implement.
Although so far we applied statistical analysis in an *inter-proband* manner, i.e. giving insight over ones shape characteristics in comparison to the shape space of human backs, our method can naturally be extended to an *intra-proband* examination: By validating recurrent range scanning of one proband, our morphable back model can be used to assess the impact and effect of scoliosis treatment using braces or surgery, and hence serve as a monitoring tool.

6 Future Work

The results achieved from the classification algorithm are encouraging such that we expect the methods presented in this paper to deliver not only qualitative but also quantitative results. The results also prove that surface topography would reflect Cobb angle[1] status with sufficient reliability, but the error margins achieved in previous approaches [GKM^{+}01] are yet wide. We believe that with our approach, reliability and precision of surface-deduced Cobb angle estimation can be significantly increased.

[1]The Cobb Angle is the classical measure to describe scoliosis quantitatively as depicted in fig. 1.

Acknowledgements

We would like to thank Ruwen Schnabel for fruitful discussions and for his implementation of the SVM classification.

References

[ACP03] Brett Allen, Brian Curless, and Zoran Popovic. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Trans. Graph.*, 22(3):587–594, 2003.

[AD85] I.A. Archer and R.A. Dickson. Stature and idiopathic scoliosis. a prospective study. *Journal of Bone Joint Surg*, 67:185–188, 1985.

[AMVK00] V. Asamoah, H. Mellerowicz, J. Venus, and C. Klockner. Measuring the surface of the back. value in diagnosis of spinal diseases. *Orthopade*, 29:480–489, 2000.

[BGV92] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM Press, 1992.

[Boo84] F.L. Bookstein. A statistical mehod for biological shape comparisons. *Journal of Theoretical Biology*, 107:475–520, 1984.

[Boo86] F.L. Bookstein. Size and shape spaces for landmark data in two dimensions (with discussion). *Statistical Science*, 1:181–242, 1986.

[BV99] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194. ACM Press/Addison-Wesley Publishing Co., 1999.

[CST03] Nello Cristianini and John Shawe-Taylor. Support vector and kernel methods. *Intelligent data analysis*, pages 169–197, 2003.

[DH94] B. Drerup and E. Hierholzer. Back shape measurement using video rasterstereography and three-dimensional reconstruction of spinal shape. *Clinical Biomechanics*, 9(1):28–36, 1994.

[DM98] I. L. Dryden and Kanti V. Mardia. *Statistical Shape Analysis*. John Wiley and Sons, 1998.

[DMS98] Douglas DeCarlo, Dimitris Metaxas, and Matthew Stone. An anthropometric face model using variational techniques. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 67–74. ACM Press, 1998.

[GKM+01] C.J. Goldberg, M. Kaliszer, D.P. Moore, E.E. Fogarty, and F.E. Dowling. Surface topography, cobb angles, and cosmetic change in scoliosis. *Spine*, 26:E55–63., 2001.

[Goo91] C. Goodall. Procrustes methods in the statistical analysis of shape. *Journal Royal Statistical Society Series B-Methodological*, 53(2):285–339, 1991.

[HCL04] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. http://www.csie.ntu.edu.tw/ cjlin/libsvm/index.html, 2004.

[HKHDL94] A.A. Hazebroek-Kampschreur, A. Hofman, A.P. Dijk, and B. Ling. Determinants of trunk abnormalities in adolescence. *Int J Epidemiol*, 23:1242–1247, 1994.

[Ken77] D.G. Kendall. The diffusion of shape. advances in applied probability., 1977.

[KS04] Vladislav Kraevoy and Alla Sheffer. Cross-parameterization and compatible remeshing of 3d models. *ACM Trans. Graph.*, 23(3):861–869, 2004.

[LHH+98] U. Liljenqvist, H. Halm, E. Hierholzer, B. Drerup B, and M. Weiland. 3-dimensional surface measurement of spinal deformities with video rasterstereography. *Z. Orthop Ihre Grenzgeb*, 136:57–64, 1998.

[LLFP98] R. LeBlanc, H. Labelle, F. Forest, and B. Poitras. Morphologic discrimination among healthy subjects and patients with progressive and nonprogressive adolescent idiopathic scoliosis. *Spine*, 23:1109–1115, 1998.

[NGBP79] C.L. Nash, E.C. Gregg, R.H. Brown, and K. Pillai. Risks of exposure to x-rays in patients undergoing long-term treatment for scoliosis. *The Journal of Bone and Joint Surgery*, 61(3):371–374, 1979.

[NHSP93] M. Nissinen, M. Heliovaara, J. Seitsamo, and M. Poussa. Trunk asymmetry, posture, growth, and risk of scoliosis. a three-year follow-up of finnish prepubertal school children. *Spine*, 18:8–13, 1993.

[NSL+85] H. Normelli, J. Sevastik, G. Ljung, S. Aaro, and A.M. Jonsson-Soderstrom. Anthropometric data relating to normal and scoliotic scandinavian girls. *Spine*, 10:123–126, 1985.

[PSS01] Emil Praun, Wim Sweldens, and Peter Schroeder. Consistent mesh parameterizations. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 179–184. ACM Press, 2001.

[RDRD04] B. Romaniuk, M. Desvignes, M. Revenu, and M.-J. Deshayes. Shape variability and spatial relationships modeling in statistical pattern recognition. *Pattern Recognition Letters*, 25(2):239–247, 2004.

[RS85] A. Rohlmann and J. Siraky. Reproducibility of surface measurements of the back using the optrimetric method. *Z Orthop Ihre Grenzgeb*, 123:205–212, 1985.

[SAPH04] John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. Inter-surface mapping. *ACM Trans. Graph.*, 23(3):870–877, 2004.

[Sch66] Peter H. Schoenemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31:1–10, 1966.

[SG02] M. B. Stegmann and D. D. Gomez. A brief introduction to statistical shape analysis, March 2002. Images, annotations and data reports are placed in the enclosed zip-file.

[SGWS02] Alfred Schmitz, H. Gabel, H.R. Weiss, and Ottmar Schmitt. Anthropometric 3d-body scanning in idiopathic scoliosis. *Z Orthop Ihre Grenzgeb*, 140:632–636, 2002.

[SJK+01] Alfred Schmitz, Ursula E. Jaeger, Roy Koenig, Joerg Kandyba, Ulrich A. Wagner, Juergen Giesecke, and Ottmar Schmitt. A new mri technique for imaging scoliosis in the sagittal plane. *European Spine Journal*, 10(2):114–117, April 2001. Issn: 0940-6719 (Paper) 1432-0932 (Online).

[SMT03] Hyewon Seo and Nadia Magnenat-Thalmann. An automatic modeling of human bodies from sizing parameters. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 19–26. ACM Press, 2003.

[Vap98] V.N. Vapnik. Statistical learnig theory. Wiley, New York, 1998.

Broadcast GL: An Alternative Method for Distributing OpenGL API Calls to Multiple Rendering Slaves

Tommi Ilmonen
Helsinki Univ. of Technology
Telecommunications Software and Multimedia Laboratory
Tommi.Ilmonen@tml.hut.fi

Markku Reunanen
Helsinki Univ. of Technology
Telecommunications Software and Multimedia Laboratory
marq@tml.hut.fi

Petteri Kontio
Helsinki Univ. of Technology
Telecommunications Software and Multimedia Laboratory
jpkontio@tml.hut.fi

ABSTRACT

This paper describes the use of UDP/IP broadcast for distributing OpenGL API calls. We present an overview of the system and benchmark its performance against other common distribution methods. The use of network broadcasts makes this approach highly scalable. The method was found effective for applications that need to transmit changing vertex arrays or textures frequently.

Keywords
Distributed rendering, OpenGL, Virtual Reality

1 INTRODUCTION

There are numerous situations where one needs to render the same 3D graphics divided to multiple displays in real time. Figure 1 shows a typical example of a virtual reality (VR) environment with multiple video walls.

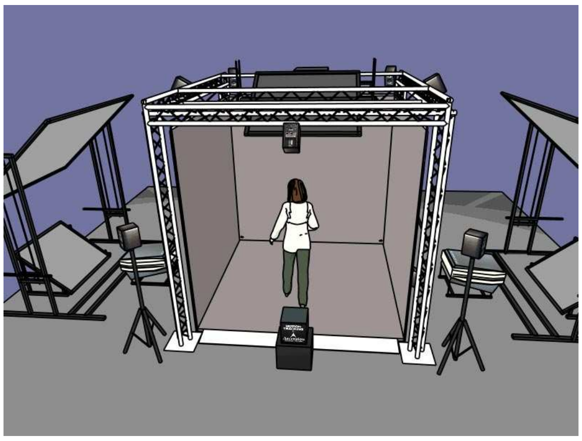


Figure 1. A VR setup with multiple walls

The Journal of WSCG, Vol.13, ISSN 1213–6964
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency – Science Press

Traditionally such situations have been handled by using a single high-performance computer with several graphics outputs. Recently a number of projects have utilized low-cost PC hardware for this purpose — using a cluster of commodity PCs to render all the walls. A similar change from an SGI Onyx2 server to a cluster of commodity PCs was the motivation behind the development of Broadcast GL as well.

2 BACKGROUND

Typically the most efficient way to accomplish high frame rates is to write applications that can be distributed and only send minimal amount of application data to the renderers. In these cases the application copies must produce identical behavior in all situations, which requires the programmer to write the application to support multiple hosts.

This is difficult if the application has a complex internal logic with plenty of user interaction. An alternative method of distributing the application is to spread the graphics API calls (OpenGL, DirectX) to multiple renderers. This is typically rather easy, since a normal 3D application already uses those calls to render its graphics. If these API calls can be distributed effectively to multiple rendering hosts, there is no need to rewrite the application. Since our software uses OpenGL, we are interested in distributing the OpenGL calls (`glVertex3f`, `glBegin`, `glEnd` etc.).

There are already several methods to spread OpenGL calls to multiple renderers. Staadt et al. have written an overview of different methods and analyzed their performance[Sta03a].

- GLX is the standard that is used in most UNIX-based operating systems that support the X windowing system [Wom98a]. GLX-based clustering integrates seamlessly to the windowing environment and it works without additional toolkits. For efficient multi-display rendering the renderer must be parallelized with one rendering thread per display pipe. There are toolkits that manage GLX contexts and set up projections matrices, for example VR Juggler [Jus98a].
- Chromium is a distributed 3D graphics system that uses the OpenGL-API to render graphics on multiple slaves [Hum02a]. Chromium optimizes the network usage by culling primitives before sending them over the network.
- Multi-display systems offered by Hewlett-Packard use a broadcasting method similar to ours. The method is briefly described in [Lef] but no benchmarks or in-depth details are provided. In addition to multi-display systems the architecture has been used in single-display environments to distribute the rendering load between multiple computers.

3 BROADCAST GL

Both GLX and Chromium transmit the rendering commands over a unicast TCP/IP connection. This approach is far from optimal if the same rendering commands need to be spread to multiple slaves. In this case both Chromium and GLX waste network resources by sending the information many times over. An example of such situation is a cluster of PCs rendering multiple walls of a VR installation: all the walls receive almost identical rendering commands, apart from the projection matrices.

Broadcast GL (BGL) solves this problem by using a broadcast technique to transmit the OpenGL API calls. As a result the BGL needs to send the graphics only once and each slave gets a copy of the rendering information.

Besides taking full advantage of the network resources this approach also simplifies the programming work, while the application can be completely single-threaded and still take full advantage of the multiple slaves. This is a relevant detail since most application programmers prefer writing non-threaded code. Potentially difficult problems such as thread synchronization and interlocking are avoided.

With the approach chosen in BGL we can implement only a subset of the OpenGL API. In practice the functions that return some data from the OpenGL system are currently only partially implemented. In theory all OpenGL functionality can be implemented, but the implementation of certain calls would be inefficient. The subset that is implemented works by caching a copy of the data in the application machine.

Due to its architecture BGL has strict requirements about the underlying network architecture. First of the network must support UDP multicast or broadcast. In practice this rules out wide-area networks. The network should also be fast and reliable. In practice these limitations imply the use of a cluster in local-area network with a number of computers connected via a switch.

4 IMPLEMENTATION

BGL uses a client-server architecture (following Staadt's taxonomy [Sta03a]). The application functions as a client that broadcasts BGL command byte stream (binary encoded OpenGL API calls) to the rendering servers over a UDP/IP socket. The slaves are independent rendering applications that receive the BGL byte stream and convert it back into OpenGL API calls. As a return channel each slave has a dedicated TCP/IP connection. BGL overview is shown in Figure 2.

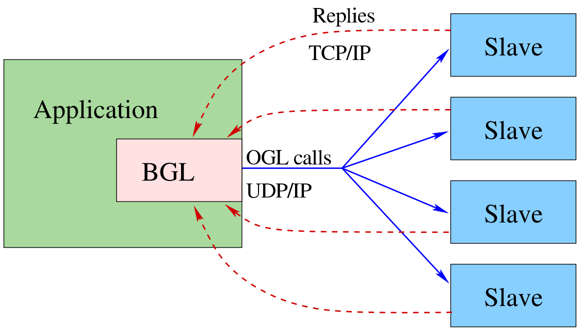


Figure 2. The networking architecture used in BGL. The rendering slaves can be either in the same machine or distributed across the network.

From the application perspective, BGL is little more than an OpenGL implementation, having the network transmission hidden behind the standard OpenGL API. Special BGL calls are used when OpenGL does not define calls that are necessary for applications. Examples of these needs are window handling, buffer swaps and selecting the rendering slave.

The BGL encoder library consists of functions that implement the OpenGL API (`glVertex3f`, `glNormal3f` etc.). The encoding functions store a number of bytes into a local data buffer. The data buffer can contain as much data as the system can fit into a single UDP packet. Once the packet is filled it is sent to the network.

Since OpenGL applications occasionally need to read back variables from the OpenGL implementation, BGL encoder keeps a local copy of some states. In practice this means that the current transformation matrices are kept in the encoder and they can be queried with normal `glGetFloatv`, `glGetDoublev` and `glGetIntegerv` functions.

BGL Specific Functions

The are also special BGL functions, such as OpenGL initialization and buffer swaps, that are needed to control behavior that is outside the basic OpenGL API, but needed by all applications. Below is a list of the BGL-specific functions that are visible to the application programmer.

- `bglInit(const char * address)` — This function initializes the BGL data transmission layer and connects to the slaves using the argument address.
- `bglQuit()` — This function shuts down the slaves and the data transmission layer.
- `bglSwapBuffers()` swaps the OpenGL buffers.
- `bglCreateWindow(int flags)` creates an OpenGL window .
- `bglResizeWindow(int w, int h)` resizes the OpenGL window.
- `bglMoveWindow(int w, int h)` moves the OpenGL window.
- `bglSelectRenderer(int id)` instructs the selected slave(s) to listen to the broadcast.
- `bglDeSelectRenderer(int id)` instructs the selected slave(s) to ignore the broadcast.

A typical way to use the "select" and "deselect" functions is in setting separate transformations for each renderer, for example:

```
// No one is listening now:
bglDeSelectRenderer(-1);
// Slaves with id 1 are listening:
bglSelectRenderer(1);
//Slaves with id 1 and 2 are listening:
bglSelectRenderer(2);
// Translate the geometry in slaves 1 and 2:
glTranslatef(0, 0, 1);
// All slaves are listening again
bglSelectRenderer(-1);
// Now we can render the scene
```

Send & Return Channels

When sending data over a socket we have to choose between UDP/IP and TCP/IP. UDP is a connectionless protocol that does not guarantee that all data that is transmitted gets to target, nor does it guarantee that the data arrives in the correct order. TCP/IP in turn provides a reliable connection, but with higher connection overhead.

In BGL the OpenGL data is sent over a UDP/IP socket since UDP offers lightweight broadcast and multicast features. TCP is used as the return channel protocol since return data rates are much lower, meaning that we can use a slower and more reliable connection.

Replies

If the application sends data at an excessive rate to the slaves it can overflow their UDP buffers, i.e. data arrives faster than it can be consumed. To avoid this the BGL requests replies from the slaves at fixed intervals (equal to "buffer flush" in [Lef]). The slaves then answer that they have received the reply request and once BGL has received all the replies it can continue transmission. For example BGL might send a reply request after sending 16 packets. After transmitting the request, BGL will send a few more packets and then collect the replies from all the slaves. If the replies were collected immediately the renderers would have to empty their buffers before they could receive more data. This asynchronous approach helps us keep a buffer of rendering content in the slaves, resulting in higher performance.

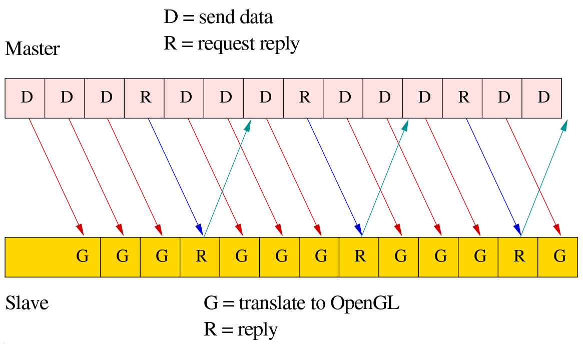


Figure 3. Asynchronous reply mechanism.

The reply system is also used when the application calls functions `glFlush`, `glFinish` or `bglSwapBuffers`. Each of these functions return only after all the slaves have replied. In the case of `bglSwapBuffers` the system first makes sure that all the slaves have done their rendering work and then issues a command to swap buffers.

Scalability

In BGL the data transfers are highly asymmetric. To render one frame the application may send out several megabytes of data, while the renderers' replies use only a fraction of that. The following calculation, which matches the benchmark setup below, gives a real-world example of the asymmetry.

When using UDP packets with 4096 bytes per packet and UPD buffers of 256 kilobytes, BGL application sends reply queries to the renderers at every 19 packets, resulting in 77824 bytes per reply request. Each reply packet uses 4 bytes, thus the downstream traffic takes roughly 19000 times more bandwidth. Since each renderer requires a separate reply connection this ratio is overly optimistic, but even with 1000 renderers the application sends out 19 times more data than it receives. The amount of data sent does not depend on the number of slaves, unless the slaves are controlled individually, as was done in the transformation example above. As long as the used network is reliable, new renderers can be added with minimal performance loss.

Recovering from Transmission Errors

UDP connections are inherently unreliable. The packets can be lost or they may arrive in the wrong order to the recipient. Altough we are using a very reliable network both error cases do occur. Since OpenGL does not tolerate missing commands these errors must be corrected in the transport layer. Both TCP and UDP guarantee the correctness of the transmitted packets, so there's no need to build an additional bit-level error correction mechanism.

BGL uses the TCP return channel to report missing packets. When a renderer receives a packet with unexpected counter value it puts the packet to a store the notifies the master that a packet was missing. The master in turn keeps the latest UDP packets in a ring-buffer and retransmits the missing packet. This error correction is not enough in the cases where a renderer loses multiple packets (including packets with reply commands). To handle these situations the master retransmits packets automatically if the renderers do no reply within a given time interval.

Together these strategies guarantee that the transmission errors are corrected as long as at least some amount of packets reach the renderers. We have tested the system by intentionally losing packets. The error recovery works correctly even when 80 % of all transmitted packets are lost.

Internal Structure

BGL is composed of two parts. The application library (libBGL) implements the OpenGL API and the BGL-specific extra functions. This library contains OpenGL encoding functions and data transport layer. The renderer is a stand-alone application that also includes the transport layer and OpenGL decoding functions.

The OpenGL API has been originally designed to be easily streamable. This makes encoding and decoding the API calls fairly easy. In BGL most OpenGL functions are defined with one-line macros. Writing the encoding and decoding layers took only two days.

The data transport layer is more demanding for the programmer. Finding the most effective way to use network resources took more time than implementation of decoding library. This part is also more easily broken by networking anomalies that may not have been present when the system was first tested.

5 BENCHMARKS

BGL was benchmarked against Chromium and a GLX-based graphics distribution mechanism. The OpenGL distribution platforms are detailed below:

1. GLX-based threaded renderer: This system uses a separate rendering thread for each X11 display, thus rendering two windows per thread. This system is similar to the VR Juggler OpenGL application framework [Jus98a]. Based on informal tests, our GLX-distribution system has performance characteristics similar to the VR Juggler implementation.

2. Chromium: We used Chromium version 1.7. Chromium's tilesort SPU was used for the graphics distribution and the render SPU for viewing the graphics. The tilesort SPU culls polygon faces before sending rendering commands to the network, thus decreasing the network load.

3. BGL: The application was linked with the BGL encoder library and a small projection management library. We used a normal broadcast address `10.0.0.255:10001` to deliver the broadcast from the application to the renderers.

All the described methods were tested in the following three test cases:

1. Display of a real-world architectural model, rendered with display lists. This benchmark represents a typical static model, for example a background scene in computer games. A part of the scene is shown in Figure 4.

2. Display of a real-world architectural model, rendered without display lists, i.e. in immediate mode. This benchmark represents volatile data sets — for example objects under deformation cannot be compiled into display lists.

3. Texture streaming. This benchmark represents a case where texture animation is made by streaming a new (sub)texture into the hardware at each frame. Such approach is commonly used when a video stream is embedded into OpenGL graphics. In our test the size of the RGB texture was 320 by 240 pixels (225 kB). A screenshot of this test is in Figure 5.

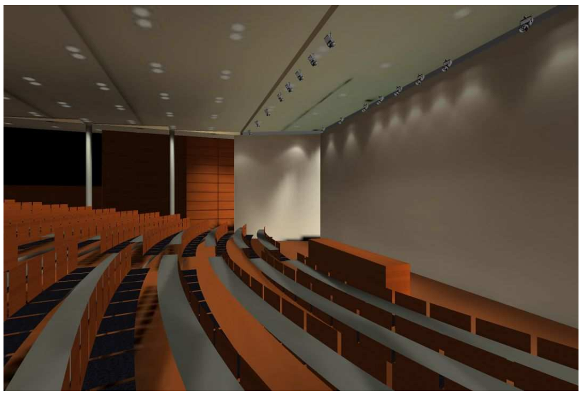


Figure 4. A screenshot of the architectural scene used in tests 1 and 2. The scene has 96733 triangles. All lighting is done with texture maps.

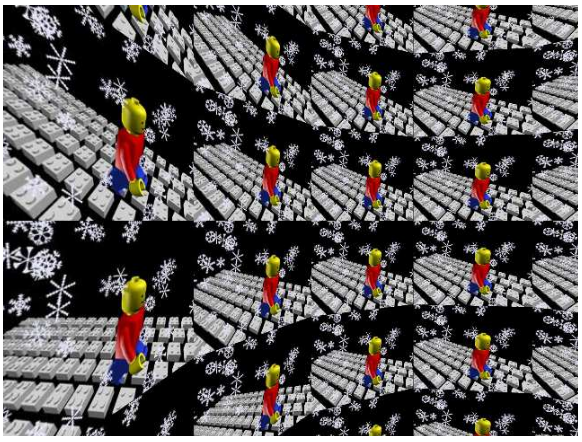


Figure 5. A screenshot of the video player test software.

Tests 2 and 3 are bandwidth-intensive, while test 1 stresses the graphics pipeline. During the tests we measured the following metrics:

1. Frame rate (frames per second, fps)
2. Network traffic in the application computer (megabytes per second)
3. Application computer load (percentage of CPU resources used)

In the test the scene was rendered on four rendering computers. Each computer displayed two separate OpenGL windows, representing the left and right eye views. The window size was 1024 x 1024 pixels. Each window had a different projection matrix, matching a typical four-wall Cave setup similar to Figure 1. Additional tests were run on an SGI Onyx2 system and a single desktop PC. The SGI rendered the graphics into four stereo windows resulting in a render load equal to the PC cluster tests. These tests were ran to compare the performance of the PC cluster to the retiring system. The stand-alone PC in turn rendered the graphics into two windows, providing an estimate of the highest achievable frame rate.

The test setup was composed of five Linux-based computers — an application PC and four rendering machines. Each computer had a 2.8 GHz Intel P4 CPU, an integrated gigabit Ethernet controller and an NVidia FX5900 graphics card. The PCs were running Linux kernels from the series 2.4 and 2.6. The SGI-based system was an Onyx2 with two IR2 pipelines and eight 200 MHz R10000 CPUs. The test results have been collected to Tables 1–3.

Test	Architecture	GLX	Chromium	BGL
1	1 GB	2.8	37	19
	100 MB	1.62	46	15
	PC/Local	16	-	-
	SGI/Local	1.3	-	-
2	1 GB	0.87	2.4	4.2
	100 MB	0.10	0.32	0.82
	PC/Local	24	-	-
	SGI/Local	1.8	-	-
3	1 GB	7.5	10	105
	100 MB	3.1	1.8	24
	PC/Local	150	-	-
	SGI/Local	20	-	-

Table 1. Frame rates for three tests in gigabit and 100 Megabit networks (frames per second).

Test	Network	GLX	Chromium	BGL
1	1 GB	25	4	0.48
	100 MB	0.38	5.3	0.47
2	1 GB	95	87	55
	100 MB	12	12	12
3	1 GB	7.5	46	29
	100 MB	6.4	11	9

Table 2. Network traffic (Megabytes transmitted per second).

Test	Network	GLX	Chromium	BGL
1	1 GB	44	20	2
	100 MB	6	20	2
2	1 GB	70	45	35
	100 MB	10	10	1.5
3	1 GB	6	24	18
	100 MB	6	7.5	5

Table 3. Application computer CPU load.

The CPU load of the application is split into user-space load and kernel-space load. The CPU loads were measured with ”top” -program that is part of standard Unix command set. This measurement is complicated by the fact that the definition of CPU load is not an obvious measure on modern hyper-threading CPU’s. In this case we took the CPU idle time from ”top” and calculated the application load from it. The idle time represents how much time the CPU has left to run other applications. These load values are shown in table 3.

While the above benchmarks measure run-time performance there are other aspects that are important for the application programmer as well. A summary of these aspects has been collected to Table 4.

System	GLX	Chromium	BGL
Network scalability	Poor	Moderate	High
OpenGL compliance	Good	Moderate	Moderate
Ease of programming	Poor*	Good	Good

Table 4. Qualitative differences between different approaches

* Requires threaded rendering into multiple GLX contexts.

It is worth noting that the test setup differs from Staadt’s. We are running a single centralized application with distributed graphics, while Staadt’s tests also included distributed applications [Sta03a].

In addition to the system benchmarks we ran a small-scale scalability test. Test 2 (immediate mode rendering of the architectural model) was run on one to four rendering computers. Tests 1 and 3 were discarded because they were too dependent on pure rendering or network speed and would not have given meaningful results about scalability. The graph shown in Figure 6 displays the frame rates obtained in this test.

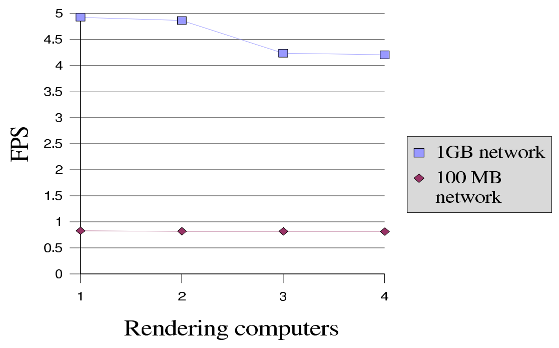


Figure 6. The effect of added rendering computers on the frame rate.

Analysis of the Benchmarks

The benchmarks above show that BGL, in many test cases, outperforms both the standard GLX-based graphics distribution and Chromium. In these tests Chromium could often use its culling algorithms to lower the network traffic. If one thinks about the usage in a fully immersive six-wall Cave, this culling cannot eventually do more than ensure that the same data is not sent from the application to the renderers more than once. Since there are two windows with nearly identical views for each wall, the vertex data will be sent twice unless the software can recognize the overlap.

In test 1 Chromium did extremely well and surpassed even the local GLX rendering. This is apparently due to its heavy culling methods that could discard even complete display lists. BGL proved its scalability by providing approximately the same frame rate as the single PC.

BGL was clearly the fastest system in tests 2 and 3, delivering higher frame rates with lower CPU load and lower network stress. In test 3 both Chromium and GLX were forced to send the texture eight times to the renderers, resulting in roughly eight times more data traffic per frame.

The 100 Megabit Ethernet was easily saturated by all systems. Surprisingly, none of the systems could saturate the gigabit Ethernet in any of the test cases. It seems that the computers have trouble moving data over the network at such high rates. Also it seems that in the renderer computers the OpenGL usage has negative effect on the networking performance, probably because both require bus resources that are mutually exclusive.

The performance of the GLX-based distribution was in most cases disappointing. Especially, one would expect that GLX-distribution would run well with display lists, but this was not the case. This problem might be caused by networking issues or problems within NVidia's GLX implementation. We have experienced similar performace problems when using VR Juggler in our test configuration. When run locally the GLX code worked fine, both in the SGI tests and in the stand-alone PC test.

When we compare the performance of the network rendering against rendering the same graphics locally we can see that with display lists (test 1) the local rendering is in fact slower. In immediate mode (test 2) the local rendering is significantly faster while the video streaming (test 3) application is 50 % faster when ran locally.

The BGL-based PC-cluster outperforms our old SGI-system in all the tests. While this information is not particularly surprising, it created significant confidence to the new platform. The scalability of the system is good (Figure 6). In gigabit network the frame rate dropped only 15 % when the number of renderers was changed from 1 to 4. In the fully saturated 100 MB network the number of renderers made no difference.

6 DISCUSSION

As the BGL implementation matures, it allows for several interesting applications. Because of the scalability of the approach, large rendering clusters can be built without significantly increasing the load of the the application computer. The broadcast graphics can be viewed across the network in different visualization devices, such as an ordinary monitor, head-mounted display or a Cave, whereas for the application code the final output device bears very little importance. The method somewhat resembles the traditional radio and TV broadcasting and could be even used for similar purposes in the form of a "3D television". Large-scale broadcasting for various bandwidths cannot be handled by a single computer, thus creating a need for a proxy or other middleware solution.

The current BGL implementation can store the OpenGL command stream to a file. This feature was created mostly as a debugging aid, but it could also be used as a 3D video format. The resulting files can readily be compressed with ordinary tools such as gzip and even further with more advanced techniques such as texture compression.

In its current state, BGL features only a bare-bone OpenGL implementation. Full OpenGL compliance is in practice difficult to achieve, mainly because the OpenGL state is distributed over a cluster of nodes. Frequent state queries from the nodes is also likely to cause performance loss due to the stalling of the rendering stream.

At the moment one badly behaving renderer can stall the whole cluster. This clearly means that the synchronization should be studied further. We suspect that once the network latency and synchronization are handled better, the overall throughput of the system will increase considerably. The symmetry of the rendering computers is vital to good performance since the slowest node effectively dictates the overall frame rate.

The tests that were conducted did not incorporate genlocking or any synchronization to display updates. This choice was intentional because we wanted to measure the maximum throughput possible with each of the systems. In practise such constraints are often present and slow down the frame rate. For example a double-buffered 100 Hz sychronized display typically limits the steady frame rates to 100, 50, 25 FPS and so on. Hardware genlocking should not affect the frame rate but a software-based approach such as SoftGenLock [All03a] does because it introduces additional system load.

7 CONCLUSIONS

We have presented and evaluated an alternative method to distribute graphics API calls to multiple rendering computers. By using the broadcast/multicast networking we have managed to ensure the same graphics data is not sent more than once across the network, regardless of the number of renderers. The current BGL implementation is far from perfect and we will continue to improve it.

In the light of the benchmark results it seems obvious that none of the OpenGL distribution systems is in all cases better than the others. Rather, the best choice depends on the application, computers used and the network characteristics. Obviously the best use cases for BGL are data-intensive applications that require good scalability to multiple displays. Furthermore, the simple single-thread application logic allows for easy adaptation of existing desktop OpenGL software.

References

[All03a] Allard, J., Gouranton, V., Lamarque, G., Melin, E., Raffin, B. Softgenlock: Active Stereo and Genlock for PC Cluster. in Proceedings of the Joint IPT/EGVE'03 Workshop, Zurich, Switzerland, May 2003.

[Hum02a] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P.D., Klosowski, J.T. Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. in ACM

Transactions on Graphics (TOG) , Proceedings of the 29th annual conference on Computer graphics and interactive techniques, Volume 21, Issue 3, 2002.

[Jus98a] Just, C., Bierbaum, A., Baker, A., and Cruz-Neira, C. VR Juggler: A Framework for Virtual Reality Development. 2nd Immersive Projection Technology Workshop (IPT98), Ames, Iowa, May 1998.

[Lef] Lefebre, K. An Exploration of the Architecture Behind HP's New Immersive Visualization Solutions. Hewlett-Packard Company.

[Sta03a] Staadt, O.G., Walker, J., Nuber, C., Hamann, B. A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. in Proceedings of the workshop on Virtual environments 2003.

[Wom98a] Womack, P., Leech, J. (eds.). OpenGL Graphics with the X Window System. Version 1.3, October 19, 1998.

Efficiently Keeping an Optimal Stripification over a CLOD Mesh

Massimiliano B. Porcu
Dip.to Matematica e Informatica
Università di Cagliari
Via Ospedale, 72
I-09124, Cagliari, Italy
massi@dsf.unica.it

Nicola Sanna
Dip.to Matematica e Informatica
Università di Cagliari
Via Ospedale, 72
I-09124, Cagliari, Italy
nsanna@unica.it

Riccardo Scateni
Dip.to Matematica e Informatica
Università di Cagliari
Via Ospedale, 72
I-09124, Cagliari, Italy
riccardo@unica.it

ABSTRACT

In this paper we present an algorithm of simple implementation but very effective that guarantees to keep an optimal stripification (in term of frames per seconds) over a progressive mesh. The algorithm builds on-the-fly the stripification on a mesh at a selected level-of-details (LOD) using the stripifications built, during a pre-processing stage, at the lowest and highest LODs. To reach this goal the algorithm uses two different operations on the dual graph of the mesh: when the user changes the mesh resolution the mesh+strips local configuration is looked up in a table and, after a vertex split operation, the strips are rearranged accordingly, immediately after a sequence of special topological operation called "tunneling" with short tunnel length are started till the number of isolated triangles in the mesh get under 10% of the total number of strips. Moreover, when the user select a relevant LOD it can trigger a tunnelling with higher tunnel length to optimize the stripification. Using these operations we are able to keep the progressive mesh stripified in a time of the same order of magnitude of the time needed to change the resolution and, only if required, to perform a time-demanding optimization. Only the stripifications generated by explicit user requests are stored to serve as optimal starting points for further inspection. In this way we can always feed the graphics board with a triangle strip representation of the mesh at any LOD.

The results we present demonstrate that we can tightly couple each sequence of vertex splits used to increase the resolution of the progressive mesh with: a simple rearrangement of the strips followed by a very cheap stripification search with a predetermined strategy. A strong feature of the method is that the local rearrangement leads to an implementation that keeps almost constant the execution time. The results of the visualization benchmarks are very good: comparing the rendering of the stripified (using this strategy) and the non stripified meshes we can, on average, double the frames per seconds rate.

Keywords

Geometry compression – Stripification – Progressive meshes.

1. Introduction

Three different lines of research are active in trying to improve the management of large meshes: developing efficient algorithms for the compression of the meshes representation; improving the methods for the construction of a multiresolution data structure and easily select a mesh among all the ones stored in the structure; developing efficient ways to best render these meshes on current computer graphics hardware.

A good example of the first type of investigation is represented by the Edgebreaker algorithm and all its improvements [Tau98, Ros99, Paj00, Gan02]. This kind of algorithms allow to lossless encode meshes and collection of meshes (simplicial complexes) of any type using a reduced number (even less than two) of bits per vertex. The methods start from a seed triangle and grow on the free frontier (the boundary with other triangles not already encoded) till all the triangles are encoded.

The most popular method for building multiresolution structure is the progressive mesh method (PM) and all its improvements [Hop96, Hop97, Pop97, Hop98, Paj00]. Its great popularity derives also from the fact to be available as part of Microsoft's DirectX since the release 5.0.

The Journal of WSCG, Vol.13, ISSN 1213-6964
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency – Science Press

Another way to try to compress the geometrical representation of a triangle mesh is the attempt to reduce the throughput between the CPU and the Graphics Processing Unit (GPU). The most common and diffused way to reach this goal is to rearrange the information describing each single triangle in the mesh in structured forms as the triangle strips and the triangle fans [Hae03].

A triangle strip is a set of connected triangles where a new vertex implicitly defines a new triangle. Triangle strips are used to accelerate the rendering of objects represented as triangle meshes, in a pre-processing stage the mesh is partitioned in a set of triangle strips (each of one can possibly be composed of one single triangle) and then each strip is passed to the GPU for rendering. The advantage of the strip representation over rendering each triangle separately, is that it makes it theoretically possible to reduce the number of vertices sent to the GPU from *3n* (where *n* is the number of triangles in the mesh) to *n+2* in the best case.

In this work we couple a selection and a stripification technique: the choice of a LOD over a PM with a method to accelerate its rendering.

The rest of this work is organized as follows: in section 2 we briefly go over the previous work done in geometry compression, focusing on selection methods and stripification algorithms; we then show, in section 3, the relations existing between the triangle mesh and its dual graph, introduce the tunnelling operator and explain how to build a stripification over the lowest resolution LOD of a progressive mesh; section 4 is dedicated to detail how the stripification is kept consistent while varying the LOD in the progressive mesh; in section 5 we show the results we obtained using our algorithm on a mesh we acquired from cultural heritage manufacts; and, finally, in section 0 we draw our conclusions and describe the future evolutions of this work.

2. Previous Work

Deering [Dee95] was the first to introduce the term *geometry compression,* to describe a set of techniques capable of reducing the space occupancy of a *generalized triangle mesh* statistically encoding XYZ positions, RGB colors and normals. These techniques operate mainly on the *geometry* of the mesh (i.e., the positions and the attributes of the vertices) relying on the triangle mesh structure to compress the information on the *topology* (i.e., how the vertices are connected to form the triangles). Since the goal of the work was to suggest a series of different operations the designer can perform to reduce the space occupancy of a triangle mesh, there wasn't any conclusion on the real possibility to move on the graphics board some of these stages.

Even if it is quite a rough classification, since there can be found many mixing approaches, we can divide the geometry compression methods in three main families:

- **Compression methods**. Allow to reduce the data needed to represent a mesh; they are well suited for transmitting and/or archiving the meshes;
- **Selection methods**. Allow to select the resolution that best fits the graphics hardware available for rendering; they are well suited for transmission with a preview effect; they are used to select a representation of the object described by the mesh, given a triangle/frame-rate budget;
- **Rendering accelerating methods** Allow to reduce the time spent in sending the information describing the mesh from the CPU to the GPU thus resulting in getting a higher frames per second rate without changing the number of triangles of the mesh (its geometry).

Compression Methods

After Deering [Dee95] several subsequent works [Tau98, Tou98], centered their attention on the problem of compressing the description of the topology arriving at a relevant result with the Edgebreaker method [Ros99, RsS99] which claims to reach less than two bits per triangle to encode a planar mesh homeomorphic to a disc.

All these techniques need a decompression stage that is not yet implemented in commercial graphics hardware, even using new programmable boards. This means that they are very efficient for transmission and archiving but cannot be used for feeding the GPU.

It is worth to mention that a useful consequence of the Edgebreaker encoding is the easy production of triangle strips while processing and decoding the compressed dataset [RsS99].

Selection Methods

Many authors presented solutions to generate multiresolution structures from an original mesh allowing the user to select a given LOD. We just limit ourselves to remind it's possible to divide the methods presenting a fixed number of LODs (usually less than ten) from the methods ranging on a continuous variation of LODs (CLODs).

Even if we don't want to rehearse all these works let's just briefly remind the main characteristics of the one we used in our implementation.

Progressive meshes (PM) represent the most popular type of continuous LOD meshes. They allow the users to easily encode a complex mesh using a single topological operation (Figure 1) called *edge collapse* (EC) and its complement, *vertex split* (VS). On the PM is possible to perform two different but equally important tasks: to select the representation best fit for the available hardware, and to progressively transmit the mesh.

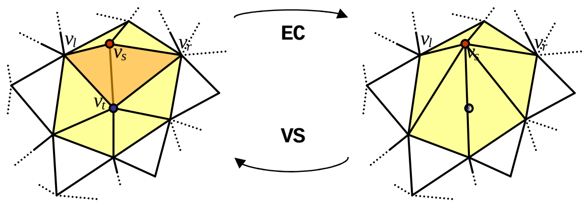


Figure 1: The two complementary operations performed on a progressive mesh

The original proposal [Hop96] has been refined during the last years: a hybrid compression and selection scheme trying to get the best of Edgebreaker compression and progressive meshes [Paj00], a further improvement, in term of bits per vertex [All01], and the extension of these techniques to arbitrary simplicial complexes [Gan02].

In our implementation we built a PM representation from the original following the longest edge rule: we collapse edges in order of decreasing length. We decided to use such a simple approach since the pre-processing in which we build the PM can be changed without affecting the rest of the process and, at this stage of development, we wanted to focus on the stripification scheme.

Stripification Techniques

The greatest advantage in using triangle strips consists of the availability of such a primitive in the OpenGL graphics library. Generating the stripification of a mesh means to be able to feed the GPU with the obtained structure without any further effort. It is actually to point out that OpenGL supports, without any vertex replication, only the sequential triangle strips. Generalized strips could thus bring to send more than once some vertices to the GPU.

Rearranging the order in which the vertices are stored is the typical way to face the problem of reducing the CPU-GPU throughput. The strips obtained are smaller than the original mesh when coming to the final rendering since, while the single triangle needs 3 vertices for its visualization to be sent to the GPU, the sequential triangle strip needs $n+2$ vertices to be sent to the GPU to render n triangles, and the generalized triangle strip $n+s+2$ where s is the number of swaps. The optimal single sequential strip encoding the whole mesh would reduce the number of vertices sent to the GPU by a factor of three.

Several papers illustrate geometrical and topological properties of a stripification [Ark96] and many variations of algorithms to partition a triangle mesh in strips [Eva96, Cho97, Ise01, Est02]. The most relevant work coupling multiresolution structure and stripification techniques is due to El-Sana et al. [ElS99, ElS00]. They introduce a data structure called Skip Strip that is used to generate the triangles strips. The method maintains a stripified progressive mesh during the refinement and coarsening process. This is an approach similar to the one we propose, but it relies on a much more complex data structure.

Working on the Dual Graph

Each triangle mesh can be alternatively represented by its *dual graph*. It is a graph in which each node is associated to a triangle of the original mesh and an edge represents an adjacency relation. One trivial property of such a graph is that each node has, at most, three incident arcs. In case the original mesh is homeomorphic to a sphere and has genus 1, each node has exactly three incident arcs.

It is quite common to use this representation to elaborate stripification algorithms: it allows to use a regular and compact data structure to represent the mesh and one can use all the results obtained from the graph theory. Unfortunately it has been proven [Ark96,. Gar76] that a problem equivalent to searching the optimal single strip (finding a Hamiltonian path on the dual graph) is an NP-complete problem, thus the stripification process should be based on local heuristics.

Two approaches for finding a stripification on the mesh's dual graph have been proposed: one is to compute a spanning tree on the dual graph, partition it into triangle strips, and then concatenate these strips into larger ones [Tou98], the second one is the so-called tunnelling algorithm and it is explained in detail in section 3.

3. Triangle Strip over the Progressive Mesh

Let us first briefly summarize the steps our method performs to keep the stripification at its best. They are:

1. Build the PM over the given mesh;
2. Build the stripification on the lowest LOD meshes using the procedure detailed in section 3;
3. Move over the PM performing either a vertex split operation (VS) on the mesh to increase the LOD or an edge collapse (EC) to decrease the LOD;
4. Rearrange the stripification using topological operations described in section 4;
5. Minimize the isolated triangles generated at the previous step using the tunnelling algorithm with short paths;
6. Build an optimal stripification using the tunnelling algorithm with longer paths, on demand and store it in the stripification data structure.

The step number *1* and *2* are pre-processing steps, we perform them on the mesh and then we can store the results in two supplementary data files, one for the PM and another one for the stripifications.

The Tunnelling Algorithm

The tunnelling algorithm, as initially proposed by Stewart [Ste01] and substantially improved by Porcu and Scateni [Por03], performs the stripification of the mesh using a simple topological operation on its dual graph.

To do so we need to think the graph edges as *colored* in two possible ways (see Figure 2):

- **solid edges** linking nodes associated to triangles in the same strip;
- **dashed edges** linking nodes associated to adjacent triangles not belonging to the same strip.

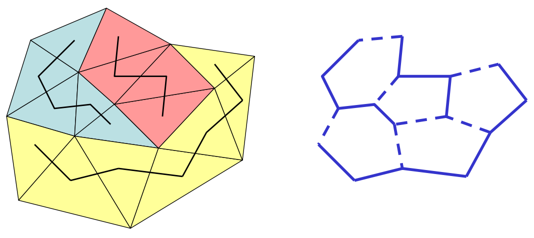


Figure 2: A stripified mesh (each color encodes a different strip) and its dual graph.

In every node there are, at most, two incident solid edges. The nodes with only one incident solid edge are *terminal nodes* (corresponding to terminal triangles of the stripification). The nodes with three incident dashed nodes correspond to isolated triangles in the stripification.

The first step of the operation consists, then, of searching a special kind of path in the graph called a *tunnel*. A tunnel is an alternating sequence of solid and dashed edges, starting and ending with a dashed edge, connecting two terminal nodes. Its length is always odd and we denote by *k*-tunnel a tunnel of length *k*.

If a tunnel is found, the second step consists, simply, of complementing the path, that is, changing each solid edge in a dashed edge and vice-versa. After this operation the number of solid paths (strips in the triangulation) on the graph is reduced by one. See Figure 3 for example.

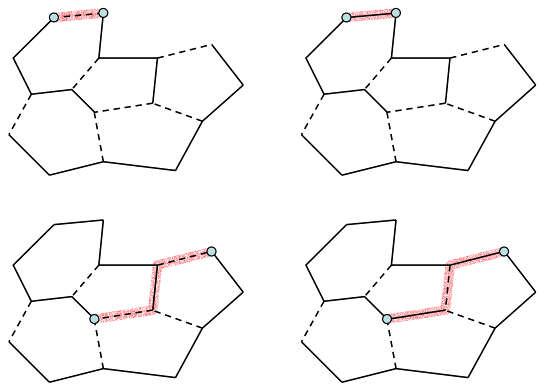


Figure 3: An example of tunnelling. In the top row a 1-tunnel is found; in the bottom row there are no 1-tunnels but only a 3-tunnel. Notice that the number of strips decreases from three to two after the first operation and to one after the second.

This technique can be used both to improve an existing stripification or to create a stripification from scratch. In the latter case the starting dual graph will have only dashed edges and every path of length one can be chosen as a tunnel. It is worthwhile to point out that isolated triangles are always considered as terminal nodes of a one-triangle strip.

The main problem when implementing the algorithm is the possibility that the graph traversal for tunnelling could select paths that, when complemented, would generate loops. It is thus necessary to follow two additional rules (we call them the *no-loop rules*) during the tunnel search to avoid this situation:

1. The last edge in a tunnel cannot connect two nodes belonging to the same strip (see Figure 4).
2. When a non-final dashed edge, *e* say, in the tunnel joins two nodes belonging to the same strip, the next solid edge should go back in the direction of the leading node of *e* (see Figure 5).

To be able to respect the no-loop rules, we need to distinguish between the different strips in the graph. This is done tagging each node of the graph (triangle) with an identifier corresponding to the strips it belongs to.

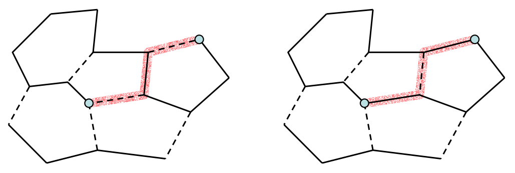


Figure 4: An incorrect tunnelling that generates a loop.

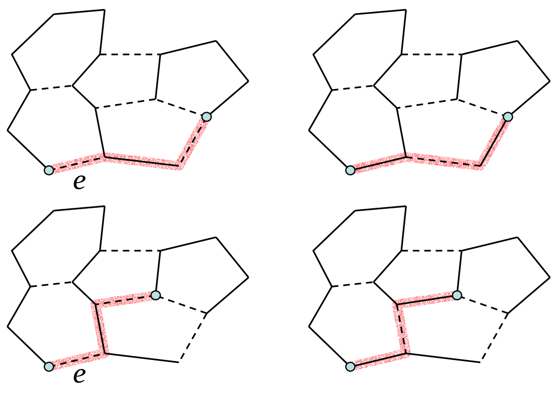
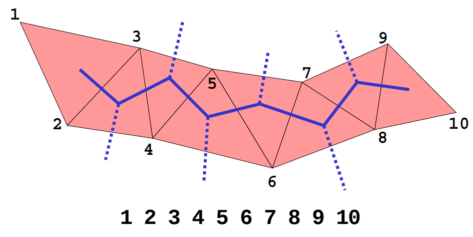


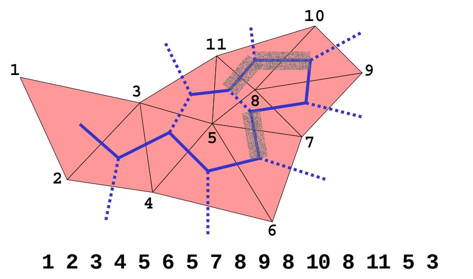
Figure 5: The non-final edge *e* in the tunnel joins two nodes belonging to the same strip. Of the two next possible steps, we must select the one corresponding to the direction that comes back to the leading node of *e* (bottom row), otherwise it will generate a loop (top row). One such step always exists because the leading and trailing nodes of e are in the same strip.

The only minor drawback of the tunnelling algorithm is that we are not able to keep the strips sequential, we are forced to use generalized strips and then introduce swap operations. This is to the fact that by its definition, the tunnelling operation *change the turns in the graph*.

A sequential strip (Figure 6.a) is, in the dual graph, a path of solid edges in which, at each node, we alternatively make a left and right turn. When tunnelling over the graph it is not possible to keep the strips sequential and we are thus obliged to use generalized strips (Figure 6.b).



(a)



(b)

Figure 6:A sequential strip (a) and a generalized one (b). For both we list the vertices to be sent to the CPU. The extra vertices (swaps) are in red. The grey edges mark the *wrong turns*.

4. Strip Rearrangement

The core of the algorithm is the rearrangement (a graph expansion or contraction) of the stripification when changing LOD. There are two method to consecutive operations applied to recolor the augmented graph: the first one is totally local to the triangle loop where the new vertex has been inserted and uses a look-up table; the second is *glocal*, it consists in launching a tunnelling on the modified stripification with a predefined stop rule.

First Step: Using a Look-up Table

We classified many different configurations that can be used to restructure the strips after a VS. Each single VS split operation insert two new triangles in the mesh, and three new edges in the dual graph.

We actually completed the task only for 4-vertices (loops of length 4 in the dual graph), where the VS can lead to two different topologies: two 4-vertices (two 4-loops sharing an edge) or a 3-vertex plus a 5-vertex (a 3-loop and a 5-loop sharing an edge). In this case all the possible configuration (*9*+*9*) allow graph recoloring without isolated triangles. In Figure 11 we list the nine configurations of the 4-loop transforming in a 3-loop plus 5-loop. Each couple of new triangles can be assigned to a single triangle strip, increasing its size by two. As it is possible to notice from the figure the strip section added is always a sequential one.

When dealing with higher degree vertices (longer loops) the cases increase rapidly. Splitting a 6-vertex, the most commonly found in triangular mesh, can lead to three different topologies: a 3-vertex plus a 7-vertex, a 4-vertex plus a 6-vertex and two 5-vertices. The problem is that, in this case, we are no longer able to always recolor the graph without leaving isolated triangles. We can see in Figure 7 a split with complete recolor while in Figure 8 there is a split leaving an isolated triangle. With 8-vertices, that appear very seldom in triangular meshes, we can be obliged even to leave both the inserted triangles isolated.

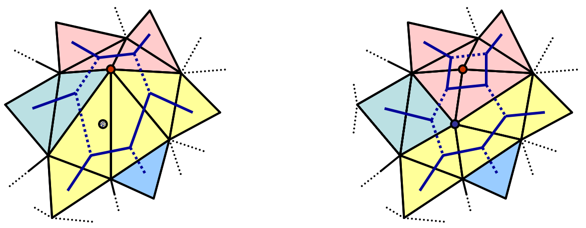


Figure 7: An example of possible graph recoloring after a VS.

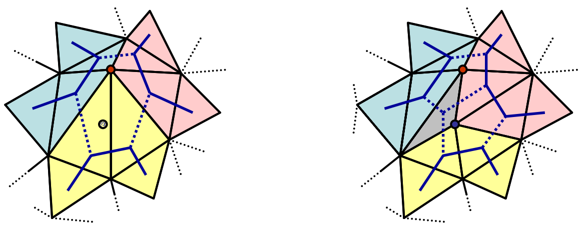


Figure 8: A configuration where the graph recoloring after a VS leaves an isolated triangle (marked grey).

In Figure 9 we can appreciate how the mechanism works. Passing from a LOD to a finer one the strip form stays more or less the same while its average length increases and a lot of isolated triangles appear.

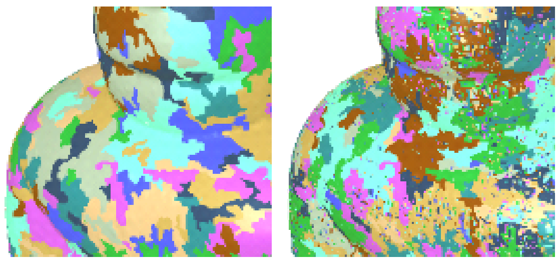


Figure 9: A close-up view of a local rearrangement performed on the *Dea madre* dataset.

Second Step: Using the Tunnelling Operator

Extensive benchmarks performed over different datasets, of different genus and size resulted in a percentage of recoloring operations introducing isolated triangles quite constant: it varies in the range 45%-50%. In other words this corresponds to the insertion of an isolated triangle every second VS operation.

We thus need to repair the strips structure using what we have called a *glocal* tunnelling. The tunnelling operation is performed transparently from the user, and uses the isolated triangles as seeds for searching very short tunnels (starting from 1-tunnels). Since we apply the global operator in a local surrounding of these triangles we can say that it is used glocally. The tunnelling is then iterated until the number of isolated triangles reach a number that is smaller than 10\% of the total number of strips. This value is quite empirical: we noticed that when reaching this ratio, the frames per second rate almost doubles at any resolution for any dataset we used.

In Figure 10 we can appreciate how the rearrangement via the tunnelling works. The strips are completely restructured, there are many less isolated triangles and the average length increase while the maximal length tends to decrease.

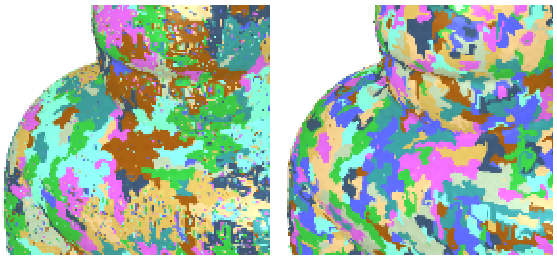


Figure 10: A close-up view of a rearrangement performed on the *Dea madre* dataset using a tunneling operation.

At any stage the user has the possibility to invoke the stripification process explicitly, specifying the maximal tunnel length. We decided to leave this possibility more for completeness than for real need. It is, in fact, quite difficult to significantly improve the results obtained automatically.

5. Results and Discussion

We have performed all our benchmark on a PC with a Pentium IV 1.5 GHz CPU with 512 MB of RAM, and a NVIDIA GeForce TI 4600 GPU with 128 MB of RAM.

For sake of simplicity we present here only the results obtained on the largest dataset we used.

In Table 1 we list the characteristics of the obtained stripifications. We can notice that the number of isolated triangles depends more on the tunnel length than on the overall number of triangles in the mesh.

LOD%	**Only local**	Maximal tunnel length		
		5	**9**	**13**
13	6.139 4.456 12.11	1.676 489 44.38	841 140 88.44	489 46 152.10
27	17.592 9.017 8.77	3.025 496 51.04	1.536 141 88.13	869 52 147.59
52	27.766 11.741 10.70	5.115 785 58.13	2.617 281 113.62	1.489 107 199.70

Table 1: In each cell the first row shows the number of strips, the second the number of isolated tri's and the last the mean strip length.

In Table 2 we show the time, in seconds, used to refine the stripification obtained with only local refinement. We remind that the cost of the local refinement is included in the cost of performing a resolution change.

LOD%	Maximal tunnel length		
	5	**9**	**13**
13	6.484	4.438	5.312
27	4.360	1.750	2.579
52	8.750	5.155	4.156

Table 2: Time in seconds to refine the stripifications.

Average Cache Miss Ratio

The number of vertex cache misses plays a fundamental role in rendering efficiency [Dee95]. If we want to achieve a good rendering sequence than, the Average Cache Miss Ratio (ACMR), whose value ranges from *0.5* to *3*, should be kept as low as possible. To get this goal, several reordering algorithms has been proposed, for standard meshes [Cho97], triangle stripes [Hop99] and progressive meshes [Bog02].

We evaluated ACMR for several data set, using stripes generated with our system using the tunnelling algorithm. Without any kind of reordering mechanism, ACMR is *~0.7* for a cache of 32 positions for all data sets, compared to a typical ACMR of *~1.0* for standard stripification procedures. This suggests that stripes calculated with tunnelling algorithm have a *built-in* cache friendly attitude.

In Table 3 several ACMR values for different data sets are listed, depending on cache size.

The tunnelling algorithm behaves well because of the stripes' shape. As one can see, for instance, in Figure 12, stripes appear to be *packed* instead of being elongated as usual. This preserves locality also in vertex ordering and then cache friendly behavior.

Data Set	Cache size		
	16	**32**	**64**
Oilpump	0.77	0.70	0.64
David	0.78	0.71	0.68
Dea Madre	0.78	0.71	0.67

Table 3: ACMR values for three different data sets based on the cache size.

6. Conclusions and Future Work

We presented a simple but very effective algorithm allowing to compute an optimal stripification on a progressive mesh. Optimal, in this context, means to at least double the frames per second rate with respect to non stripified mesh.

The method we used is a two steps one: first we recolor the dual graph of the mesh using a look-up table and then we transparently launch a tunneling algorithm with a short tunnel length.

We are already planning to get a better insight about the look-up table. As we already mentioned we are not able to automatically avoid the creation of isolated triangles only looking at the strips passing through the loop of triangles sharing the vertex to split. We think that extending the analysis also to the neighbor triangles (say, the triangles that can be reached from the split vertex traversing two edges) can help to increase the number of recoloring without creation of isolated triangles.

Another line of development regards a better analysis of the capabilities of vertex arrays on the GPU. At present we don't clip the triangle strips in chunks the best fit on the arrays and we should insert a further parameter in the visualization tool to take this into account.

The last improvement we are planning is on the fine tuning of the rendering. At present we can select the LOD and then verify the fps obtainable. In the next release it will be possible to set the fps budget and let the system select the possible LOD visualizable.

Acknowledgements

The *Dea madre* dataset was obtained from tridimensional scans of manufacts exposed at the Museo Archeologico Nazionale in Cagliari. We are indebted to its director, Carlo Tronchetti, for letting us use these digital data and to the VCG of the ISTI-CNR in Pisa for the hardware and software used in the acquisition and reconstruction.

We thank Daniele Vacca for his work on the visualization tool.

7. References

[All01] Alliez P. and Desbrun M. *Progressive compression for lossless transmission of triangle meshes*. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pages 195–202. ACM Press, 2001.

[Ark96] Arkin E. M., Held M., Mitchell J. S. B., and Skiena, S. S. *Hamiltonian triangulations for fast rendering*. The Visual Computer 12, 9 (1996), 429–444.

[Bog02] Bogomjakov A. and Gotsman, C. *Universal Rendering Sequences for Transparent Vertex Caching of Progressive Meshes*. In Computer Graphics Forum 21 , 2 (June2002)

[Cho97] Chow M. M. *Optimized geometry compression for real-time rendering*. In IEEE Visualization ’97 (Nov. 1997), pp. 346–354.

[Dee95] Deering, M.F. *Geometry Compression*. In Proceedings of SIGGRAPH 95, pp. 13–20.

[ElS00] El-Sana, J., Evans, F., Kalaiah, A., Varshney, A., Skiena, S., and Azanli, E. *Efficiently computing and updating triangle strips for real-time rendering*. Computer-Aided Design 32, 13 (Oct. 2000), 753–772.

[ElS99] El-Sana J. A., Azanli E., and Varshney A. *Skip strips: Maintaining triangle strips for view dependent rendering*. In IEEE Visualization ’99 (Oct. 1999), pp. 131–138.

[Est02] Estkowski R., Mitchell J. S. B., and Xiang, X. *Optimal decomposition of polygonal models into triangle strips*. In Proceedings of the eighteenth annual symposium on Computational geometry (2002), ACM Press, pp. 254–263.

[Eva96] Evans F., Skiena S. S., and Varshney A. *Optimizing triangle strips for fast rendering*. In IEEE Visualization ’96 (Oct. 1996), pp. 319–326.

[Gan02] Gandoin P.M. and Devillers O. PROGRESSIVE LOSSLESS COMPRESSION OF ARBITRARY SIMPLICIAL COMPLEXES. In 2002 Proceedings of the 29th annual conference on Computer graphics and interactive techniques, San Antonio, Texas, ACM Press, pp. 372–379.

[Gar76] Garey M. R., Johnson D. S., and Tarjan R. E. *The planar hamiltonian circuit problem is NP-complete*. SIAM Journal of Computing 5, 4 (Dec 1976), 704–714.

[Hae03] Haeyoung L., Desbrun M. and Schröder, P. Progressive encoding of complex isosurfaces. In ACM Trans. Graph. 22, 3, pp. 471–476.

[Hop96] Hoppe H. *Progressive meshes*. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996, ACM Press, pp. 99–108.

[Hop97] Hoppe, H. *View-Dependent Refinement of Progressive Meshes*. In Proceedings of SIGGRAPH 97, pp. 189–198.

[Hop98] Hoppe, H. *Efficient implementation of progressive meshes*. In Computers & Graphics 22, 1, pp. 27–36.

[Hop99] Hoppe, H. *Optimization of mesh locality for transparent vertex caching*. In Proceedings of SIGGRAPH 99 (Aug. 1999), Computer Graphics Proceedings, Annual Conference Series, pp. 269–276.

[Ise01] Isenburg M. *Triangle strip compression*. Computer Graphics Forum 20, 2 (2001), 91–101.

[Paj00] Pajarola R. and Rossignac J. *Compressed Progressive Meshes*. In 2000 IEEE Transactions on Visualization and Computer Graphics 6, 1 (Jan. - Mar. 2000), pp. 79–93.

[Pop97] Popovic J. and Hoppe H. *Progressive Simplicial Complexes*. In Proceedings of SIGGRAPH 97, pp. 217–224.

[Por03] Porcu M. and Scateni R. *An Iterative Stripification Algorithm Based on Dual Graph Operations*. In Proceedings of Eurographics 2003 (short presentations) (Sep. 2003) pp. 69–75.

[Ros99] Rossignac J. *Edgebreaker: Connectivity Compression for Triangle Meshes*. In 1999 IEEE Transactions on Visualization and Computer Graphics 5, 1 (Jan. - Mar. 1999), pp. 47–61.

[RsS99] Rossignac J. and Szymczak A. *Wrap&Zip decompression of the connectivity of triangle meshes compressed with Edgebreaker*. In Computational Geometry 14, 1-3 (1999), pp. 119–135.

[Spe97] Speckmann B. and Snoeyink J. *Easy triangle strips for TIN terrain models*. In Canadian Conference on Computational Geometry (1997), pp. 239–244.

[Ste01] Stewart A. J. *Tunneling for triangle strips in continuous level-of-detail meshes*. In Graphics Interface (June 2001), pp. 91–100.

[Tau98] Taubin G. and Rossignac J. *Geometric Compression Through Topological Surgery*. In 1998 ACM Transactions on Graphics 17, 2 (Apr. 1998), pp. 84–115.

[Tou98] Touma C. and Gotsman C. *Triangle Mesh Compression*. In Graphics Interface '98 (Jun. 98), pp. 26–34.

[Xia99] Xiang X., Held M., and Mitchell J. S. B. *Fast and effective stripification of polygonal surface models*. In 1999 ACM Symposium on Interactive 3D Graphics (Apr. 1999), pp. 71–78.

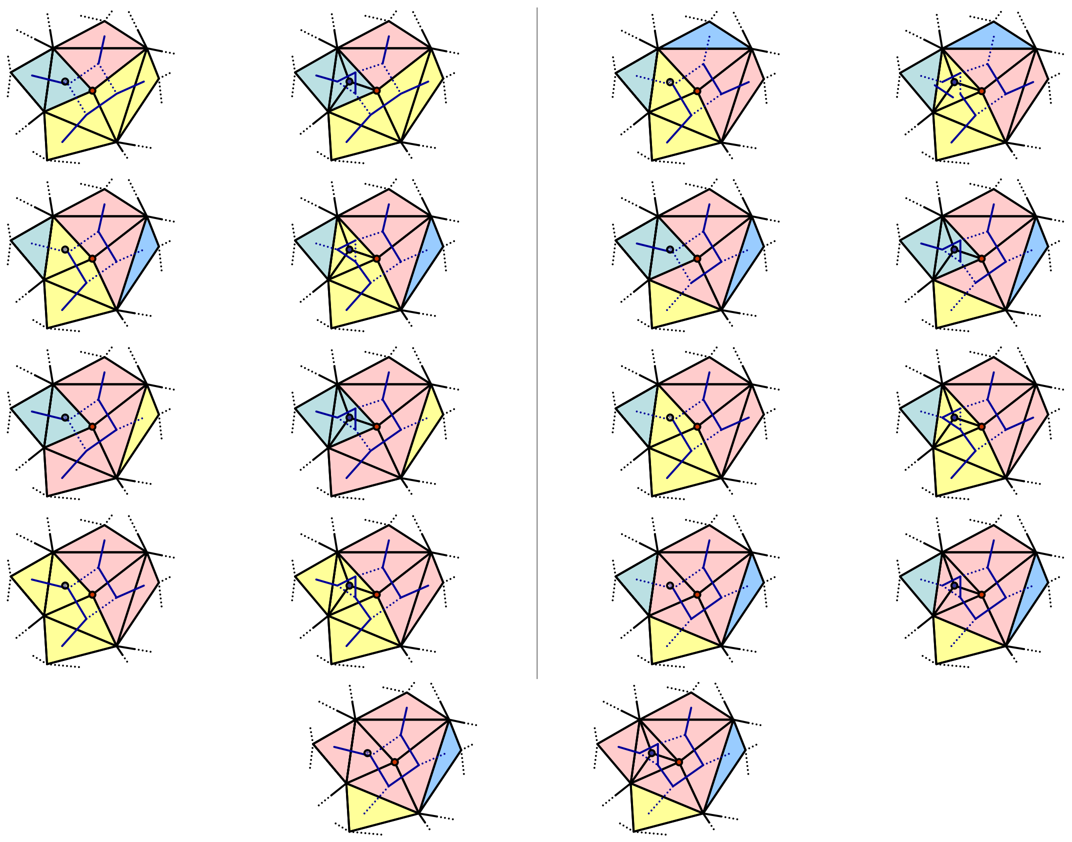


Figure 11: The graph rewriting rules to apply when inserting a new vertex with a VS operation. In each couple, on the left the configuration before the VS (the vertex to be split is marked in red), on the right the configuration after the VS (the new inserted vertex is marked in blue).

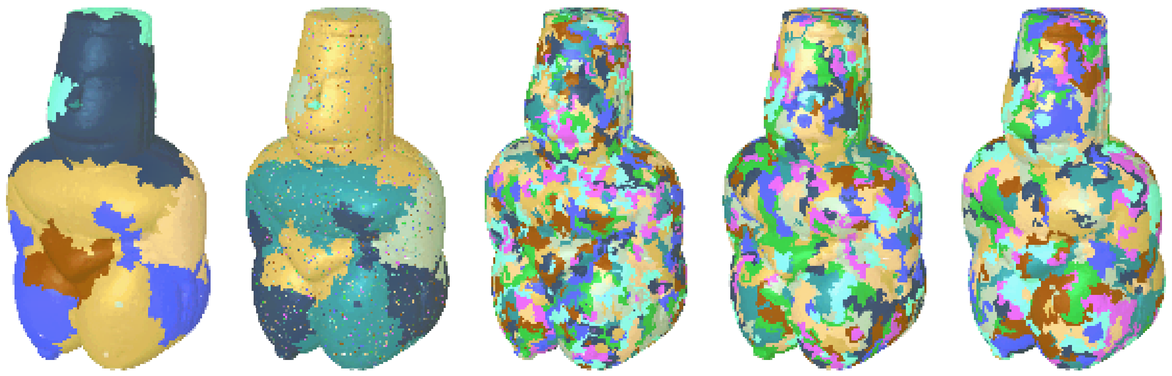


Figure 12: An example of LOD change on the "Dea madre" dataset. From left to right: the 9% LOD optimally stripified (51,511 tri's and 10 strips); the 13% LOD obtained from the 9% one only with local mesh restructuring operations (this and the subsequent are meshes of 74,381 tri's, 6,139 strips with 4,456 isolated tri's); the same mesh after a 6-tunnels search (1,676 strips with 489 isolated tri's); after a 10-tunnels search and graph recoloring (841 strips with 140 isolated tri's); after a 14-tunnels search and graph recoloring (489 strips with 46 isolated tri's).

Tree Growth Visualization

Lars Linsen[†*] Brian J. Karis[†] E. Gregory McPherson[‡] Bernd Hamann[†]

[†] Institute for Data Analysis and Visualization (IDAV)
Department of Computer Science
University of California, Davis

[*] Department of Mathematics and Computer Science
Ernst-Moritz-Arndt-Universität Greifswald, Germany

[‡] USDA Forest Service, Pacific Southwest Research Station
Department of Environmental Horticulture
University of California, Davis

ABSTRACT

In computer graphics, models describing the fractal branching structure of trees typically exploit the modularity of tree structures. The models are based on local production rules, which are applied iteratively and simultaneously to create a complex branching system. The objective is to generate three-dimensional scenes of often many realistic-looking and non-identical trees. Our goal, instead, is to visualize the growth of a prototypical tree of certain species. It is supposed to look realistic but, more importantly, has to conform with real, measured data. We construct a tree model being similar to existing ones and extend it by coupling the branching production rules with dynamic tree-growth rules. The latter are based on equations derived from measured street tree data for London Plane tree (Platanus acerifolia) such as tree height, diameter-at-breast-height, crown height, crown diameter, and leaf area. We map the global, measured parameters to the local parameters used in the tree model. The mapping couples knowledge from plant biology and arboriculture, as we deal with trees that are trained and manipulated to achieve desired forms and functions within highly urbanized environments.

Keywords
Tree Growth, Animation, L-systems, Scientific Visualization.

1 Introduction

Several methods exist in computer graphics to describe and model computer-generated trees. Their common goal is to generate, from scratch, photorealistic images of many trees of a selected species. The trees are designed to appear as natural as possible, one species at a time. Many trees of one species should vary in appearance, so that together they resemble a naturally grown forest stand.

Few approaches have tackled the animation of trees growing over time, as the growing process of a plant is complex and many biological phenomena need to be considered. To verify and validate tree models, authors typically refer to the human eye, which is easy to fool. The generated images are supposed to appear “natural”, i. e., as if they were exact copies of trees as they occur in natural settings. To our knowledge, none of these approaches verify their tree models by quantitatively comparing tree dimensions and other parameters with measured data from actual trees.

We generate tree growth animations from tree dimensions measured by scientists with the USDA Forest Service, Center for Urban Forest Research at the University of California, Davis. Their street tree growth study considered parameters such as tree height, crown height, crown diameter, diameter-at-breast-height, and leaf area. The study was conducted in Modesto, California and led to equations for predicting these parameters and their correlation. Tree growth equations were used with numerical models to estimate the annual benefits for pollutant uptake, energy savings, rainfall interception, carbon dioxide sequestration, and property value increase. We describe the study and measured parameters in Section 3.

When modeling a tree with a computer system, the structure of the tree is usually described procedu-

[*]linsen@uni-greifswald.de
[†]zeroprey@gmail.com, hamann@cs.ucdavis.edu
[‡]egmcpherson@ucdavis.edu

The Journal of WSCG, Vol. 13, ISSN 1213-6964
WSCG 2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency - Science Press

rally, where the model comprises information about the branching system such as branch length, branching angle and twist, or fractal dimension. Typically, the modularity of tree branching structures is exploited by defining local production rules, that are applied iteratively to create complex branching systems. The tree model parameters are local and the production rules are applied to each branch individually, while the measured parameters from the study are global, describing the overall shape and appearance of the tree. In Section 4, we describe the tree model we have chosen for our purposes; and in Section 5, we describe how the globally measured parameters are mapped to parameters locally controlling tree growth (using the tree model) and how the tree is grown in an animation.

2 Related Work

Computer graphics has been using formal descriptions for the modeling, simulation, and rendering of trees and plants for decades. The formal description is typically based on local production rules. Starting from the trunk of a tree, the production rules generate new branches and are applied iteratively to the individual parts of the tree until the desired branching structure is reached. This method of generating plants is based on the assumption of plant modularity, which leads to repeated patterns being observed throughout the plant structure. The production rules are typically "context-free" or "context-sensitive" in the context of formal languages.

The most common representative of such formal descriptions for tree modeling is the so-called *L-system* or *Lindenmayer-system*, named after the theoretical biologist Aristid Lindenmayer who introduced the concept in [Lin68]. Prusinkiewicz and Lindenmayer developed many algorithms to model different species with different characteristics, all based on L-systems [PL90]. One major development was the introduction of parametric L-systems, where the production rules depend on the values of some locally stored and updated parameters. Other authors picked up on their methods and developed them further. Examples can be found in [AK85, Blo85, LD99]. In [FH79], a tree model is discussed that maximizes total leaf area while varying the branching geometry. A survey of existing L-system approaches is given in [PHMH95].

In [PHHM97], more emphasis is given to how the L-system model applies to nature. Life, death and reproduction are discussed, as well as information flow in growing plants. Also, the influence of the environment on the growth of plants is considered.

For computer graphics applications such as computer-animated movies or video games, the main objective of modeling plants is to generate a scene being highly realistic. Stochastic tree models have been introduced to simulate variety within one species. The individual plants can be organized in a sophisticated way to create forests or fields [CSHD03] and even entire ecosystems [DCSD02].

The methods mentioned above are mainly targeted toward the generation of static tree models. To animate plant development, L-systems can be extended to *dL-systems* or *differential L-systems*, as introduced in [PHM93]. The production rules of dL-systems are parametric, where the values of the parameters are defined as the solution of differential equations. Recently, implicit surface representations for growing trees were used in [GMW04]. Inverse modeling techniques were used to define the tree structure and its development.

3 Tree Parameters

The study of street tree species underlying our method was conducted by the Center for Urban Forest Research. Tree size, management, and site conditions were measured for twelve common street tree species in the San Joaquin Valley city of Modesto, California. However, for the tree growth visualization described in this paper, we focus on one species, namely, the London Plane tree (Platanus x acerifolia). The 27 randomly sampled London Plane trees were planted from two to 89 years ago. The study is described in detail in [PMM01].

Data collected for each tree during June through September 1998 include species, age, address, diameter-at-breast-height, tree height, crown diameter in two directions (maximum and minimum axis), height to the base of crown, and leaf area. Observational data include a visual estimate of crown shape, pruning level, tree condition code, and planting location (front lawn, planting strip, or sidewalk cutout).

Condition code was calculated as per the Guide for Plant Appraisal (Council of Tree and Landscape Appraisers 1992). Pruning level estimation, distinguishing between no pruning, less than 10% of crown pruned, 10% to 39% pruned, and 40% or more pruned, was based on total percentage of crown removed due to crown raising, reduction, thinning, and heading during the last four-year pruning cycle. As trees matured, pruning included crown raising. Mature tree maintenance typically consisted of crown cleaning and thinning.

Two digital photos of each tree crown, taken at perpendicular angles (chosen to provide an unobstructed view of the crown) were used to estimate leaf area using an image processing method [PM98, PM03]. Ages of trees for which age data were missing or entered incorrectly in the database, were verified through searching handwritten planting records, interviewing residents and city arborists, or increment coring to count growth rings. Crown height was calculated by subtracting the bole height (distance to base of crown) from total tree height.

Typically, street tree databases include diameter-at-breast-height size classes but rarely any age information for each tree. Therefore, in this study only diameter-at-breast-height was regressed on age; all other variables were regressed on diameter-at-breast-height (DBH), enabling users to predict the other dimensions using measures of diameter-at-breast-height alone. Three curve-fitting models were tested to a small sample of healthy trees. A logarithmic regression model provided the best fit for predicting all parameters except leaf area, for which a non-linear exponential model was used. The resulting functions for DBH, height, crown diameter, crown height, and leaf area of the London Plane trees are shown in Figure 1.

In the following, we refer to these functions as $f_{DBH}(t)$, $f_H(t)$, $f_{CD}(t)$, $f_{CH}(t)$, and $f_{LA}(t)$, respectively, where parameter t is time. Visual observation of the data revealed increasing variability with age and size of the trees. Therefore, we assumed the error to be multiplicative as is indicated by the confidence intervals shown in the graphs. A complete description of the analysis and models, including the necessary standard error of estimates, response sample mean and correlation values needed for calculating confidence intervals are available on the Center for Urban Forest Research website[1].

4 Tree Model

The canonical parts of a branching structure are bifurcations and branches. In plant science, they are referred to as nodes and internodes, respectively. Due to the modularity of nodes and internodes, repeating patterns, and the fractal structure of trees, computer models typically use iteratively, simultaneously, and locally applied production rules to generate complex branching structures. Thus, the entire tree can be generated based on local operations and local parameters.

A branch or internode is defined by its length l, diameter d, start point $\mathbf{s}$, and direction $\mathbf{l}$, as shown in Figure 2(a). A bifurcation or node is defined by the angles ϕ_i between the axes of the parent branch and the child branches and by the ratios in length $\frac{l_i}{l_0}$ and diameter $\frac{d_i}{d_0}$ between the parent branch and the child branches, $i = 1,2$, as shown in Figure 2(b). When one of the child branches bifurcates again, it will, in general, not lie in the same plane but in a plane of different orientation. The change in orientation is defined by the divergence or twisting angle θ_i, $i = 1,2$. In addition to the nodes and internodes, there are leaves and flowers. No production rules are applied to leaves and flowers, but they can grow in size s. In our application, we only require leaves, but for other applications flowers can be integrated in the same way.

We use a parametric L-system to describe our tree model. The chosen parametric L-system can be defined as a context-free or context-sensitive grammar $G = (V,T,S,\Pi)$, where the set of variables V consists of branches $B(l,d,\mathbf{s},\mathbf{l})$ and the trunk $T(l,d,\mathbf{s},\mathbf{l})$, the set of terminals T consists of leaves $L(s)$, the start symbol

[1]`http://wcufre.ucdavis.edu`

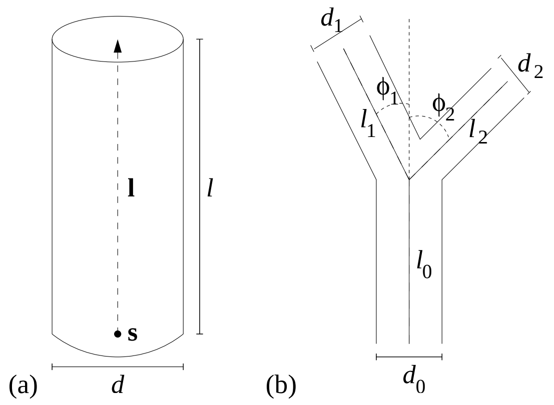


Figure 2: Branching structure consists of internodes (a) and nodes (b).

is the trunk $T(l,d,\mathbf{s},\mathbf{l})$, and a set of production rules, which are defined in the remainder of the paper. The trunk is, in principle, also a branch, but its parameter values cannot be derived from a parent branch, as there is none, which requires us to treat the trunk separately.

The branching structure is stored in a binary tree. In nature, there may occur, for example, ternary branching, but we are applying our methods to urban street trees that are frequently pruned. Since ternary branching is not beneficial for robust and balanced tree growth, such structures are regularly removed during pruning.

Each branch in the binary tree stores length l and diameter d. Start point $\mathbf{s}$ and direction $\mathbf{l}$ are not stored in a global coordinate system, but are computed in a local coordinate system with respect to the parent branch. Thus, each branch stores an orientation in form of a bifurcation angle ϕ and a divergence angle θ. To control growth of branches over time, we also store its time of creation t_0 and some growth factors, whose use is explained in the subsequent section. Growth can be limited by storing maximum length and diameter, which are, again, computed from the parameters of the parent branch.

5 Tree Growth

To grow a tree, we have to extend the static L-system tree model by introducing a continuous time dimension. Prusinkiewicz et al. [PHM93] enhanced parametric L-systems by solving differential equations to update local parameters. This so-called dL-system treats the solving of differential equations in the same way as the application of update rules. Thus, depending on the values of the considered parameters either production rules are applied or differential equations are solved for these parameters.

Since we are using measured values and since our goal is to visualize the measured data, there is no need to define plausible differential equations. Instead, we can directly incorporate the functions derived in Section 3

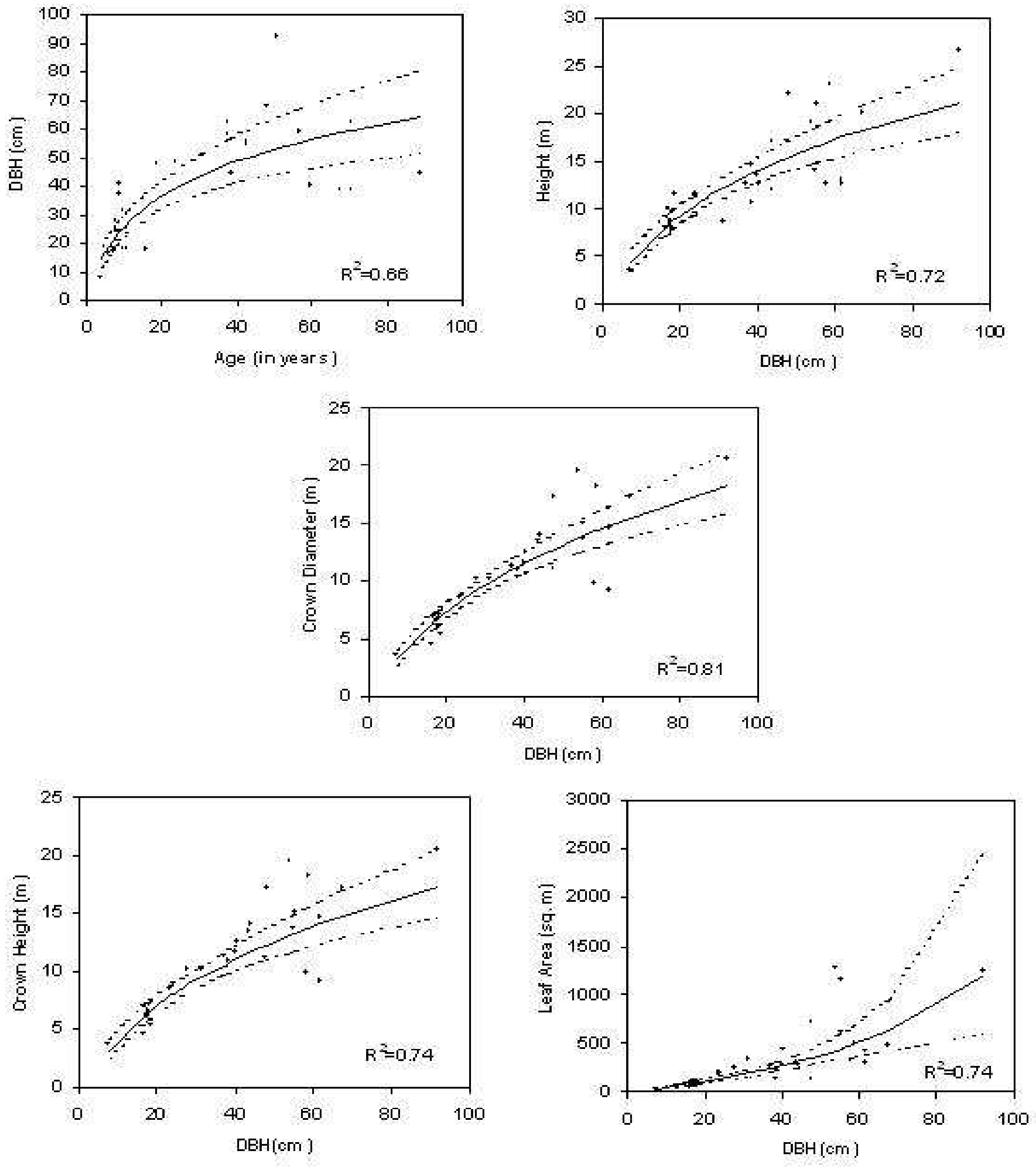


Figure 1: Experimental data for diameter-at-breast-height (DBH), height, crown diameter, crown height, and leaf area of London Plane trees.

and shown in Figure 1. The functions describe how the measured parameters are supposed to be updated over time. It remains to be explained how these measured global parameters are used to update and control the local parameters of the dynamic L-system. We make use of certain facts known from plant biology, see [HKVF88, KK79, Nik94]. We describe the relevant parameters for our model next.

Trunk length and diameter.
The length l and the diameter d of the trunk are directly controlled by the global functions. The length $l = l(t)$ is defined as the difference between the measured height of the tree and the measured height of the crown, i. e.,

$$l(t) = f_H(t) - f_{CH}(t)\ .$$

The diameter $d = d(t)$ is directly proportional to the measured DBH, i. e.,

$$d(t) = c_{DBH} \cdot f_{DBH}(t)\ ,$$

where $c_{DBH} \in [1, 1+\varepsilon)$ for a small $\varepsilon > 0$.

Branch length.
When a branch grows, it exhibits a similar growth rate as the trunk or the tree as a whole. Thus, the length of a branch follows the growth rates of the respective functions. To assure that our tree model has the actual, measured tree height, we use the function $f_H(t)$ to control the elongation of internodes.

Intuitively, primary branches (i. e., branches that emanate from the main branch/trunk) start growing before secondary branches (i. e., branches that emanate from primary branches) exist, and so on. Thus, primary and secondary branches do not grow at the same rate; while primary branches may already have reached a slow-growing phase, the secondary branches may still be in their initial fast-growing phase (Figure 1). This fact requires us to keep track of the time of creation t_0 of a branch and to compute the growth with respect to this point in time.

Moreover, a secondary branch does not reach the length of a primary branch, and a tertiary branch does not reach the length of a secondary branch, etc. Therefore, we multiply the growth function with a scaling coefficient c_l. The scaling coefficient c_l of a branch is obtained from the scaling coefficient of its parent branch multiplied by the scaling factor $s_l \in (0,1)$, where the trunk has a scaling coefficient c_l of value one. The scaling factor s_l depends on the species. For the London Plane tree, we use random values $s_l \in (0.6, 1)$. The randomness is required to make the tree appear less symmetric and thus more realistic.

In summary, the length $l = l(t)$ of a branch at time t is given by

$$l(t) = \frac{l_{max}}{H_{max}} \cdot c_l \cdot f_H(t - t_0) ,$$

where l_{max} and H_{max} are the maximum length of the branch and the maximum measured height of the tree, respectively. The maximum length l_{max} of a branch is determined by the maximum length of the parent branch multiplied by the scaling factor s_l. The growth of the branch terminates when the maximum length is reached.

Branch diameter.
When a branch bifurcates, the child branches have a smaller diameter than the parent branch. Leonardo da Vinci postulated that the square of the parent's diameter is the sum of the squares of the diameters of the children. In a dynamic setting, we use the measured function $f_{DBH}(t)$ multiplied by a scaling coefficient c_d to determine the growth of the diameter $d = d(t)$.

The scaling coefficient c_d is based on the scaling coefficient c'_d of the parent branch but also on the scaling coefficient c_l, which establishes a correlation between the scaling in length and diameter. The scaling coefficient is computed as $c_d = c_l \cdot c'_d \cdot (1 - 0.7 \cdot c'_d)$. The trunk has a scaling coefficient c_d of value one. Different diameters for different branches are induced by the randomness in the scaling coefficient c_l.

The growth in diameter is computed with respect to the time of creation t_0. The diameter $d = d(t)$ is defined by

$$d(t) = c_d \cdot f_{DBH}(t - t_0) .$$

Branch orientation.
The orientation of a branch is determined by the orientation of the parent branch, the bifurcation angle ϕ, and the divergence angle θ. The bifurcation angle ϕ is based on the ratio of crown diameter $f_{CD}(t)$ and crown height $f_{CH}(t)$, which defines the shape of the crown. London Plane trees are vertically ellipsoidal, which means that their crown height is greater than crown diameter. The ratio of crown diameter $f_{CD}(t)$ and crown height $f_{CH}(t)$ is approximately constant over time. We define the bifurcation angle by

$$\phi = \arctan\left(\frac{f_{CD}}{f_{CH}}\right) \pm \alpha ,$$

where α is a small random angle to make the tree less symmetric and thus more realistic.

When choosing divergence angles θ, we have to consider that we are visualizing urban street treesregularly pruned to obtain an "optimal" shape. A balanced tree, where primary branches called scaffolds are evenly spaced radially around the trunk, is considered optimal. Also, lower branches are removed to allow for clearance by trucks. Therefore, we choose

$$\theta = 130^\circ ,$$

which results in evenly spaced branches spiraling up the trunk.

It remains to discuss how to decide when to apply the update rules leading to tree growth and when to apply production rules leading to bifurcation. Our approach is to grow each branch using the update rules, until the branch has reached its maximum length, and to create a new branch using the production rules, once the branch has reached its maximum length.

Leaves are grown on all branches being smaller than a predefined threshold. The threshold is, again, dependent on the species. Leaves spiral around the branch at a set interval and have randomized orientation.

6 Results and Discussion

To visualize tree structure, we render each branch as a cylinder. The stored diameter d is always the diameter at the beginning of a branch. The diameter at the end of the branch is determined by the diameter of the adjacent branch. For an ending branch, which has no child branches, the cylinder degenerates to a cone.

We have taken digital photographs of both the bark and the leaves of a London Plane tree, which we use as textures for the branches and leaves in our renderings. We have modified the bark texture such that the texture can be wrapped around a branch without discontinuities in the transition area and such that multiple copies of the textures can be stitched together without discontinuities.

We animate tree growth by using the tree model of Section 4 and the local growth parameters discussed in Section 5, which are used to visualize the global parameters from Section 3. Snapshots of the animation, taken at ages $t = 10$, 20, 30, 40, and 50 years, are shown in Figure 3.

To obtain a better feeling for the dimensions of the tree, we add context in form of a human standing next to the tree. For reference, we also display the age of the tree during animation.

In addition to the quantitative, measured parameters that directly influence the visual appearance of the tree, we are also interested in visualizing quantitative benefit-cost parameters. The dollar (US) value of annual benefits for the London Plane tree in Modesto were numerically modeled for energy savings, air pollutant uptake, CO_2 sequestration, stormwater runoff reduction, and aesthetics [PM03]. Average annual costs for the same species were based on an analysis of tree work records for plant/water, prune, remove, infrastructure repair, and storm clean-up. Their values are displayed by benefit-cost bars animated to reflect the typical stream of benefits and costs over time for this species in Modesto.

The results (Figure 3) are quite satisfactory, as we successfully animate growth of a realistic-looking London Plane tree over 50 years, while conforming to measured tree dimensions. The emphasis of our work was not to make the tree look as realistic as possible but to display its growth in terms of trunk height and width, crown height and width, and leaf area. Growth of these parameters is represented in a visually appealing and intuitive way. For example, one can observe how the diameter of the trunk increases steadily but with a decreasing rate due to the fact that the trunk grows a new ring every year, but annual ring width decreases over time.

Although we do not have a video recording available of a real tree growing over time, we can, at least, compare visually the results in Figure 3 with the digital photograph shown in Figure 4. Our goal was not to replicate this particular tree in Figure 4, but to grow a prototypical London Plane tree.

We use knowledge from plant biology where possible, e. g., to estimate certain coefficients needed to map measured parameters to our tree model parameters. On the other hand, we have developed methods for urban street trees, where pruning practices modify tree architecture. Thus, certain concepts, such as natural death of certain branches or leaves or information flow in growing plants as described in [PHHM97], are not relevant. Instead, we complement biological knowledge with arboricultural knowledge, for instance, to estimate the orientation and spacing of scaffold branches.

The appearance of our tree could still be improved. The growth direction of the branches would benefit from more equal spacing [FH79]. The concept of having branches grow toward the sky and toward least-crowded areas could be introduced. This concept also includes the thinning of branches in the tree's interior caused by lack of light. The implementation of this concept would require us to change the tree model, as it requires global information; our tree model only stores local information. For example, when growing one branch, we can only retrieve information about the branch itself and its parent and child branches. We cannot retrieve information about spatially close branches, which, if known, would allow us to bend the current branch to achieve an equal distribution.

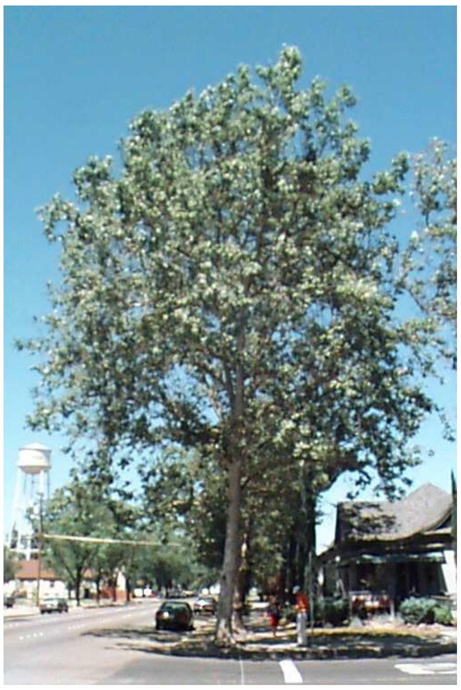


Figure 4: Digital photograph of a London Plane tree in the San Joaquin Valley city of Modesto, California.

7 Conclusions and Future Work

We have introduced an approach to model and visualize the growth of urban street trees. Growth is controlled by measured, global parameters such as tree height and width, crown height and width, and leaf area. We map these measured parameters to the local parameters of a computer-generated tree model. The tree model is based on a formal description using locally, iteratively, and simultaneously applied production rules, which exploit the modular structure of trees and allow for easy modeling of fractal branching. The production rules are coupled with local update rules that describe the dynamic growth of individual branches. The update rules are based on functions derived from measured data. Hence, we can animate and visualize the growth of a realistic-looking tree based on real data. The animation also includes additional benefit-cost parameters.

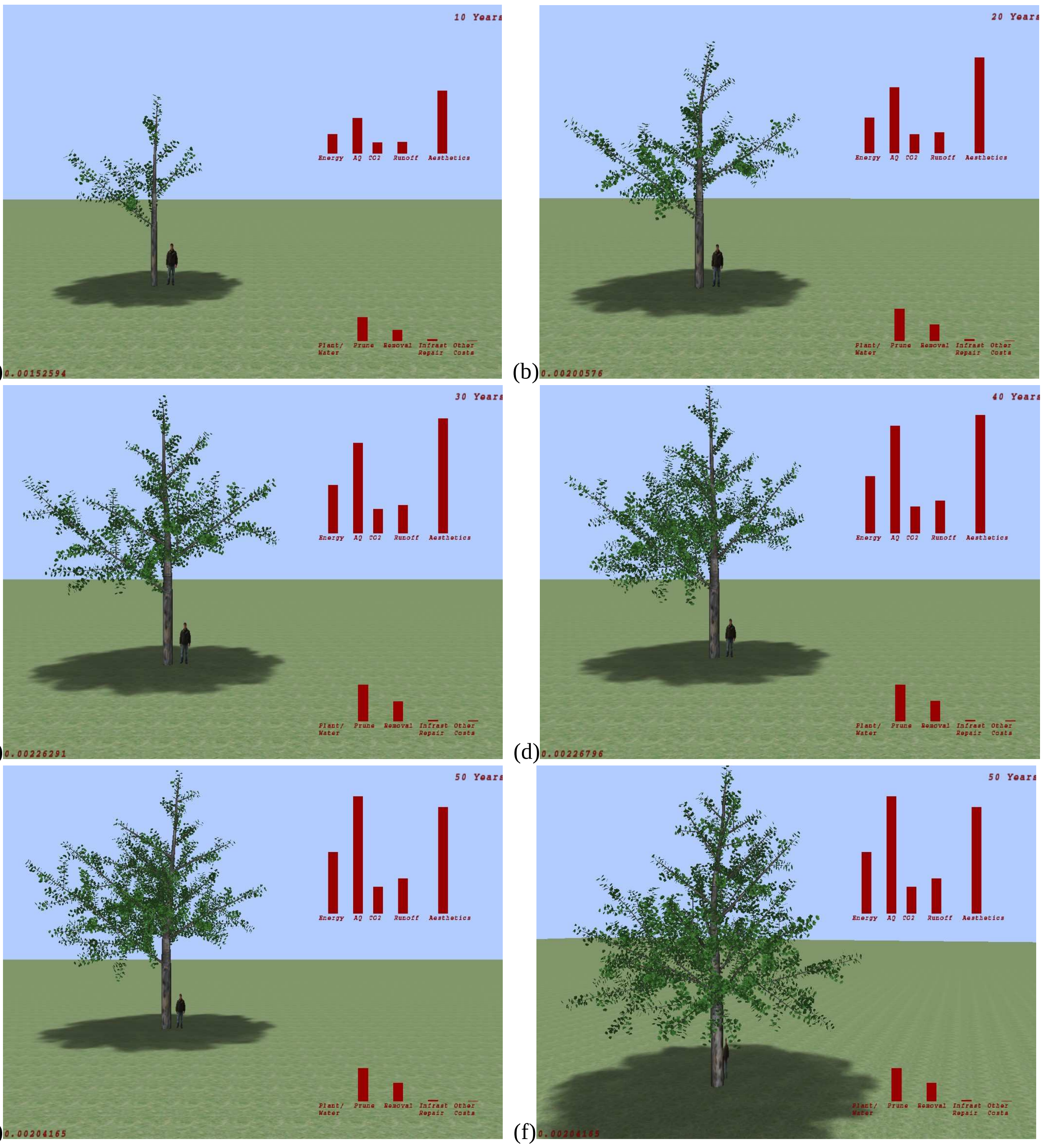


Figure 3: Tree growth visualization of a London Plane tree; ages shown: (a) 10 years, (b) 20 years, (c) 30 years, (d) 40 years, and (e),(f) 50 years.

We have used our method for modeling the London Plane tree, whose parameters were measured in the San Joaquin Valley city of Modesto, California. Up to now, we have used only this species, but we plan to use our methods for all twelve street tree species of the San Joaquin Valley study. This information will help gardeners, designers, planners, and tree managers to decide which species are most appropriate to grow in terms of size, form, benefits, and costs. Because trees are long-term investments, selecting the right species is critical to achieving maximum net benefit. The application of our methods to other species is straight forward, as it only requires us to exchange the growth functions derived from the measured values, and to use the appropriate textures. In addition, the species-dependent coefficients c_{DBH}, s_l (controlling c_l and c_d), and l_{max} and the threshold for growing leaves must be adjusted.

We plan to enhance our tree model by adding capability to store and retrieve global shape and structure information of the tree. The local structure, which is based on production rules, makes it easy to model the fractal branching structure of a tree, but limits the control of global shape.

By coupling the L-system-based model with a data structure capable of retrieving global shape information, we hope to achieve a more equal branch distribution, where branches grow in preferred directions. Global shape control will make it easier to match crown shape more precisely.

Retrieving global information from the tree model also will facilitate fast computation of leaf area at any time during animation. Thus, leaf area could be compared to measured data and the derived leaf-area growth function $f_{LA}(t)$. We plan to use leaf area to control the branching time of individual branches, the number and distribution of leaves, and the fractal dimension of the branching structure.

Acknowledgments

This work was supported by the Elvenia J. Slosson Fund for Ornamental Horticulture, University of California, Division of Agriculture and Natural Resources; the National Science Foundation under contract ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, through the National Partnership for Advanced Computational Infrastructure (NPACI) and a large Information Technology Research (ITR) grant; the National Institutes of Health under contract P20 MH60975-06A2, funded by the National Institute of Mental Health and the National Science Foundation; and the U.S. Bureau of Reclamation. We thank the members of the Visualization and Graphics Research Group at the Institute for Data Analysis and Visualization (IDAV) at the University of California, Davis, and the members of the Center for Urban Forest Research at the University of California, Davis.

References

[AK85] M. Aono and T.L. Kunii. Botanical tree image generation. *IEEE Computer Graphics & Applications*, 4(5):10–34, 1985.

[Blo85] J. Bloomenthal. Modeling the mighty maple. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 305–311. ACM SIGGRAPH, 1985.

[CSHD03] M. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for texture and image generation. In A.P. Rockwood, editor, *SIGGRAPH '03 Proceedings*. ACM SIGGRAPH, 2003.

[DCSD02] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. In M. Gross, K.I. Joy, and R.J. Moorhead, editors, *Proceedings of IEEE Visualization '02 conference*. IEEE Computer Society Press, 2002.

[FH79] J. Fisher and H. Honda. Branch geometry and effective leaf area: a study of terminalia-branching pattern; 1: Theoretical trees, and pattern; 2: Survey of real trees. *American Journal of Botany*, 66:633–655, 1979.

[GMW04] Callum Galbraith, Lars Muendermann, and Brian Wyvill. Implicit visualization and inverse modeling of growing trees. *Computer Graphics Forum (Proceedings of Eurographics 2004)*, 23(3), 2004.

[HKVF88] H.T. Hartmann, A.M. Kofranek, V.E.Rubatzky, and W.J. Flocker. *Plant Science: Growth, Development and Utilization of Cultivated Plants*. Englewood Cliffs, NJ, Prentice-Hall, 1988.

[KK79] P.J. Kramer and T.T. Kozlowski. *Physiology of Woody Plants*. Academic Press, New York, 1979.

[LD99] B. Lintermann and O. Deussen. Interactive modeling of plants. *IEEE Computer Graphics & Applications*, 19(1), 1999.

[Lin68] A. Lindenmayer. Mathematical models for cellular interaction in development. *Journal of Theoretical Biology*, 18:280–315, 1968.

[Nik94] K.J. Niklas. *Plant Allometry: The Scaling of Form and Process*. University of Chicago Press, Chicago, 1994.

[PHHM97] P. Prusinkiewicz, M. Hammel, J. Hanan, and R. Mech. Visual models of plant development. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. III: Beyond Words*, pages 535–597. Springer-Verlag, Berlin, 1997.

[PHM93] P. Prusinkiewicz, M. Hammel, and E. Mjolsness. Animation of plant development. In J.T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 351–360. ACM SIGGRAPH, 1993.

[PHMH95] P. Prusinkiewicz, M. Hammel, R. Mech, and J. Hanan. The artificial life of plants: Artificial life for graphics, animation, and virtual reality. In *SIGGRAPH '95 Course Notes*, pages 1–38. ACM SIGGRAPH, 1995.

[PL90] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990.

[PM98] P.J. Peper and E.G. McPherson. Comparison of five methods for estimating leaf area index of open-grown deciduous trees. *Journal of Arboriculture*, 24(2):98–111, 1998.

[PM03] P.J. Peper and E.G. McPherson. Evaluation of four methods for estimating leaf area of isolated trees. *Urban Forestry & Urban Greening*, 2:19–29, 2003.

[PMM01] P.J. Peper, E.G. McPherson, and S.M. Mori. Equations for predicting diameter, height, crown width, and leaf area of San Joaquin Valley street trees. *Journal of Arboriculture*, 27(6), 2001.

VSA-based Fractal Image Compression

Huaqing Wang[1], Meiqing Wang[2], Tom Hintz[1], Qiang Wu[1], Xiangjian He[1]
[1] Faculty of Information Technology, University of Technology, Sydney
PO Box 123, Broadway 2007, Sydney, NSW, Australia
{huwang,hintz, wuq, sean}@it.uts.edu.au
[2]College of Mathematics and Computer Sciences, Fuzhou University
No.502 Gong Ye Road, Fuzhou, Fujian, China, 350002
wangmeiqing@hotmail.com

ABSTRACT

Spiral Architecture (SA) is a novel image structure which has hexagons but not squares as the basic elements. Apart from many other advantages in image processing, SA has shown two unbeatable characters that have potential to improve image compression performance, namely, *Locality of Pixel Density* and *Uniform Image Partitioning*. Fractal image compression is a relatively recent image compression method which exploits similarities in different parts of the image. The basic idea is to represent an image as fixed points of *Iterated Function Systems* (IFS). Therefore, an input image can be represented by a series of IFS codes rather than pixels. In this way, an amazing compression ratio 10000:1 can be achieved. The application of fractal image compression presented in this paper is based on Spiral Architecture. Since there is no mature capture and display device for hexagon-based images, the experiments are implemented on a newly proposed mimic scheme, called *Virtual Spiral Architecture* (VSA). The experimental results in the paper have shown that introducing Spiral Architecture into fractal image compression will improve the compression performance in image quality with little trade-off in compression ratio. A lot of research work exists in this area to further improve the results.

Keywords
Fractals, image compression, image encoding, Virtual Spiral Architecture, hexagonal structure.

1. INTRODUCTION

Needless to say, visual information is of vital importance if human beings are to perceive, recognize and understand the surrounding world. With the tremendous progress that has been made in computer power, the corresponding growth in the multimedia market and the advent of the World Wide Web, it is becoming more than ever possible for images to be widely utilized in our daily life. In general, an image file contains much more data than a text file. An image with a large amount of data requires much memory to store, takes longer to transfer, and is intricate to process. For example, a grey scale image with 256 × 256 pixels requires about 64 KB of memory space and more than 18 seconds to download using a 28.8K Dialup Modem. As a consequence, image compression becomes necessary due to the limited communication bandwidth, CPU speed and storage size. Image compression has been one of the most challenging fields in the image processing research.

Fractal image compression is a relatively recent image compression method which exploits similarities in different parts of the image. For example, with a picture of a fern (Figure 1) one can see easily where these similarities lie: each fern leaf resembles a smaller fern. This is known as the famous Barnsley fern [Barnsley1985]. During more than two decades of development, the *Iterated Function System* (IFS) based compression algorithm stands out as the most promising direction for further research and improvement [Barnsley1993]. The basic idea is to represent an image as the fixed points of IFSs. An appropriately chosen IFS consists of a group of affine transformations [Fisher1995]. Therefore, an input image can virtually be represented by a series of IFS codes. In this way, a compression ratio 10000:1 can be achieved

The Journal of WSCG, Vol.13, ISSN 1213-6964
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency – Science Press

[Barnsley1988]. In short, for fractal image compression an image is represented by fractals rather than pixels. Each fractal is defined by a unique IFS consists of a group of affine transformations. Therefore the key point for this algorithm is to find fractals which can best approximate the original image and then to represent them as a set of affine transformations.

Figure 1. A fern leaf

The application of fractal image compression presented in this paper is based on a novel image structure, Spiral Architecture [Sheridan1991], which is inspired from anatomical considerations of the primate's vision [Schwartz1980]. On The Spiral Architecture, an image is a collection of hexagonal elements [Sheridan2000]. In the case of human eye, these elements (hexagons) would represent the relative position of the rods and cones on the retina. Each pixel on The Spiral Architecture is identified by a designated positive number, called Spiral Address as shown in Figure 2. The numbered hexagons form the cluster of size 7^n. The hexagons tile the plane in a recursive modular manner along the spiral direction [He1999]. Any hexagonal pixel has only six neighboring pixels which have the same distance to the centre hexagon of the seven-hexagon unit of vision.

This paper is organized as follows. Beginning with a review of fractal image compression in Section 2, an introduction of the Spiral Architecture is presented in Section 3. In Section 4, we describe the procedure of adopting the fractal image compression algorithm on The Spiral Architecture and the experimental results are supplied in Section 5 with some quantified analysis. We conclude in Section 6 by summarizing the opportunity of better performance for fractal image compression on the Spiral Architecture and by mentioning areas for future research.

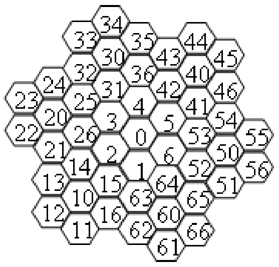


Figure 2. A collection of 7^2 = 49 Hexagons with labelled addresses

2. CONCEPTS OF FRACTAL IMAGE COMPRESSION

In the following section, the basic concepts of fractal image compression on the traditional square structure would be introduced. Before delving into details, there are some highlights of fractal image compression.

- It is a promising technology, though still relatively immature.
- The fractals are represented by Iterated Function Systems (IFSs).
- It is a block-based lossy compression method.
- Compression has traditionally been slow but decompression is fast.

Theory and Math Background

The fundamental principle of fractal image compression consists of the representation of an image by an iterated function system (IFS) of which the fixed point is close to that image. This fixed point is named as '*fractal*' [Fisher1995]. Each IFS is then coded as a contractive transformation with coefficients. Banach's fixed point theorem guarantees that, within a complete metric space, the fixed point of such a transformation may be recovered by iterated implementation thereof to an arbitrary initial element of that space [Kreyszlg1978]. Therefore, the encoding process is to find an IFS whose fixed point is close to the given image. The usual approach is based on the *collage theorem*, which provides a bound on the distance between the image to be encoded and the fixed point of an IFS (more details please refer to [Fisher1995] chapter 2). A suitable transformation may therefore be constructed as a 'collage' from the image to itself with a sufficiently small 'collage error' (the distance between the collage and the image) guaranteeing that the fixed point of that transformation is close to the original image [Wohlberg1999].

In the original approach, devised by Barnsley, this transformation was composed of the union of a number of affine mappings on the entire image [Barnsley1993]. While a few impressive examples of image modelling were generated by this method (Barnsley's fern, for example [Barnsley1988]), no automated encoding algorithm was found. Fractal image compression became a practical reality with the introduction by Jacquin of the partitioned IFS (PIFS) [Jacquin1993], which differs from an IFS in that each of the individual transformation operates on a subset of the image, rather than the entire image. Since the image support is tiled by 'range blocks', each of which is mapped from one of the 'domain blocks' as depicted in Figure 3, the combined mappings constitute a transformation on the image as a whole. The transformation minimizing the collage error within this framework is constructed by individually minimizing the collage error for each range block, which requires locating the domain block which may be made closest to it under an admissible block mapping. This transformation is then represented by specifying, for each range block, the identity of the matching domain block together with the block mapping parameters minimizing the collage error for that range block.

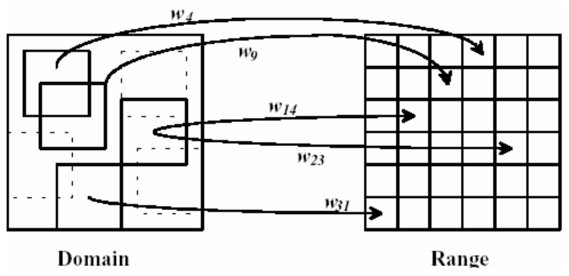


Figure 3. Each range block is constructed by a transformed domain block

Basic Fractal Image Encoder

The encoder has to solve the following problem: for each range block R the best approximation

$$R \approx sD + oI \qquad (2.1)$$

needs to be found, where D is a codebook block transformed from a domain block to the same size as R. The coefficients s and o are called *scaling* and *offset*. We work out this problem with the Euclidean norm. That is, to minimize

$$E(D,R) = \min_{s,o} \|R - (sD + o\mathbf{1})\| \qquad (2.2)$$

we can use the well known method of least squares to find the optimal coefficients directly as follows.

Given a pair of blocks R and D of n pixels with intensities $r_1,\ldots, r_n$ and $d_1,\ldots, d_n$ we have to minimized the quantity

$$\sum_{i=1}^{n}(s \cdot d_i + o - r_i)^2 \qquad (2.3)$$

The best coefficients s and o are

$$s = \frac{n\left(\sum_{i=1}^{n} d_i r_i\right) - \left(\sum_{i=1}^{n} d_i\right)\left(\sum_{i=1}^{n} r_i\right)}{n\sum_{i=1}^{n} d_i^2 - \left(\sum_{i=1}^{n} d_i\right)^2} \qquad (2.4)$$

and

$$o = \frac{1}{n}\left(\sum_{i=1}^{n} r_i - s\sum_{i=1}^{n} d_i\right) \qquad (2.5)$$

With s and o given the square error is

$$E(D,R)^2 = \frac{1}{n}\left[\sum_{i=1}^{n} r_i^2 + s\left(s\sum_{i=1}^{n} d_i^2 - 2\sum_{i=1}^{n} d_i r_i + 2o\sum_{i=1}^{n} d_i\right) + o\left(on - 2\sum_{i=1}^{n} r_i\right)\right] \qquad (2.6)$$

If the denominator in equation 2.4 is zero then $s = 0$ and $o = \sum_{i=1}^{n} r_i/n$

In summary the baseline fractal encoder with fixed block size operates in the following steps.

1. **Image segmentation.** Segment the given image using a fixed block size, for instance, 4×4. The resulting blocks are called ranges R_i.
2. **Domain pool and codebook blocks definition.** By stepping through the image with a step size of l pixel(s) horizontally and vertically create a set of domain blocks, which are four times as the size of range blocks. By averaging the intensities of four neighboring pixels each domain blocks shrinks to match the size of the ranges. This produces the codebook blocks D_i.
3. **The search of best *s* and *o*.** For each range block R_i an optimal approximation $R_i \approx sD_i + oI$ in the following steps:

   a) For each codebook block D_i compute an optimal approximation $R_i \approx sD_i + oI$ in three steps:

   i. Perform the least squares optimization using formulas 2.4 and 2.5, yielding a real coefficient scalar s and an offset o.

   ii. Quantize the coefficients using a uniform quantizer.

   iii. Using the quantized coefficients s and o compute the error $E(R_i, D_i)$.

   b) Among all codebook blocks D_i find the block D_k with minimal error

   $E(R_i, D_k) = \min_i E(R_i, D_i)$.

   c) Output the code for the current range block consisting of indices for the quantized coefficient s and o and the index k identifying the optimal codebook block D_k.

3. SPIRAL ARCHITECTURE AND IMAGE REPRESENTATION

A digital image contains thousands of pixels to represent the real world and when we touch the term ‘pixel’ so far, that means a rectangular box in an image. Almost all the previous image processing and image analysis research is based on this traditional image structure. However, we do have a relatively new image structure called *Spiral Architecture* (SA) [Sheridan1996]. Spiral Architecture is inspired from anatomical considerations of the primate’s vision [Schwartz1980]. From the research about the geometry of the cones on the primate’s retina (See Figure 4) we can conclude that the cones’ distribution has inherent organization and is featured by its potential powerful computation abilities. The cones with the shape of hexagons are arranged in a Spiral clusters. This cluster consists of the organizational units of vision. Each unit is a set of seven hexagons compared with the traditional rectangular image architecture using a set of 3×3 vision unit as shown in Figure 5.

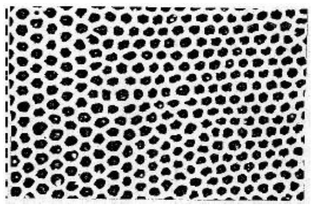


Figure 4. Distribution of Cones on the Retina

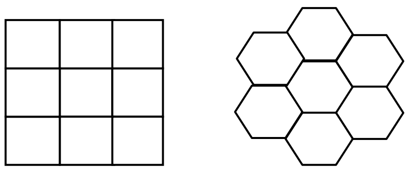

(a) Rectangular Architecture (b) Spiral Architecture

Figure 5. Unit of vision in the two image architectures

Spiral Addressing

The first step in SA formulation is initially labeling each of the individual hexagons with a unique *address*. The addresses of these hexagons will then be simply referred to the hexagons. This is achieved by a process that is initially applied to a collection of seven hexagons. Each of these seven hexagons is labeled consecutively with addresses 0, 1, 2, 3, 4, 5 and 6 as displayed in Figure 6.

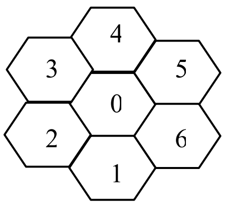


Figure 6. A collection of seven hexagons with unique addresses

Dilate the structure so that six additional collections of seven hexagons can be placed about the addressed hexagons, and multiply each address by 10. For each new collection of seven hexagons, label each of the hexagons consecutively from the centre address as we did for the first seven hexagons (see Figure 7).

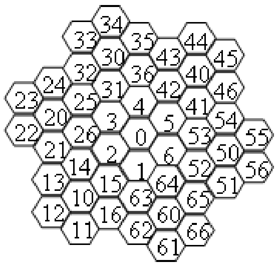


Figure 7. A collection of $7^2 = 49$ hexagons with labelled addresses

The repetition of the above steps permits the collection of hexagons to grow in powers of seven with uniquely assigned addresses. It is this pattern of growth of addresses that generates the *Spiral*. Furthermore, the addresses are consecutive in base seven.

The important aspect of each hexagon is that it has six neighboring hexagons. This establishes the property that for all hexagons, the centre of each hexagon has a constant distance from every one of its six neighbors. According to Umbaugh [Umbaugh1996], the difference of light intensities between pixels is highly related to the distance between them: the closer they are, the less difference observed. Hence, the light intensity of a hexagonal pixel can be considered being equally affected by the light intensities of its six neighboring pixels [He1999]. Moreover, each set of seven hexagons may enjoy very similar light intensities and the difference between the centre and others would be quite small. This idea is the foundation stone when considering image compression on SA.

Spiral Counting

Spiral Counting [Sheridan1996] is an algorithm that designates a sequence of hexagons in SA. It can be considered as a Spiral movement that given a commencing hexagon, counts for a pre-determined number and terminates at another certain hexagon. Any hexagon in an image can be reached by Spiral counting from any other given hexagon in the same image. When applying Spiral counting, it is strictly dependent on a pre-determined key define by Sheridan in [Sheridan1991]. A key is the first hexagon to be reached in an instance of a Spiral counting, which determines two important parameters: the distance and the orientation. For instance, given a Spiral address 15, the key of 15 can determine two values. One is the distance between the given hexagon 15 to the hexagon 0; the other is the orientation of hexagon 15 from hexagon 0. We could use the angle ω to represent the orientation (see Figure 8).

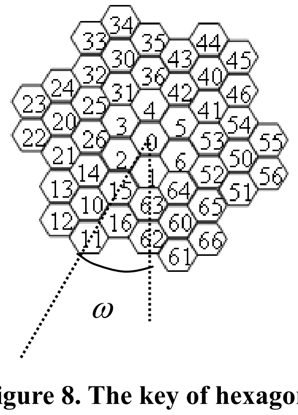


Figure 8. The key of hexagon 15

Spiral counting is used to define two operations in the SA, which are *Spiral Addition* and *Spiral Multiplication* [Sheridan1991]. Let a and b be Spiral addresses of two arbitrarily chosen hexagons in SA. Then,

- Spiral addition of a and b, denoted by a + b, is the Spiral address of the hexagon found by Spiral counting b hexagons in the key of Spiral address 1 from the hexagon with Spiral address a;
- Spiral multiplication of a and b, denoted by a x b, is the Spiral address of the hexagon found by Spiral counting b hexagons in the key of Spiral address a from the hexagon with Spiral address 0.

Spiral Architecture together with the operations of Spiral Addition and Spiral Multiplication is a Euclidean Ring [Sheridan1991]. This property is necessary to further implement SA for image compression.

Virtual Spiral Architecture

SA has two unbeatable characters that are expected to improve image compression performance: *Locality of Pixel Intensity* and *Uniform Image Partitioning* [Hintz2003]. However due to the lack of capture and display devices, SA has not yet been widely used in image processing. In order to make SA applicable on the current available devices, Wu constructed a mimic scheme called *Virtual Spiral Architecture* (VSA) [Wu2004], with which images on rectangular structure can be smoothly converted to SA.

VSA mimicking scheme is so called 'virtual' because it only exists on computer memory during the procedures of image processing. The processing result will still be displayed on the traditional rectangular structure (see Figure 9).

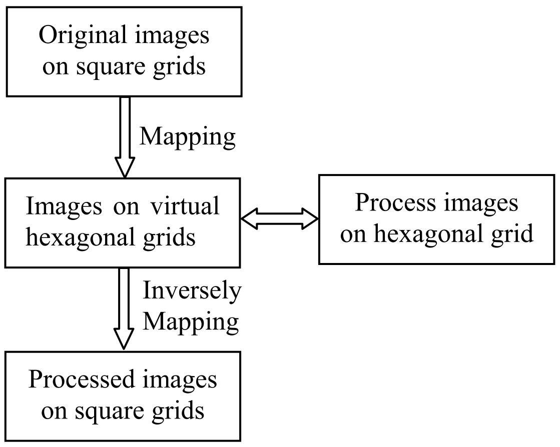

Original images on square grids
Mapping
Images on virtual hexagonal grids
Process images on hexagonal grid
Inversely Mapping
Processed images on square grids

Figure 9. Flowchart of image processing on virtual Spiral Architecture

In order to keep the resolution, hexagonal and square pixels are defined as the same size, i.e. 1 unit area. Then if we map the SA on a traditional image and then let *N* denote the number of square pixels covered by a hexagonal pixel and let s_i represent the size of overlapped area in a certain square pixel *i* (See Figure 10), so the contribution of gray level given by this square pixel to the hexagonal pixel is measured by the percentage of the overlapped area, i.e. p_i.

$$p_i = s_i / 1 \times 100\% \qquad (3.1)$$

Therefore the grey value of this hexagonal pixel is

$$g_{Hex} = \sum_{i=1}^{N} (g_{Squ_i} \cdot p_i), \qquad (3.2)$$

where g_{Squ_i} is the grey level of the i square pixel.

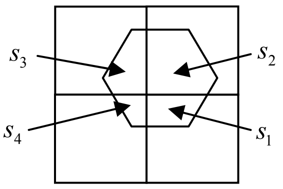


Figure 10. The relationship between a virtual hexagonal pixel and overlapped square pixels.

As a result, the grey level information for SA is now available during the procedure of image processing and the experiment result can be displayed back on a traditional square-structure-based device following the similar mapping method (see Figure 11)

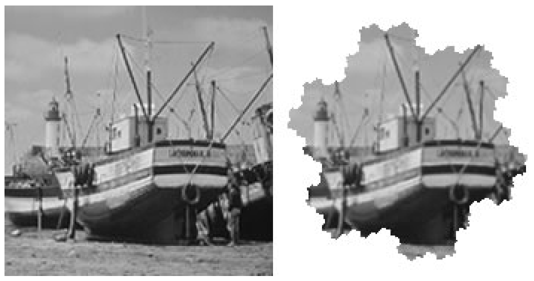


Figure 11. Boat in Square Structure and Virtual Spiral Architecture Displayed on Normal Device

4. FRACTAL IMAGE COMPRESSION ON SPIRAL ARCHITECTURE

In this preliminary research on adopting fractal image compression into Spiral Architecture, we follow the same idea applied on square structure, i.e. PIFS as described earlier. Firstly we separate the image into range blocks of seven hexagonal pixels and define the domain blocks of seven times more, i.e. 49 pixels (see Figure 12). Each pixel in the image can be the centre of a domain block. Then we include the neighboring 48 pixels around it based on Spiral counting to form a domain block unless any pixel of this domain block is out of the given image.

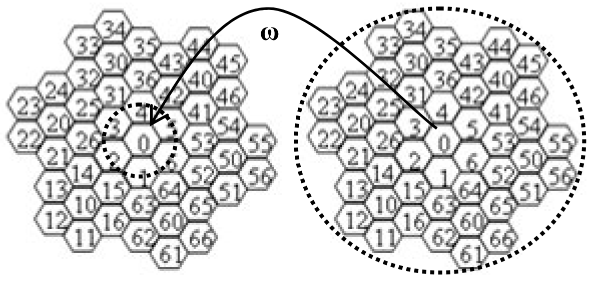


Figure 12. Range and domain blocks in Spiral Architecture

A number of researchers have noticed a tendency for a range block to be spatially close to the matching domain block, [Beaumont1990; Barthel1994], based on the observed tendency for distributions of spatial distances between range and matching domain blocks to be highly peaked at zero [Jacquin1993; Woolley1995]. Motivated by this observation, the domain pool for each range block may be restricted to a region about the range block [Jacquin1990], or a spiral search path may be followed outwards from the range block position [Beaumont1990; Barthel1994]. Therefore, in order to reduce the computational complexity, for each range block we only search for up to 343 domain blocks, which are around this range block. Each of those range blocks has at most 343 domain blocks in the domain pool and the centers of domain blocks in the pool are the first 343 pixels counting from the centre of range block through spiral direction.

5. EXPERIMENTAL RESULTS

We use the same algorithms mentioned before on square and Spiral Architecture for four popular images: a building, a boat and a house. Figures 13 through 18 show the experimental results and we summarize them in two tables.

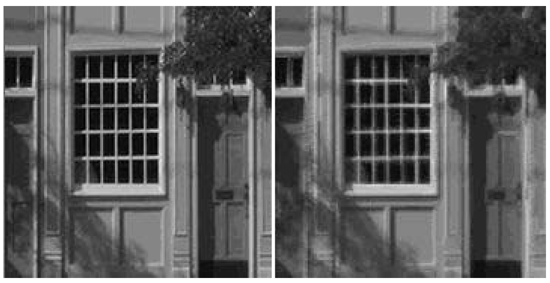


Figure 13. Original and compressed 'building' in square structure

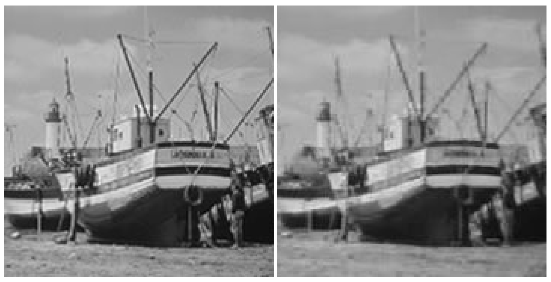


Figure 14. Original and compressed 'boat' in square structure

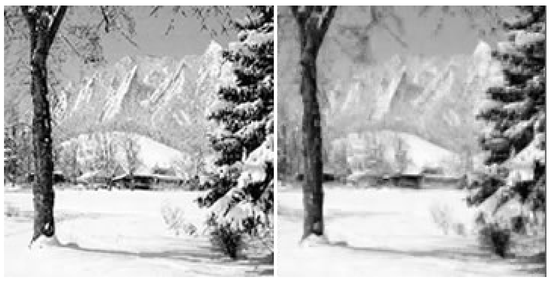


Figure 15. Original and compressed 'house' in square structure

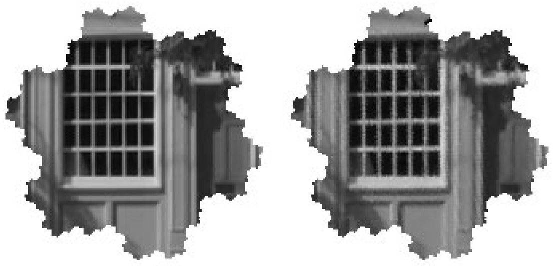


Figure 16. Original and compressed 'building' in Spiral Architecture

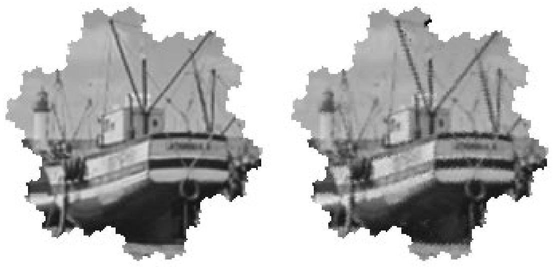


Figure 17. Original and compressed 'boat' in Spiral Architecture

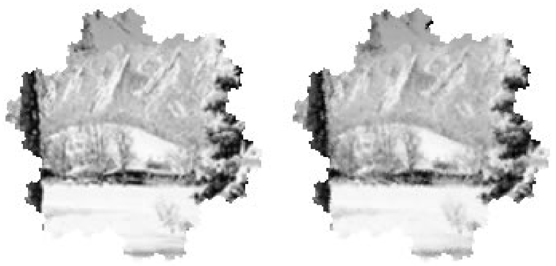


Figure18. Original and compressed 'house' in Spiral Architecture

Image	Compression ratio	PSNR
Building	3.37	23.40
Boat	3.37	26.56
House	3.37	22.41

Table 1. Summary for images on square structure

Image	Compression ratio	PSNR
Building	2	25.43
Boat	2	29.73
House	2	26.20

Table 2. Summary for images on Spiral Architecture

As the range block on SA is of 7 pixels (compare with 16 pixels in square structure), the compression ratio is slightly lower but the quality of decompressed image has increased.

6. CONCLUSIONS AND FUTURE WORK

According the experiments done so far, we have found that Spiral Architecture has a great potential in improving fractal image compression. Knowing the fact that there have been a large number of methods found to optimize fractal image compression on traditional image structure, we would try some of them on Spiral Architecture. Moreover, we may take advantage of spiral multiplication to find out the self-similarity in an image with less computational complexity. The following are some proposed methods:

1. Apply spiral multiplication to have a number of sub-images with 7^n pixels as range blocks. Define domain blocks as the sub-images with 7^{n+1} pixels obtained by spiral multiplication to form the domain pool. This method is expected to take advantage of the self-similarity introduced by spiral multiplication so that the time to search pairs between range and domain blocks will reduce significantly.
2. In order to have a more accurate domain pool, instead of averaging the neighboring seven pixels intensities to scale a domain block to be a codebook block, the medium value of these seven pixels could be used to represent their intensity.
3. Based on lots of experimental results, the larger errors between fractals and the original images always happen along the contour or edge of objects in the image. We are able to classify the range blocks into three categories by their frequency in intensity – shade, edge and midrange. During the search process, we then can enlarge the domain pool for range blocks with higher frequency.

In short, with the implement results it can be seen that introducing Spiral Architecture into fractal image compression has great future in improving the compression performance and a lot of researches exist in this area.

7. REFERENCE

[Barnsley1988] Barnsley, M., Fractal Everywhere, New York: Academic,1988.

[Barnsley1985] Barnsley, M. and S. Demko, Iterated Function Systems and the Global Construction of Factals, Royal Soc., London.

[Barnsley1993] Barnsley, M. and L. P. Hurd, Fractal Image Compression, AK Peters. Ltd,1993.

[Barnsley1988] Barnsley, M. and A. D. Sloan,A better way to compress images, BYTE: 215-223.1988.

[Barthel1994] Barthel, K. U. and T. Voye Adaptive fractal image coding in the frequency domain Porc. Int. Workshop Image Processing: 33~38,June 1994.

[Beaumont1990] Beaumont, J. M. Advances in block based fractal coding of still pictures Proc. IEE Colloq.: The Application of Fractal Techniques in Image Processing: 3.1~3.6,Dec, 1990.

[Fisher1995] Fisher, Y., Fractal Image Compression: Theory and Application, New York, Springer-Verlag New York, Inc.,1995.

[He1999] He, X., 2D-object Recognition with Spiral Architecture, PhD. Thesis, Faculty of Information Technology, University of Technology, Sydney1999.

[Hintz2003] T. Hintz and Q. Wu, Image Compression on Spiral Architecture, The International Conference on Imaging Science, Systems and Technology, Las Vegas, Nevada, USA.

[Jacquin1990] Jacquin, A. E. Fractal image coding based on a theory of iterated contractive image transformations Pro. SPIE: Vis. Commun. Image Processing 1360: 227~239,1990.

[Jacquin1993] Jacquin, A. E. Fractal image coding: a review Proceedings of the IEEE 81(10): 1451-1465,1993.

[Kreyszlg1978] Kreyszlg, E., Introductory Functional Analysis with Applications, New York: Wiley,1978.

[Schwartz1980] Schwartz, E. Computational Anatomy and Functional Architecture of Striate Cortex: A Spatial Mapping Approach to

Perceptual Coding Vision Research 20: 645-669,1980.

[Sheridan1996] Sheridan, P., Spiral Architecture for Machine Vision, PhD. Thesis, Faculty of IT, University of Technology, Sydney1996.

[Sheridan2000] Sheridan, P., T. Hintz, et al. Pseudo-invariant image Transformations on a hexagonal lattice Image and Vision Computing 18: 907-917,2000.

[Sheridan1991] Sheridan, P., T. Hintz, et al. Spiral Architecture in Machine Vision Australian Occam and Transputer Conference,1991.

[Umbaugh1996] Umbaugh, S. E., Computer Vision and Image Processing: A Practical Approach Using CVIP tools, Prentice Hall,1996.

[Wohlberg1999] Wohlberg, B. and G. d. Jager, A Review of the Fractal Image Coding Literature, IEEE Transaction on Image Processing.

[Woolley1995] Woolley, S. J. and D. M. Monro Optimum parameters for hybrid fractal image coding Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing 4: 2571~2574,1995.

[Wu2004] Wu, Q., X. He, et al., Virtual Spiral Architecture, The International Conference on Parallel and Distributed Processing Techniques and Applications.

Parameterization of Tubular Surfaces on the Cylinder

Toon Huysmans
Vision Lab, Dept. of Physics
University of Antwerp
Groenenborgerlaan 171
Belgium 2020, Antwerp
toon.huysmans@ua.ac.be

Jan Sijbers
Vision Lab, Dept. of Physics
University of Antwerp
Groenenborgerlaan 171
Belgium 2020, Antwerp
jan.sijbers@ua.ac.be

Brigitte Verdonk
Dept. of Math. and CS.
University of Antwerp
Middelheimlaan 1
Belgium 2020, Antwerp
brigitte.verdonk@ua.ac.be

ABSTRACT

In this paper we develop a method to parameterize tubular surfaces onto the cylinder. The cylinder can be seen as the natural parameterization domain for tubular surfaces since they share the same topology. Most present algorithms are designed to parameterize disc-like surfaces onto the plane. Surfaces with a different topology are cut into disc-like patches and the patches are parameterized separately. This introduces discontinuities and constrains the parameterization. Also the semantics of the surface are lost. We avoid this by parameterizing tubular surfaces on, their natural domain, the cylinder. Since the cylinder is locally isometric to the plane we can do calculations on the cylinder without loosing efficiency. For speeding up the calculation we use a progressive parameterization technique, as suggested in recent literature. Together, this results in a robust, efficient, continuous, and semantics preserving parameterization method for arbitrary tubular surfaces.

Keywords

parameterization, remeshing, geometry images, texture mapping

1. INTRODUCTION

Surface parameterization is a technique to convert a mesh, described using primitives like triangles, quadrilaterals, or polygons, into a parametric description of the surface. In most applications the surface is two-dimensional and it is embedded in a three dimensional space. Thus, a parameterization is a map from a two-parameter domain onto the three-coordinate surface.

During the last ten years, parameterization has become an important topic in computer science and especially in computer graphics. It has a variety of applications such as: texture-mapping [LPRM02, SGSH02], rendering acceleration [GGH02], morphing [Ale02, ZSH00], remeshing and level of detail [EHL+95, PH03, AMD02], surface fitting [BGK95], surface description [SD02] and form analysis [Sty01].

Most of the techniques in the literature are concerned with the parameterization of topological discs. The parameterization of surfaces of other topology is addressed by cutting the surface into one or more patches, of disc topology, and parameterizing the patches separately. This cutting constrains the parameterization process from the beginning and it also introduces discontinuities into the parameterization. For some applications, like global form analysis, morphing, and surface fitting, this is undesirable. The only way to parameterize a surface of non-disc topology, without cutting it, is by parameterizing it on a domain that has the same topology as the surface. For example, surfaces with spherical topology can be parameterized on the sphere [PH03, GGS03, GWC+03]. In [KS04] and [SAPH04] triangle surfaces are parameterized onto other triangle surfaces that share the same topology.

We are interested in parameterizing surfaces with cylindrical topology onto the cylinder. This is done by Zöckler et al. in their paper on morphing [ZSH00], where they parameterize the cylindrical surface in two stages: first they cut the surface and parameterize it onto the plane, and then the parameterization is glued back together and optimized on the cylinder. Since the surface is cut, distortions are introduced in the first optimization and therefore they have to optimize the parameterization a second time. For complex surfaces this method might not find the optimal parameterization. Our algorithm is different; we directly parameterize onto the cylinder without cutting the surface. This way our algorithm is capable of parameterizing

The Journal of WSCG, Vol.13, ISSN 1213-6964
WSCG2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency–Science Press

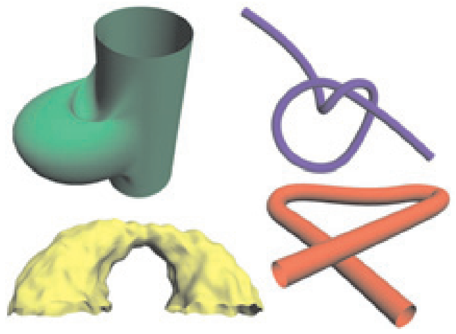


Figure 1: Some examples of tubular surfaces.

virtually any tubular surface with low distortion.

Goal The goal of this work is to find a one-to-one mapping from the surface of the cylinder to an arbitrary tubular surface. With 'tubular surface' we mean any elastic deformation of a sphere with two holes (boundaries), see figure 1 for a number of examples. The upper boundary of the cylinder should map to one of the boundaries of the tubular surface and the lower boundary of the cylinder should map to the other boundary of the tubular surface. The interior of the surface of the cylinder then has to be mapped to the interior of the tubular surface. This is illustrated in figure 2.

There are an infinite number of maps possible between the cylinder and a tubular surface, but we desire a map that is a balanced tradeoff between the following two properties. First, we require that the semantics of the cylinder are ported to the tubular surface. By this we mean that axial lines on the cylinder are mapped to lines that run in the axial direction on the tubular surface and that radial curves of the cylinder are mapped to curves that run in the radial direction on the tubular surface (see figure 3). Second, we also want that a uniform distribution of points on the cylinder is mapped to a quasi uniform distribution of points on the tubular surface. The results in section 4. will show that minimizing the stretch [SGSH02] of the map, will produce a map with a balanced tradeoff between the semantics and the uniformity property.

The remainder of this paper is divided into the following sections: Section 2. contains some theory about parameterizations and the cylinder that is important for the rest of the paper. Section 3. explains our approach to the parameterization of tubular surfaces on the cylinder. Section 4. shows some results obtained with an implementation of our technique. Some surfaces together with their cylindrical parameterization and also some cylindrical geometry images are shown. Section 5. concludes the paper and suggests directions of future research.

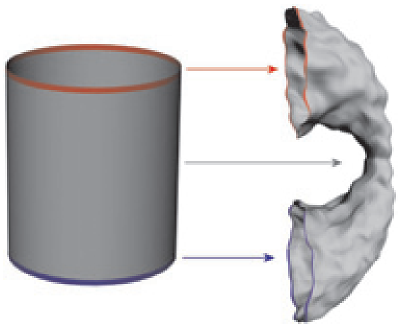


Figure 2: The upper boundary of the cylinder is mapped to the red boundary of the tubular surface and the lower boundary is mapped to the blue boundary of the tubular surface. The interior of the cylinder surface is mapped to the (grey) interior of the tubular surface.

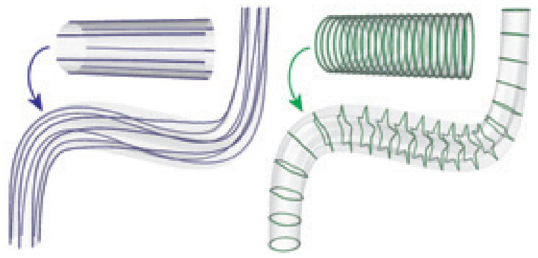


Figure 3: Semantics of the cylinder: Axial and radial lines on the cylinder are mapped to axial and radial lines on the tubular surface.

2. THEORY

This section introduces some basic differential geometry notions and explains some of the geometric properties of the cylinder, which are important to understand the rest of the paper. Most of this can be found in an elementary differential geometry book, for example [dC76].

Parameterization Informally, a parameterization of a surface $\mathcal{M}$ is a bijective map from a domain $\mathcal{D}$ to the surface $\mathcal{M}$. Mostly, $\mathcal{D}$ is a simple mathematical surface, for example the plane [SGSH02] or the sphere [PH03].

More formally, a parameterization of the surface $\mathcal{M}$, on the domain $\mathcal{D}$, is a *homeomorphism* $\mathbf{\Phi}$ between $\mathcal{D}$ and $\mathcal{M}$. The domain $\mathcal{D}$ is chosen so that it is *homeomorphic* to $\mathcal{M}$. This leaves us to explain the terms homeomorphic and homeomorphism: suppose $\mathcal{D}$ and $\mathcal{M}$ are topological spaces, and $\mathbf{\Phi}$ is a function from $\mathcal{D}$ to $\mathcal{M}$. Then $\mathbf{\Phi}$ is a homeomorphism iff the following holds:

- $\mathbf{\Phi}$ is a bijection;
- $\mathbf{\Phi}$ is continuous;
- the inverse function $\mathbf{\Phi}^{-1}$ is continuous.

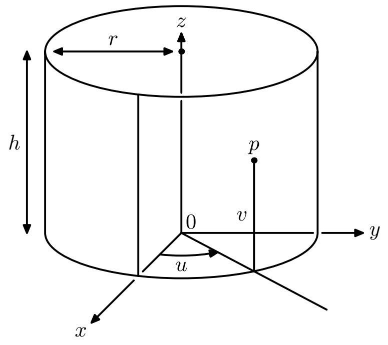


Figure 4: The right circular cylinder of radius r and height h. A point p on the lateral surface of the cylinder is defined by a cylindrical coordinate pair (u, v).

If there exists a homeomorphism $\mathbf{\Phi} : \mathcal{D} \mapsto \mathcal{M}$, then $\mathcal{M}$ is said to be homeomorphic to $\mathcal{D}$; $\mathcal{D}$ is also homeomorphic to $\mathcal{M}$, since $\mathbf{\Phi}^{-1}$ is a homeomorphism.

In our specific setting of parameterization of tubular surfaces, we choose $\mathcal{D}$ as the surface of the cylinder and $\mathcal{M}$ the surface of the tubular object. We say that $\mathbf{\Phi}$ is a cylindrical parameterization of the tubular surface $\mathcal{M}$.

So, in this paper we are concerned with the automatic construction of such a homeomorphism for any surface homeomorphic to the cylinder.

The Right Circular Cylinder There are many definitions for the concept cylinder, but we choose a specific one: the right circular cylinder. This cylinder is depicted in figure 4. The base of the right circular cylinder is a circle of radius r and the centers of the sections form a straight line perpendicular to the base of the cylinder. We choose the lateral surface of the cylinder as our parameterization domain $\mathcal{D}$; it is parameterized by $\mathbf{c} : U \mapsto \mathbb{R}^3$:

$$\begin{aligned} U &= \{(u, v) \in \mathbb{R}^2 | 0 \le u < 1, 0 \le v \le 1\} \\ \mathbf{c}(u, v) &= (r\cos(2\pi u), r\sin(2\pi u), v). \qquad (1) \end{aligned}$$

Geodesic Triangulation of the Cylinder In this work we are only concerned with the parameterization of piecewise linear triangle surfaces. This has the interesting side effect that we do not have to calculate the parameterization $\mathbf{\Phi}$ for every point explicitly. If we define the parameterization for the vertices and the edges of the triangle surface, then the parameterization of all other points can be found using interpolation.

We choose the parameterization of an edge between two vertices on the surface to be a geodesic of the cylinder that connects the parameterization of those two vertices. We choose a geodesic because it is a locally length minimizing curve. On the cylinder, each geodesic γ is a helix, a circle parallel to the base, or a line perpendicular to the base, defined by:

$$\begin{aligned} \gamma(t) &= (r\cos(at+b), r\sin(at+b), ct+d), \\ & \quad a, b, c, d \in \mathbb{R}. \end{aligned}$$

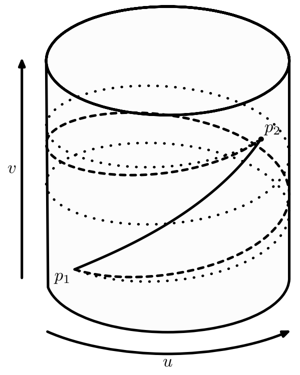


Figure 5: Three geodesics of the cylinder between points p_1 and p_2. The solid is the shortest geodesic, the dashed adds one turn in the positive u-direction and the dotted adds two turns.

There are an infinite number of geodesics between any two points on the cylinder, each with a different number of turns or a different direction. In figure 5 there are three geodesics (helices) all connecting the same two points. The solid line is the shortest geodesic of all possible geodesics between p_1 and p_2. It is important to specify the geodesic for each edge in the parameterization. How we do this will become clear in section 3.

If we parameterize the vertices and the edges of the surface onto the cylinder with the same connenctivity as the surface mesh and if the resulting triangles on the cylinder are not overlapping, then we get a triangulation of the cylinder. This, we call a *geodesic triangulation* because the edges of the triangles are geodesics of the cylinder. Such a triangulation induces a map from points on the cylinder to points on the surface. It is clear that this map is bijective, continuous, and its inverse is also continuous: it is a homeomorphism and thus also a parameterization.

Local Cylinder-Plane Isometry As we already mentioned in the introduction, most parameterization algorithms have the plane as their parameterization domain; calculations done in this plane are mostly fast and easy. When the parameterization domain is not flat, the computations can be harder. For example in [PH03] the domain is the sphere and calculations involve numerical integration which slows down the parameterization process.

A surface is flat if it has zero gaussian curvature, for example the plane. To check that the cylinder is flat, we compare the first fundamental form of the cylinder with the first fundamental form of the xy-plane.

If they coincide, then the cylinder is isometric to the plane. As a concequence the cylinder has zero gaussian curvature and therefore is flat.

The cylinder is parameterized by $\mathbf{c}$ in (1) and the xy-plane on the other hand is parameterized by $\mathbf{p} : \mathbb{R}^2 \mapsto \mathbb{R}^3$:

$$\mathbf{p}(u, v) \quad = \quad (u, v, 0) \tag{2}$$

The first fundamental form of the cylinder is given by:

$$E_c = \left|\frac{\partial \mathbf{c}}{\partial u}\right|^2 = r, \quad F_c = \frac{\partial \mathbf{c}}{\partial u} \cdot \frac{\partial \mathbf{c}}{\partial v} = 0, \quad G_c = \left|\frac{\partial \mathbf{c}}{\partial v}\right|^2 = 1$$

The first fundamental form of the xy-plane is given by:

$$E_p = \left|\frac{\partial \mathbf{p}}{\partial u}\right|^2 = 1, \quad F_p = \frac{\partial \mathbf{p}}{\partial u} \cdot \frac{\partial \mathbf{p}}{\partial v} = 0, \quad G_p = \left|\frac{\partial \mathbf{p}}{\partial v}\right|^2 = 1$$

We can see that the only difference between the first fundamental forms is between $E_c = r$ and $E_p = 1$, but if we choose the radius of the cylinder to be $r = 1$ then the first fundamental forms coincide.

This means that the cylinder and the plane are *locally* isometric, yet they are not *globally* isometric because the plane and the cylinder are not homeomorphic. This local isometry can be grasped visually: by cutting the cylinder along a line perpendicular to the base, the cylinder can be unfolded to the plane without distortion. This property has several interesting consequences.

First of all, due to the isometry, every geodesic of the cylinder corresponds to a geodesic of the plane and vice versa. The geodesics of the plane are all straight lines, so a geodesic triangle of the cylinder corresponds to a straight-line triangle in the plane. Now, if we have to apply an algorithm to geometry on the cylinder, we can simply transform the geometry from the cylinder to the plane by the isometry and apply ordinary algorithms to the planar geometry. Once the result is obtained in the plane, it can be transformed to the result on the cylinder.

Another advantage of working with the corresponding plane geometry, is that we can use ordinary 2d-optimization algorithms, like the conjugate gradient algorithm, for optimization of the vertex positions. Also, during the optimization of the vertex positions, we have to calculate the distortions of a geodesic triangle caused by the parameterization. But thanks to the isometry, this distortion can be calculated using the corresponding triangle in the plane. This means that we can calculate the distortion, using the formulas from planar parameterization algorithms as in [SGSH02].

3. METHOD

The parameterization can be computed in two steps: first, find a geodesic triangulation of the cylinder using the connectivity of the tubular surface so that we have a homeomorphism. Second, optimize the positions of the vertices on the cylinder so that the distortion of the parametrization is minimized. Although this method is correct, it has the disadvantage that the optimization step is very hard and that it will probably get stuck in a bad local minimum. It is better to construct the parameterization in a hierarchical way, as in [HGC99] and [SGSH02]. The hierarchical parametrization utilizes the progressive mesh of the tubular surface and proceeds as follows: first the base mesh is parameterized and then we iteratively split the vertices and locally optimize their placement while avoiding foldovers. This method is outlined in the following algorithm:

Algorithm 1 Parameterize($\mathcal{M}$)

```
 1: $(\mathcal{M}_0, \{\text{vsplit}_1, \ldots, \text{vsplit}_m\}) = \text{ProgMesh}(\mathcal{M})$;
 2: $\mathcal{P}_0 = \text{ParameterizeBaseMesh}(\mathcal{M}_0)$;
 3: $i_{prev} = 0$;
 4: **for** $i = 1$ to $m$ **do**
 5:     $\mathcal{P}_{i-1} \xrightarrow{\text{vsplit}_i} \mathcal{P}_i$;
 6:     place new vertex $v$ inside kernel of its 1-ring;
 7:     OptimizePlacement($v$);
 8:     **if** $\#P_i > factor \times \#P_{i_{prev}}$ **then**
 9:         OptimizePlacement() for all $v$ in $\mathcal{P}_i$;
10:         $i_{prev} = i$;
11:     **end if**
12: **end for**
13: OptimizePlacement() for all $v$ in $\mathcal{P}_m$;
14: return $\mathcal{P}_m$;
```

We will now go into more detail:

Progressive Mesh Construction We first construct the progressive mesh [Hop96] of our surface $\mathcal{M}$ using a quadratic error metric. A progressive mesh is constructed by successively collapsing an edge of the mesh; the next edge to collapse is chosen so that the introduced quadratic error metric is minimal and that the collapse does not violate any constraints. We impose three constraints:

- Both boundaries of the tubular surface should have at least three vertices.
- Collapse a boundary vertex only into a vertex of the same boundary. This avoids that a vertex of one boundary is collapsed into a vertex of the other boundary, which would generate a degenerate mesh. We also require this out of convenience, because this way we know that an internal vertex can never be split into a boundary vertex which eases the parametrization process.
- The third constraint says that there may be no triangles with all three vertices on one boundary,

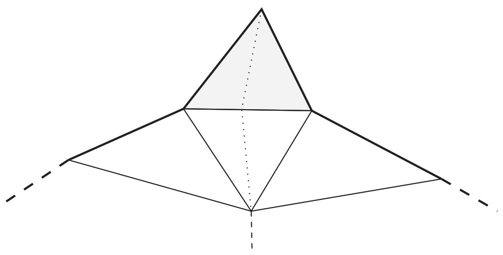


Figure 6: The shaded triangle is violating the third constraint because its three vertices are on one boundary (bold line). We remedy this by splitting the edge that is not on the boundary, this results in two extra triangles.

because the parametrization of such a triangle would result in a triangle of zero area, which is undesirable.

We have added these constraints to the progressive mesh construction algorithm. We also require that the original surface does not violate any of the above constraints. If the first or the second constraint is violated in the original surface, then we reject the mesh. When the third constraint is violated in the original surface, we have a remedy: split the edge of the triangle that is not on the boundary, this is depicted in Figure 6.

The progressive mesh is represented by the base mesh $\mathcal{M}_0$ and a set of vertex splits $\{\text{vsplit}_1, \text{vsplit}_2, \ldots, \text{vsplit}_m\}$, which are the reverse operations of the edge collapses in reversed order.

Base Mesh parameterization If we construct the progressive mesh of a tubular surface, as explained in the previous section, then the base mesh $\mathcal{M}_0$ will be an open prism with a triangle as its base. This mesh is depicted in Figure 7. Each of the three square sides of the base mesh consists of two triangles. This mesh is parameterized on the cylinder by separating the three points on both boundaries by 120 degrees. Then the vertices on one of the boundaries are rotated until three of the edges, connecting both boundaries, are perpendicular to the base of the cylinder. The (u, v)-coordinates of the parameterized base mesh are displayed in Figure 7.

We also have to determine the parameterization of the edges; the parameterization of an egde is a geodesic of the cylinder. A geodesic can be determined by specifying its direction (negative or positive u-direction) and its number of turns (0,1,2,...). The parameterization of an edge $((u_1, v_1), (u_2, v_2))$ of the base mesh is always a geodesic with 0 turns, because the length of the edge is at most $1/3$ in the u-direction, and its direction is positive if $u_1 <= u_2$ and negative otherwise.

In this way we obtain a geodesic triangulation of the

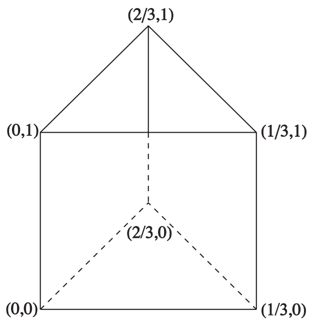


Figure 7: Base mesh of the progressive mesh for a tubular object, together with the (u, v)-coordinates of its parameterization.

cylinder with the connectivity of the base mesh, and thus we have found the parameterization $\mathcal{P}_0$ of the base mesh.

Next Level Parameterization Once we have the parameterization of the base mesh, we start by iteratively refining the resolution of the parameterization using vertex splits until we end up with the parameterization $\mathcal{P}_m$ of $\mathcal{M}_m = \mathcal{M}$. The step we explain here is generic and takes us from a parameterization $\mathcal{P}_i$ to the parameterization $\mathcal{P}_{i+1}$.

We start by applying vsplit_i to $\mathcal{P}_i$, this results in a new vertex v. In order to avoid foldovers we have to put this vertex inside the kernel of the polygon formed by the triangles of its 1-ring. We will put the vertex v in the center of this kernel. The kernel is computed in the plane using the isometry. But first we will have to transform the polygon to the plane. We set the y-coordinates of the planar polygon equal to the v-coordinates of the cylindrical polygon. We then choose one point of the polygon as a reference and set its x-coordinate to 0. Then we determine the x-coordinate of the next vertex in the polygon by calculating the u-length of the geodesic edge between this vertex and the reference vertex(taking into account the direction and the number of turns of the geodesic). Then the u-coordinate of the next vertex is determined relative to the previous vertex until al vertices are assigned a u-coordinate and we have obtained the planar version of our geodesic polygon.

We construct the kernel of this planar polygon using line clipping and calculate its geometric center. Then we transform the center to the cylinder and use this coordinate as the placement for v. We transform the center from the plane to the cylinder using a vertex of the polygon as a reference. We also update the direction and number of turns of each of the edges incedent to the vertex v. We now have a parameterization of $\mathcal{M}_{i+1}$.

In order to obtain a parameterization that is a balanced trade-off between the semantics and the uniformity property (see section 1.), we have to optimize the parameterization. After we have split a vertex, the placement of the new vertex v will be optimized, then we optimize the placement of each of its neighbours and we end with optimizing the new vertex v again. Also, when the number of vertices in the parameterization has increased with a factor (for example 1.5), we do this optimization for each of the vertices of the parameterization. A single vertex is optimized by the following steps:

1. transform the vertex v and its 1-ring polygon to the plane;
2. use the current position of v as an initial guess for the optimization;
3. minimize the symmetric version of the geometric stretch of the barycentric map summed over the 1-ring triangles as defined in [SGSH02]. The calculation of geometric stretch is based on the Jacobian of the barcentric map, since the Jacobian is invariant to isometry we can calculate the stretch using the planar triangles instead of the geodesic triangles of the cylinder. The optimization of the metric is also done in the plane using a standard 2D-optimization routine, while constraining the position of v to the kernel of the 1-ring polygon in order to avoid foldovers;
4. in the end, transform the optimized position of v back to the cylinder and update the direction and number of turns of each geodesic incident to the optimized vertex.

There is one remark we have to make: when we parameterize tubular objects that are very long in the axial direction compared to the radial direction, the parameterization gives bad results since the triangles are compressed in the axial direction to fit on the cylindrical domain of length 1. This can be remedied by changing the length of the cylindrical domain. For example when parameterizing a tubular surface that is twice as long in the axial direction as it is in the radial direction, we have to set the length of the cylindrical domain to the double of the radius of the cylindrical domain. Currently this length has to be estimated by the user, in the future we hope to automate this.

Sampling the Parameterization Up till now we have only defined the parametrization of the vertices and the edges. If we would like to sample the parametrization at arbitrary points of the cylindrical domain, then we have to define the parameterization at every point. As we have seen in the previous section, the interior of a triangle is parameterized using the barycentric map. Therefore if we want to sample the parameterization on the point (u, v) we only have to find the geodesic triangle on the cylinder that contains the point (u, v), the value of the parametrization is then determined by the barycentric map from that triangle to the corresponding triangle on the tubular surface. To find the triangle containing the point (u, v), we utilize a point location technique using bounding volume hierarchies [GLM96].

surface	# faces	h	time (s)
knot	12768	7.0	55
pipe	23248	3.0	117
head	11538	1.0	64
bow	33702	2.0	143
spring	19152	10.0	89
screwdriver	53782	3.0	268

Table 1: parameterization results of surfaces from $11K$ to $50K$ faces within 1 to 5 minutes. The height of the cylinder (h) ranges from 1 to 10 times the radius of the cylinder.

4. RESULTS

We have tested an implementation of the algorithm on different tubular surfaces, the results are summarized in Table 1. We have parameterized surfaces with $11K$ to $50K$ faces, within 1 to 5 minutes on a $1.2GHz$ computer. In Figure 8 the parameterized surfaces are displayed. The parameterization is revealed by the texture of the surface, the blue and the red lines on the surface are the iso-parameter lines for respectively the u and v parameter. Also, the quality of the parameterization can be derived from this figure. First, the semantics of the cylinder are ported to the surfaces because the red lines (iso-u) are running in the radial direction and the blue lines (iso-v) are running in the axial direction. Second, the distortion is kept low, which we can see because the iso-parameter lines form squares or rectangles. However, the size of the squares or rectangles can vary on the same surface (for example on the screwdriver), which tells us that the parameterization suffers from scale distortion. This is unavoidable when parameterizing onto the cylinder. This is also the reason why we did not add the stretch of the parameterizations in Table 1, there would be no point in comparing them.

Once a parameterization is obtained it is also possible to generate a geometry image [GGH02] of the surface. Geometry images are a completely regular image based surface representation with implicit connectivity. They have a number of applications: hardware accelerated rendering, adaptive remeshing, compression, etc. Our geometry images are constructed by

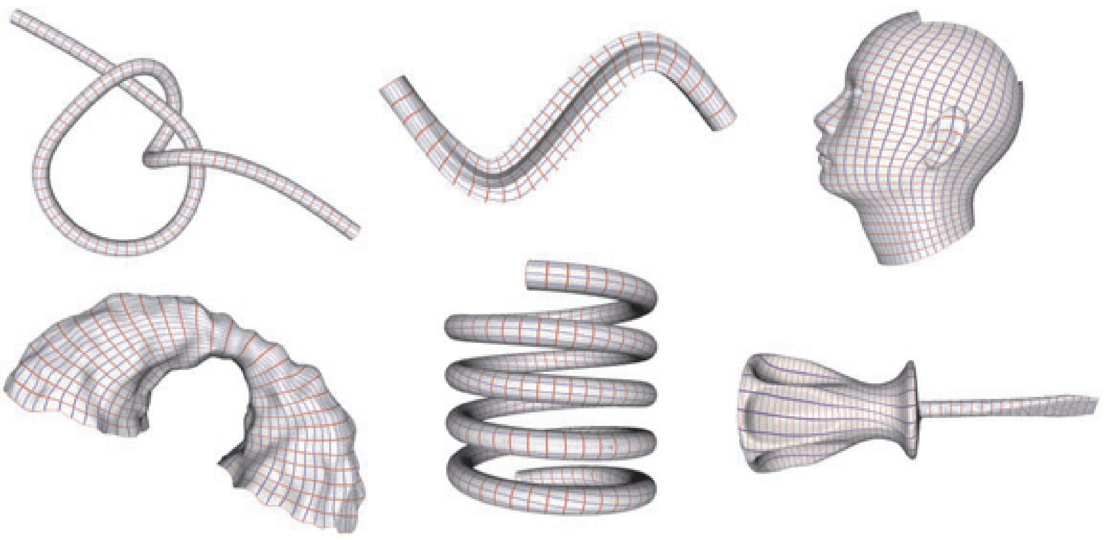


Figure 8: parameterization results, from left to right and from top to bottom: a knot, a pipe with a cross section that morphs from a circle to a star and back to a circle, a head with the bottom of the neck open and a square hole in the top, a bow, a spring, and a screwdriver with a hole in the tip and in the top. The texture visualizes the iso-parameter lines of the parameterization.

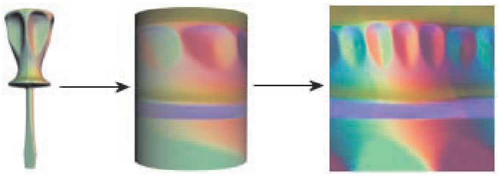


Figure 9: A geometry image is generated by parameterizing the surface on the cylinder and unfolding the cylinder to the plane.

sampling the cylindrical parameterization on a regular (u, v)-grid and unfolding this grid to the plane, this process is visualized in Figure 9. One side-effect of cylindrical geometry images is that the u and v resolution can be controlled separately. This results in rectangular geometry images, which can be useful for elongated surfaces. Figure 10 displays the cylindrical geometry image and normal map of the bow surface at different resolutions.

5. CONCLUSION

In this paper we propose a new method to parameterize tubular surfaces. We parameterize the surfaces on their natural domain i.e., the cylinder, which avoids cutting. By minimizing our symmetric stretch metric we obtaine a parameterization with a balanced trade-off between cylindrical semantics and uniform sampling. We test the algorithm on several surfaces and summarize the results. We also propose a new kind of geometry images for cylindrical surfaces and show some results.

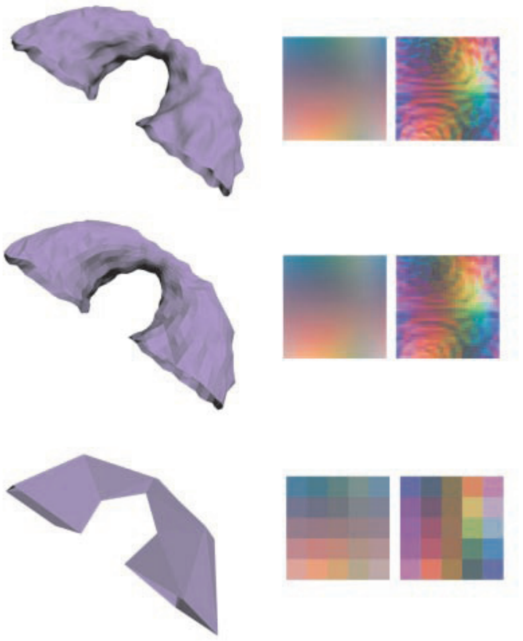


Figure 10: Cylindrical geometry image and normal map of the bow surface at following resolutions: 257×257, 33×33, 5×5. The remesh of the geometry image at each resolution is displayed on the left and is flat shaded.

Future directions of research: we would like to find a method to automatically determine the optimal length of the cylinder when parameterizing a surface or adapt the parameterization method so that we can use a cylinder of unit length without artifacts. We would also like to extend the parameterization method for feature correspondance. This should enable us to use the tubular parameterization for shape analysis of biological tubular objects as for example the human cochlea.

ACKNOWLEDGMENTS

This work was financially supported by the Flemish Institute for the Promotion of scientific and Technological Research in the Industry (I.W.T.) - Brussels, Belgium and the Fund for Scientific Research (F.W.O.) - Flanders, Belgium.

References

[Ale02] Marc Alexa. Recent advances in mesh morphing. *Computer Graphics Forum*, 21(2):173–196, June 2002.

[AMD02] Pierre Alliez, Mark Meyer, and Mathieu Desbrun. Interactive geometry remeshing. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 347–354. ACM Press, 2002.

[BGK95] Ch. Brechbühler, G. Gerig, and O. Kübler. Parametrization of closed surfaces for 3-D shape description. *Computer Vision and Image Understanding: CVIU*, 61(2):154–170, March 1995.

[dC76] Manfredo P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976. 503 pages.

[EHL+95] Matthias Eck, Hugues Hoppe, Michael Lounsbery, Tom Duchamp, Tony DeRose, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. *Proc. Conf. on Computer Graphics (SIGGRAPH '95)*, pages 173–182, January 1995.

[GGH02] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. In John Hughes, editor, *SIGGRAPH 2002 Conference Proceedings*, Annual Conference Series, pages 335–361. ACM Press/ACM SIGGRAPH, 2002.

[GGS03] C. Gotsman, X. Gu, and A. Sheffer. Fundamentals of spherical parameterization for 3d meshes. *ACM Transactions on Graphics*, 22, 2003.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proc. SIGGRAPH '96*, pages 171–180, 1996.

[GWC+03] X. Gu, Y. Wang, T. Chan, P. Thompson, and S. Yau. Genus zero surface conformal mapping and its application to brain surface mapping. In *Information Processing Medical Imaging 2003*, 2003.

[HGC99] K. Hormann, G. Greiner, and S. Campagna. Hierarchical parametrization of triangulated surfaces. In B. Girod, H. Niemann, and H.-P. Seidel, editors, *Proceedings of Vision, Modeling, and Visualization 1999*, pages 219–226, Erlangen, Germany, November 1999. infix.

[Hop96] Hugues Hoppe. Progressive meshes. *Proc. 23rd Int'l. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH'96)*, pages 99–108, January 1996.

[KS04] Vladislav Kraevoy and Alla Sheffer. Cross-parameterization and compatible remeshing of 3d models. *ACM Trans. Graph.*, 23(3):861–869, 2004.

[LPRM02] Bruno Levy, Sylvain Petitjean, Nicolas Ray, and Jerome Maillot. Least squares conformal maps for automatic texture atlas generation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 362–371. ACM Press, 2002.

[PH03] Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. *ACM Transactions on Graphics*, 22(3):340–349, July 2003.

[SAPH04] John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. Inter-surface mapping. *ACM Trans. Graph.*, 23(3):870–877, 2004.

[SD02] Jan Sijbers and Dirk Van Dyck. Efficient algorithm for the computation of 3D fourier descriptors. In Guido M Cortelazzo and Concettina Guerra, editors, *Proceedings of the 1st International Symposium on 3D Data Processing Visualization and Transmission (3DPVT-02)*, pages 640–643, Los Alamitos, CA, June 29–21 2002. IEEE Computer Society.

[SGSH02] Pedro V. Sander, Steven J. Gortler, John Snyder, and Hugues Hoppe. Signal-specialized parametrization. In *13th Eurographics Workshop on Rendering*. Eurographics Association, 2002.

[Sty01] Martin Styner. *Combined Boundary-Medial Shape Description of Variable Biological Objects*. PhD thesis, University of North Carolina, Dept. of Computer Science, June 2001.

[ZSH00] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Fast and intuitive generation of geometric shape transitions. *The Visual Computer*, 16(Issue 5):241–253, 2000.

Silhouette Enhanced Point-Based Rendering

José Luiz Luz, Luiz Velho, Paulo Cezar P. Carvalho

IMPA–Instituto Nacional de Matemática Pura e Aplicada
Estrada Dona Castorina, 110, 22460
Rio de Janeiro, RJ, Brasil
{josell, lvelho, pcezar}@visgraf.impa.br

ABSTRACT

With the recent advances in the 3D scanning field, the size of datasets to be displayed has increased up to billions of points. Typically, we have a dense, unstructured set of points without connectivity information. Most researchers have proposed the point-surfel association to represent the surface's geometry and to render it using a planar approximation for each point. This paper proposes an alternative approximation, where curved surface elements (c-surfels) are employed, in order to get better adaptation to the surface to be rendered. We also use texture mapping and blending, to produce a perceptually better visualization. Improvements caused by using curved surfels instead of planar ones are especially noticeable at the object's silhouette.

Keywords: Point-based Rendering, Graphics Data Structures, Texture Mapping.

1 INTRODUCTION

The problem of handling 3D datasets obtained from real-world objects has drawn the attention of the research community. Typically, we have a dense, unstructured set of points (sometimes, billions of them) without connectivity information. The techniques to treat these datasets have evolved, especially due to research on triangle meshes, since triangles are the most popular modeling primitives. Nevertheless, with the growing use of complex geometries the overhead associated with polygonal meshes is reaching prohibitive levels. As a consequence, other representations become more attractive.

More recently, there has been a trend to use point-based representations. Given the simplicity of points, they seem natural for modeling and rendering. We can obtain them from parametric representations (polygonal meshes, splines patches, subdivision surfaces) and non-parametric ones (implicit surfaces, fractals). Other representations use directly the point samples, such as particle systems, volumetric data in medical images, and image-based rendering.

Point-based representations can compensate for their lack of connectivity information, by spatial proximity between the points in a sufficiently dense sample, without causing loss of quality in the final image. With the texture mapping technique introduced by Catmull [Cat74] we can improve the visualization while keeping the object fundamental geometry, getting better results for planar surfaces or slightly curved surfaces. Moreover, blending operations can be used to reduce discontinuities in the texture mapping of overlapping surfaces.

This paper proposes an alternative approximation where curved surface elements (c-surfels) are employed, in order to get better adaptation to the surface to be rendered. We also use texture mapping and blending, to produce a perceptually better visualization. Improvements caused by using curved surfels instead of planar ones are especially noticeable at the object's silhouette.

We discuss related work in Sec. 2 and then describe the steps to build our primitives in Sec. 3. In Sec. 4, point out some factors that contribute to the use of c-surfels at the object's silhouette and show some results and applications, and in Sec. 5 we present some conclusions, and discuss limitations and future work.

2 RELATED WORK

Levoy and Whitted [LT85] in 1985 proposed the use of points as universal rendering primitives. The conceptual idea was to have a single element good enough to model and render any kind of object. The surface could be represented by points, considering it differentiable, and estimating the tangent plane and the normal from a small set of neighboring points.

About a decade later, in 1998, Grossman and Dayle [GD98] addressed object sampling from a set of orthographic views. They used a hierarchy of depth buffers to determine when a pixel is considered a hole or not.

Various researchers have published their ideas relative to point rendering and modeling. In 2000 three papers introduced the ground ideas for our work.

Pfister et al. [PZBG00] extended Grossman and Dayle's work by adding hierarchical level of detail (LOD) control and hierarchical visibility culling. They proposed the paradigm of surface elements (surfels) to efficiently render complex geometric objects. Surfels are primitives without explicit connectivity, with attributes such as depth, texture color, and normal. The objects are sampled from three orthogonal views and the sampling is stored in a octree. When rendering, the visible surfels and the holes are detected; the surface attributes are interpolated at the pixels that have samples.

Rusinkiewicz and Levoy [RL00] devised a rendering system called Qsplat. It allows real-time viewing of models consisting of hundred of millions of points samples. They used a bounding sphere hierarchy for hierarchical LOD control and culling and they employed splatting for surface reconstruction. The splats are oriented along the view plane and rendered in a back-to-front order.

Schauffer and Jensen [JS00] used small surfels to render point-based representations. They considered these surfels as tangent plane approximations and employed ray tracing to interpolate per-point attributes.

3 POINT RENDERING

There are many approaches to render objects from its point-based representation. We can distinguish two different proposals. The first renders the primitives as 0-dimensional points, while the second renders the primitives considering an area for each point.

We can also classify the algorithms to render point-based surfaces in two groups: those that do *forward mapping* and those that do *backward mapping*. The methods in the first group send points directly to rendering pipeline and compute their contributions to the pixels; thus we have projection from object-space to image-space. Splatting [Räs02] is an example of this type of algorithm. The methods in the second group compute for each pixel in the image the object that projects on it; therefore, we have projection from image-space to object-space. Ray tracing and polygon texture mapping are examples of this type of algorithm. Some algorithms use a combination of both techniques.

Point rendering requires information about point attributes such as position, normal, color, texture coordinates, etc. We may also associate an element of surface to a point, i.e. a surfel. The surface area at each point can be considered circular and characterized by a radius, that must be sufficiently large to ensure a hole-free reconstruction. We can store other attributes for a surfel, such as transparency and material properties.

In this paper we have as input a set of point samples on a smooth surface, which are assumed to be sufficiently dense so that the distribution of the points over the surface can be considered approximately uniform. We also assume that a normal is available at each point, and, in some cases, textures coordinates are also available. We do forward mapping and texture mapping, and regard each point as either a surfel or curved surfel (c-surfel), with the same fixed radius for all surfels which is computed before sending them to the rendering pipeline.

3.1 Building our primitives

A topological surface is a subset S of an Euclidean space $\mathbb{R}^3$, which is locally homeomorphic to the Euclidean space $\mathbb{R}^2$, that is, for each point $p \in S$ there is a spherical neighborhood $B^3_\varepsilon \subset \mathbb{R}^3$ with center p, in such a way that the subset $B^3_\varepsilon \cap S$ is homeomorphic to the open unit disk in the Euclidean plane (Figure 1).

$$B^2_1 = \left\{(x,y) \in \mathbb{R}^2; x^2 + y^2 < 1\right\}$$

Intuitively this definition says that a surface is obtained by overlapping several deformed pieces of the plane [VG03].

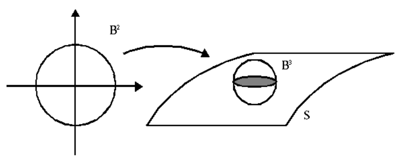


Figure 1: Homeomorphism

Let us represent these pieces by means of the function $\phi(r)$ ($r \in [0, \varepsilon]$), which is given by:

$$\phi(r) = \begin{cases} 0 & \text{, planar approximation} \\ 1 - e^{-\left(\frac{r^2}{h^2}\right)^2} & \text{, almost planar approximation} \end{cases}$$

The expression "almost planar" is used to represent our c-surfel, and h is a constant which defines the surfel curvature. From this function we can obtain a plateau-like surface (gaussian approximation) or a planar one, defined by the points $(r.\cos\theta, r.\sin\theta, \phi(r))$ $(\theta \in [0, 2\pi])$, which allows one to use it as a local approximation to a point, and allowing a perceptually better adaptation to the surface (Fig. 2).

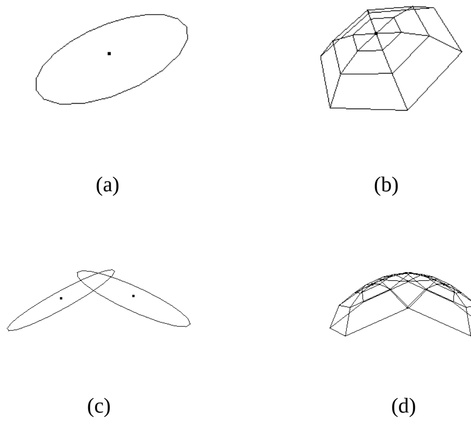


Figure 2: (a) planar surfel. (b) c-surfel. (c),(d) overlapping surfels

When we put these approximations on the surface we obtain overlapping surfels (c-surfels). Therefore, to give an appearance of continuity to the surface we use texture mapping and blending operations.

3.2 Texture mapping and blending

We map to our surfels a single texture with only one color, and opacity falling off radially according to a gaussian approximation. The alpha-value of pixel (i, j) (initially set to 1.0) is multiplied by a factor $f(i, j)$ given by:

$$f(i,j) = e^{-((i-x_0)^2+(j-y_0)^2)/d^2},$$

where

(i, j)	- position at texture;
(x_0, y_0)	- texture center;
d	- radial fall-off factor.

We map the texture considering initially $r \in (0, 1]$, and using the function:

$$(r.\cos(\tfrac{2k\pi}{n}), r.\sin(\tfrac{2k\pi}{n}), \phi(r))$$

$$\downarrow$$

$$(\tfrac{1}{2}\cos(\tfrac{2k\pi}{n}) + \tfrac{1}{2}, \tfrac{1}{2}\sin(\tfrac{2k\pi}{n}) + \tfrac{1}{2}).$$

Where n is the number of sides of the surfel and k is in the interval $(0, n]$. Thus, the surfel (c-surfel) center corresponds exactly to the texture center in a parametric space uv defined in $[0, 1]$x$[0, 1]$, and transparency is observed at the surfel border, as shown in Figure 3. Further the surfel is scaled in according to the computed radius.

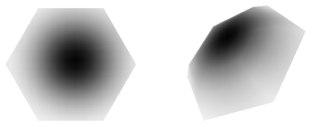


Figure 3: mapped texture surfel.

To handle overlapping surfels we need to know how to use alpha values to combine the currently processed color and the one previously stored at color-buffer.

The color c at position (x, y) in the final image is computed as a normalized weighted mean of contributions from mapped texture colors (surfels). The normalization is necessary since the weights (alpha values) do not necessarily constitute a partition of the unity at screen-space, due to irregular surfel position and the truncation of the ideal alpha mask (Figure 4).

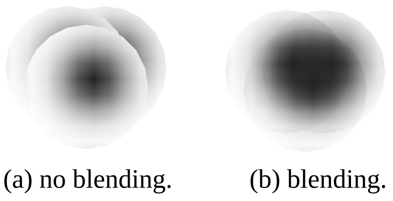


Figure 4:

We have:

$$c(x,y) = \frac{\sum_i c_i.w_i(x,y)}{\sum_i w_i(x,y)}$$

c_i	- i^{th} polygon color
$w_i(x, y)$	- weight at position (x, y)

We sort the points before rendering their corresponding surfels, since ordering affects smoothness at the final image, and we use a multipass rendering due to the interaction between blending and Z-buffering.

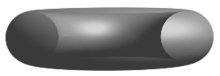


Figure 5: incorrect occlusion and blending

3.3 Visibility

Rusinkiewicz and Levoy [RL00] proposed a multipass rendering in OpenGL to ensure that both occlusion and blending happen correctly (Fig. 5). For the first pass, we render the surfel with an offset z_o away from the viewer. We do this only into the depth buffer. For the second pass we turn off the depth offset allowing depth comparison, without updating the depth buffer and writing to color-buffer. This steps blend together with the correct occlusion all surfels within a depth range z_o of the surface.

3.4 Surfel size

We need to compute the correct size of the surfels, which will be the same for all of them. To compute the size of the surfels we use eigenanalysis of the covariance matrix of a local neighborhood (Principal Component Analysis - PCA) to estimate local surface properties, as proposed by Pauly [MPK02]. Given a point cloud $P = \left\{p_i \in \mathbb{R}^3\right\}$, the covariance matrix C for a sample point p is given by

$$\mathbf{C} = \begin{bmatrix} p_{i_1} - \overline{p} \\ \cdots \\ p_{i_k} - \overline{p} \end{bmatrix}^T \cdot \begin{bmatrix} p_{i_1} - \overline{p} \\ \cdots \\ p_{i_k} - \overline{p} \end{bmatrix} \quad , i_j \in N_p$$

where $\overline{p}$ is the centroid of the neighbours p_{i_j} of p, and N_p is the index set of the k-nearest neighbours of the sample p. The principal components are the solutions to the following eigenvector problem:

$$C \cdot v_l = \lambda_l \cdot v_l \quad , l \in \{0, 1, 2\}$$

We use the eigenvalues and their corresponding eigenvectors to do a space partition. Since eigenvalues give a measure to the variation of the points in N_p, we take the eigenvector corresponding to the greatest eigenvalue, and define a splitting plane in a BSP-tree, that we use to perform hierarchical clustering.

Assuming that $\lambda_0 \le \lambda_1 \le \lambda_2$, the eigenvalue λ_0 describes the variation along the surface normal. We define

$$\sigma_n(p) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2},$$

as the surface variation at point p in a neighborhood of size n. If $\sigma_n(p) = 0$ all points lie in the plane. Then we use that as a subdivision criterion to locate clusters with exactly three non-collinear points (Figure 6). The size of the surfel corresponding to a cluster is the diameter of the smallest circle circumscribed to its associated triangle. Since we assume that our sample is approximately uniformly distributed, we expect that the triangles have approximately the same area, and use the average of all surfel sizes as a common size used in the rendering process. But if we have regions with uneven distribution, holes can appear on the surface.

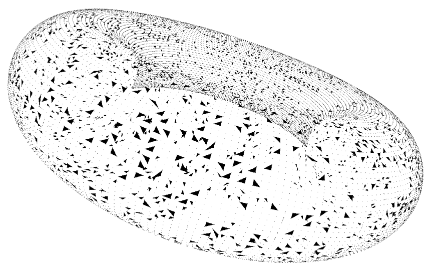


Figure 6: triangles on the surface obtained by clustering

4 VISUALIZING WITH OUR SURFELS

We associate to each point either a planar surfel or a c-surfel, which are constructed, and stored. Then they are oriented, translated and scaled accordingly to the point attributes. Some factors contribute to obtain good results when using c-surfels, Among them we can highlight the following ones:

- Overlapping c-surfels provides a better local approximation to the points on the surface, since they have an associated mesh, which provides more details;
- All normals in the c-surfel have the same orientation as the normal at the point; thus shading and blending operations give an appearance of continuity to the rendered surface.

Figure 7 shows a result using only c-surfels. However, as the c-surfel have a mesh, the computational cost due to rendering them can become high, depending on the number of polygons of the mesh. The table 1 shows the performance of our unoptimized C implementation for different c-surfels (Pentium IV 1.4 GHz 512 RAM).

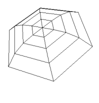
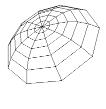
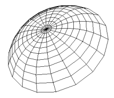
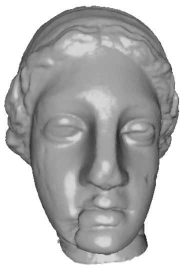
c-surfel	 24 polygons	 60 polygons	 200 polygons
Igea (134.345 points)	1.06 *fps*	1.00 *fps*	0.41 *fps*

Table 1: different c-surfel resolutions

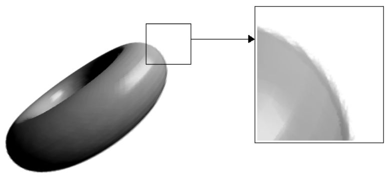
Instead of using only c-surfels, we propose to use both planar and curved surfels, and since using flat surfels at the silhouettes of the object may result in less



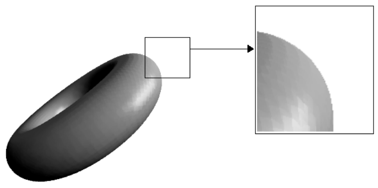
(a) igea (134.345 points).

Figure 7:

precise rendering (Figure 8), we use c-surfels at the silhouette points (and close to them) and flat ones for the rest of the surface as illustrated in Figure 9.



(a) flat surfels at the silhouette.



(b) c-surfels at the silhouette.

Figure 8:

Then if the angle between the vector from a point p to the observer's eye and the normal at this point is in the interval $[90^o - \epsilon, 90^o + \epsilon] \quad (\epsilon \in [0^o, 20^o])$, we use a c-surfel for this point, otherwise we employ a flat surfel. The table 2 shows some time measures using planar surfels with 6 sides, and c-surfels formed by 24 polygons.

We can see that using c-surfels at the object's silhouette does not increase the cost significantly, when comparing to the all planar case. Therefore, their use seems a good option to enhance the level of details at these regions. Some rendering results are shown in the figure 10.

When we have texture coordinates available for the points, we can do texture mapping by blending the texture image colors: given the texture coordinates we verify the corresponding color in the texture-space, and map it to the surfel (Figure 11).

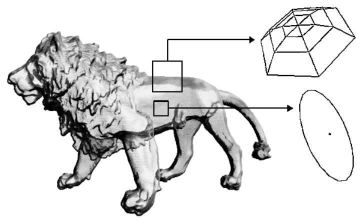


Figure 9: surfel and c-surfels

Model	planar surfel	c-surfel	planar surfel + c-surfel
Igea (134.345 points)	1.22 *fps*	1.06 *fps*	1.18 *fps*
Ball joint (137.059 points)	1.08 *fps*	0.96 *fps*	1.03 *fps*
Budha (389.347 points)	0.35 *fps*	0.3 *fps*	0.33 *fps*

Table 2: using planar and almost planar approximations

5 CONCLUSIONS

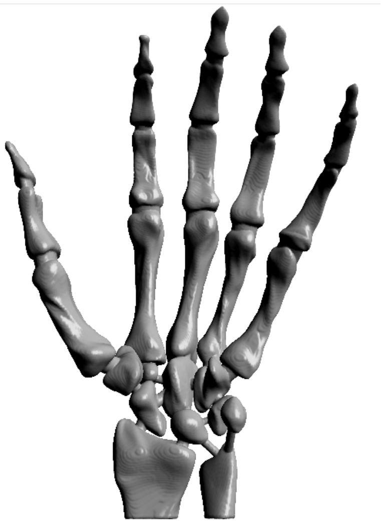
We proposed the use of curved surfels and texture blending to visualize a set of point samples, by exploring the adaptation to the surface when the c-surfels overlapping each other, which provides more details because of their mesh. Since the associated computational cost can be high, we used these c-surfels only at the models silhouette, without increasing the overhead very much.

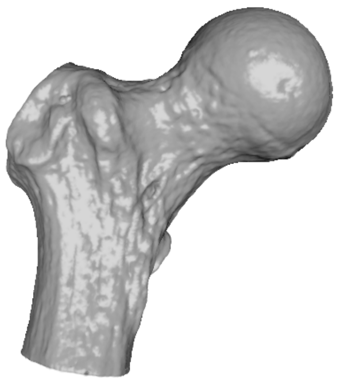
In future work we intend to improve the technique to handle difficult regions, such as those with high-curvature, perhaps storing a curvature information for each point, estimated in terms of its local neighborhood, and use that to adapt the surfel curvature.

6 ACKNOWLEDGMENTS

The authors are partially supported by CNPq research grants. This research has been developed in the VISGRAF Laboratory at IMPA. VISGRAF is sponsored by CNPq, FAPERJ, FINEP and IBM Brasil.

(a) budha (389.347 points).


(b) hand (327.323 points).


(c) ball joint (137.059 points).

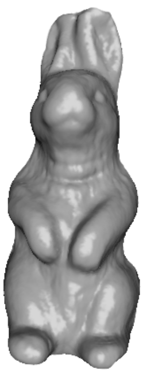
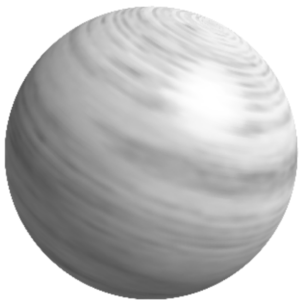
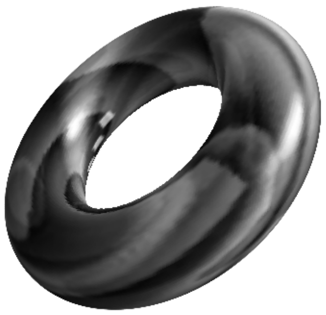

(d) rabbit (44.691 points).

Figure 10:



(a) texture-mapped sphere.



(b) texture-mapped torus.

Figure 11:

7 REFERENCES

[Cat74] E. E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Department of Computer Science, University of Utah, 1974.

[GD98] J. P. Grossman and William J. Dally. Point sample rendering. *Eurographics Rendering Workshop 1998*, pages 181–192, 1998.

[JS00] Henrik Wann Jensen and Gernot Schauffer. Ray tracing point sampled geometry. In Springer-Verlag, editor, *Rendering Techniques 2000*, pages 319–328. Eds. Peroche and Rushmeier, 2000.

[LT85] Marc Levoy and Whitted Turner. The use of points as a display primitive. Technical Report 85-022, University of North Carolina at Chapel Hill, 1985.

[MPK02] Markus Gross Mark Pauly and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. *In Proceedings IEEE Visualization 2002*, pages 163–170, Computer Society Press, 2002.

[PZBG00] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Barr, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of ACM SIGGRAPH 2000*, pages 335–342. ACM Press/ ACM SIGGRAPH/ Addison Wesley Longman, 2000.

[Räs02] Jussi Räsänen. Surface splatting: Theory, extensions and implementation. Master's thesis, Dept. of Computer Science, Helsinki University of Technology, 2002.

[RL00] Szymon Rusinkiewicz and Mark Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000*, pages 343 – 352. ACM Press/Addison-Wesley Publishing Co, 2000.

[VG03] L. Velho and J. Gomes. *Fundamentos de Computação Gráfica*. Impa, Rio de janeiro, 2003.

Shape from Silhouette: Image Pixels for Marching Cubes

Bruno Mercier
SIC Lab, Bât. SP2MI, téléport 2
Bd Marie et Pierre Curie
86962 Futuroscope Chasseneuil, France
mercier@sic.univ-poitiers.fr

Daniel Meneveaux
SIC Lab, Bât. SP2MI, téléport 2
Bd Marie et Pierre Curie
86962 Futuroscope Chasseneuil, France
daniel@sic.univ-poitiers.fr

ABSTRACT

In this paper, we propose to use image pixels for geometry reconstruction with a shape from silhouette approach. We aim at estimating shape and normal for the surface of a single object seen through calibrated images. From the voxel-based shape obtained with the algorithm proposed by R. Szeliski in [18], our main contribution concerns the use of image pixels together with marching cubes for constructing a triangular mesh. We also provide a mean for estimating a normal inside each voxel with two different methods: (i) using marching cubes triangles and (ii) using only voxels. As seen in the results, our method proves accurate even for real objects acquired with a usual camera and an inexpensive acquisition system.

Keywords
Geometry reconstruction from images, shape from silhouette, marching cubes.

1. INTRODUCTION

Since the early years of computer vision, much effort has been dedicated to automatically digitizing shape and reflectance of real objects. For instance, Stanford Digital Michelangelo [11] project aims at digitizing large statues, Callet. et al. [1] digitize statuettes and reconstruct plaster models covered with bronze. Hasenfratz et al. [7] use a digitize shape for placing a real actor in a virtual environment so that shadows and lighting be properly computed.

In most cases, acquisition hardware play a major role for reconstructing objects shape and research efforts have increased a lot during the last decade. Our concern is about objects only described with a set of calibrated photographs. Our final goal is to insert real objects (corresponding to lightfields/lumigraphs or described by a series of images) into virtual environments with proper lighting and global illumination. The whole problem is thus not only geometry estimation but also initial light sources position, reflectance properties for the real object as well as rendering process. This paper only addresses a small part of the whole work: geometry and normal estimation.

The basis of our work is the voxel-based shape from silhouette technique presented by Szeliski in [18]. We propose a new method for combining the marching cubes algorithm with image pixels for precisely recovering a triangular mesh corresponding to the object shape. We also propose two methods for estimating surface normal. As shown in the results, our method has a consequent impact on geometry. Based on this method, we have successfully recovered light sources geometry and object surface reflectance properties [14].

This paper is organized as follows. We firstly describe work most related to our concerns. Section 3 presents the acquisition system we use and work overview. We then detail our reconstruction and normal estimation method. Finally, we provide a series of results before we conclude.

2. RELATED WORK

The literature concerning geometry reconstruction is vast and this section only presents a quick run through the area for most closely related works.

Stereo vision [5, 2] uses two cameras located close one to another so that images of the object be slightly different. Internal and external camera parameters knowledge help to determine corresponding points on images and deduce depth with the help of textures, laser grids or structured light. *Shape from shading* methods aim at recovering objects shape with the assumption that surfaces are lambertian (or almost lambertian) [16, 8].

Shape from silhouette approaches [13, 3] are more adapted to our problem since we do not want to make any assumption about images. Objects can have glossy surfaces, with or without textures and we cannot use active laser grids or test patterns to reconstruct the object geometry. Most shape from silhouette methods rely on a voxel-based datastructure and the approach described by R. Szeliski in 1993 [18] is often used as a basis. For such methods, a well-known drawback concerns cavities. For example a coffee cup handle will be recovered since the hole can be deduced from the visual hull on the image, but the inside will be filled-up with material (unless there is no bottom). Improvements have been proposed for solving this problem with voxel coloring [15], space carving [9] or gradient voxel flux [4] with the assumption that objects surface is mostly lambertian.

From voxels, the marching-cubes algorithm can easily generate a triangular mesh corresponding to the object surface [17]. For example Hasenfratz et al. use such a reconstruction method for interactively integrate a real person into a virtual environment [7]. Our method focuses on such approaches and aims at improving the triangular shape accuracy. To achieve this goal, we propose to use image pixels for guiding the marching cubes algorithm and estimating accurate surface normal.

3. WORK OVERVIEW

Acquisition System

For this work, we used images of both virtual objects and real objects. Virtual objects are convenient for validating the method and providing result quality since camera parameters and object geometry are known. For real objects, we devised the acquisition system described in figure 1. Object and light sources are fixed on a turntable: a camera is located on a tripod with fixed aperture and shutter speed. During acquisition, camera position does not change. Every 5 degrees, two images are acquired with the same viewpoint. The first one is overexposed with an additional light source for separating object from background (a lambertian black cloth) while the second one is used for acquiring the actual object radiance. After one turn, the camera is raised of several centimeters. In practice, only 1 turn (76 viewpoints) is necessary for precisely recovering the object shape, but we also used this system for acquiring complete image-based objects.

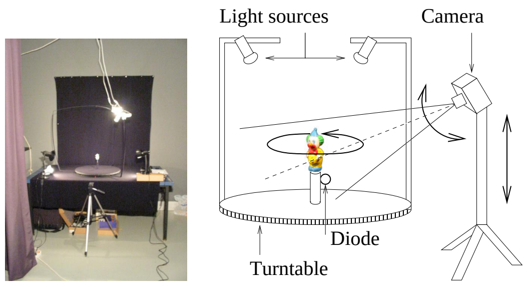


Figure 1: Acquisition system.

Image Processing

For separating background from the object, we use a powerful light source for overexposing images. The object is so bright that it is easy to determine the black background with a seed-fill algorithm even with dark regions on the object. Except background, only 2 connected regions remain: the object and the red diode used for estimating camera orientation. Background pixels are then set as perfectly black on images: (0,0,0) for R,G,B values.

The diode is used to determine the rotation axis of the turntable seen on photographs. Focal length is known a priori (fixed camera parameters) and 3D position is manually estimated. Orientation can thus be deduced from the red diode. We did not use any test pattern for estimating camera parameters.

Reconstruction Process

Our reconstruction method is composed of 3 main steps:

1. the shape from silhouette approach proposed by R. Szeliski in [18] provides a set of voxels;
2. a triangular mesh is generated from marching cubes and image pixels;
3. a normal is estimated for each voxel either with the marching cubes triangles or with a method using only voxels.

4. SHAPE FROM SILHOUETTE

Octree Construction

As a first (rough) approximation of the object geometry, we used the shape from silhouette approach proposed in [18]. With this approach, all the images are used iteratively to hierarchically sculpt the object. The shape initially corresponds to a voxel, recursively refined during the reconstruction process. For each view of the object, the octree voxels obtained so far are projected onto the image plane and compared to image pixels. When a voxel is seen outside the object for one image, it is actually marked as *out* (figure 2(a)); when a voxel is seen inside the object for all the images, it is marked as *in* (figure 2(b)); all the others are often seen inside the object and sometimes on the object boundary, they are marked as *ambiguous* (figure 2(c)) and subdivided into 8 sub-voxels. This process is repeated until no ambiguous voxels exist or a minimum size criterion has been reached. For our method, the algorithm should

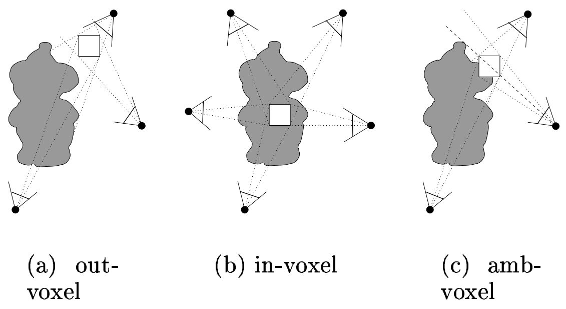


Figure 2: Voxel classification.

stop when ambiguous voxels correspond to a series of 4 (or 9) pixels for more reliable results. Figure 3 shows some results for a clown (real object). Note that for a 256x256 image, pixel resolution corresponds to a depth of 8 in the octree.

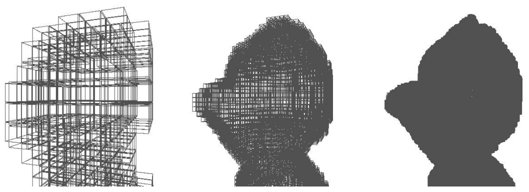


Figure 3: Reconstruction results for levels 5,7 and 9.

Obviously, all ambiguous voxels have the same size, according to the reconstruction process. At the opposite, voxels marked as *in* or *out* are not further subdivided and have various sizes.

Practical Aspects

For our method, voxel projection on the image plane is performed with raytracing. This will be further used with marching cubes in the following section. Each pixel corresponds to a ray originating at the image center-of-projection and going through the pixel (figure 4). These rays are called *pixel-rays* from now on. Pixel-rays corresponding to the object silhouette are called *in-rays* since they hit the object and rays corresponding to background are called *out-rays*.

Surface Thickness

The reconstruction process results in a set of ambiguous voxels called *discrete surface*. For marching cubes, this surface needs to be 6-connected which is not ensured by the previous algorithm. To achieve this goal, we propose to modify the discrete surface with an ad-

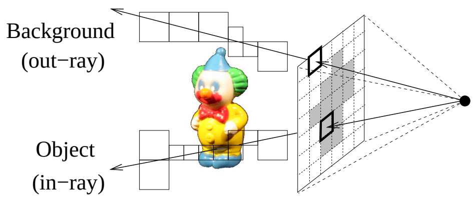


Figure 4: Pixel-rays.

ditional process applied every time ambiguous voxels are subdivided. Voxels classified as in or out can be reclassified as ambiguous when the discrete surface is not 6-connected (as illustrated in figure 5): when two adjacent in and out voxels do not correspond to the same hierarchy level in the octree, we choose to reclassify the smallest one, V_{small}, as ambiguous (since V_{small} parents had been longer classified as ambiguous). In the case of two voxels with similar size, we consider that out-voxels should remain outside the object according to the sculpture method seen above; the in-voxel will consequently be reclassified as ambiguous. A final operation reduces the surface thickness while keeping the 6-connection.

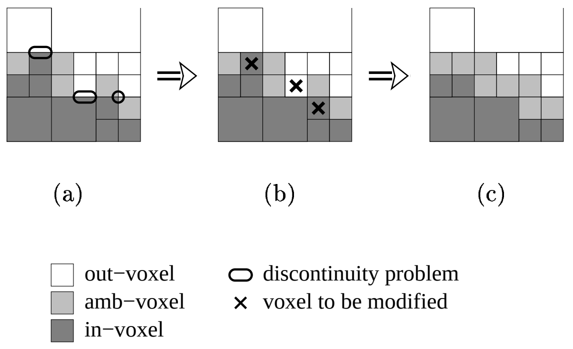


Figure 5: Voxels reclassification; a. "holes" in the surface; b. voxels to be modified; c. modified discrete surface (6-connected in 3D).

5. PIXELS FOR MARCHING CUBES

Original Marching Cubes

For reconstructing a triangular mesh from a set of ambiguous voxels, marching cubes [12] are obviously well-suited to the problem. Originally, the algorithm is dedicated to medical images and uses some density values (weights) associated with each voxel vertex; as a rough simplification, positive weights correspond to points inside the object and negative weights are outside the object. A linear interpolation provides the (estimated) intersection between each voxel edge and the actual object surface. According to these intersections, a triangular mesh can be defined for each voxel with a limited number of configurations (see figure 6).

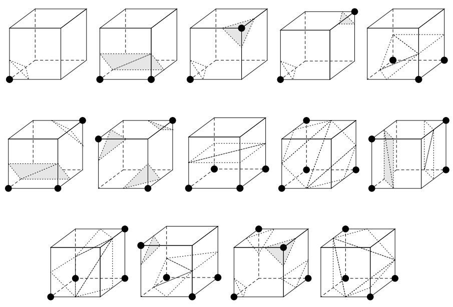


Figure 6: Marching cubes configurations (applied to ambiguous voxels only), black dots indicate vertices located outside the surface.

Since weighting values cannot be computed directly from the discrete surface, edges centers are often used for generating triangles. However, as explained in the following, it is possible to use pixel-rays with marching cubes so that triangles fit the model shape more precisely.

Refining the Method

As a first estimation, voxel vertices are classified according to neighborhood. Each voxel vertex of the 6-connected discrete surface has (at least) either one in-voxel or one out-voxel neighbor. For placing triangles in *ambiguous voxels*, we also use pixel-rays (figure 7).

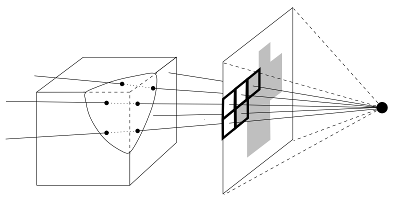


Figure 7: Out-rays should not touch the surface inside an ambiguous voxel; in-rays are not used.

Our algorithm firstly computes the intersection points between out-rays and voxel faces (figure 8). Then, a line corresponding to the object surface on each face is estimated (called *surface line* from now on). This line is placed as close as possible to all intersection points and out vertices. Let us consider the 2D convex hull associated to all these points $\{P_i\}$. It can be shown that for the general case, such a line corresponds to a convex hull line segment L_k. The surface line is thus chosen among $\{L_k\}$ so that the following function be minimized:

$$dist_k = \sum_i dist(P_i, L_k)^2$$

Unfortunately, in some cases, a surface line cannot be computed using the convex hull (see figure 8(d) for example). For such cases, the surface line is defined directly using in-vertices.

Finally, the intersection between surface lines and voxel edge defines the point (*mc-vertex*) used for marching cubes. Note that when the surface line contains a face vertex, our algorithm slightly shift the mc-vertex for avoiding degenerated triangles.

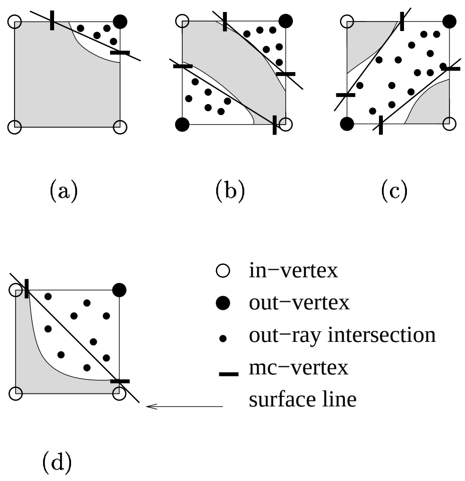


Figure 8: Intersection between out-rays and voxel faces - several cases; a. for usual cases; b. & c. with a disconnected surface; d. voxel vertices used as a boundary.

The defined surface line is bounded by voxel vertices according to classification so that surface continuity be maintained (figure 8(d)). For two adjacent faces, the corresponding surface lines have to be connected (see figure 9(a)). The only adequate choice consists in using the mc-vertex closer to the *in-vertex* since the other one would re-introduce out-rays intersection points inside the object surface. Finally, the same principle applies to adjacent voxels on the discrete surface (figure 9(b)).

6. SURFACE NORMAL

For some application, a normal can be needed for surface voxels (for instance, for estimating light sources properties and BRDF of the object [14]). We propose two methods: the first one uses the triangles generated by our adapted marching cubes algorithm while the second one relies on the discrete surface and its neighborhood.

Normal from Triangles

Inside each ambiguous voxel, several triangles define the object surface. We propose to compute the average triangle normal weighted by area.

Using triangles located in only one voxel leads to a

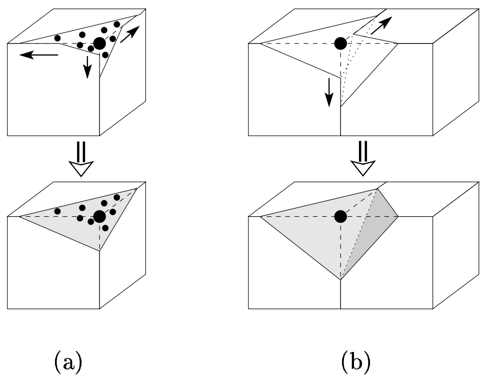


Figure 9: Surface continuity and mc-vertices.

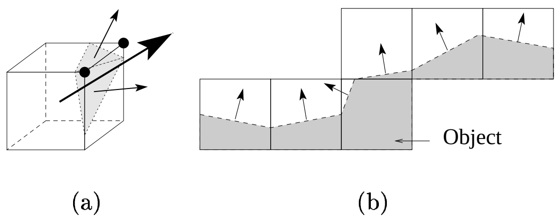


Figure 10: Normal from marching cubes triangles; the surface is *bumpy*.

bumpy normal, introducing artifacts for light sources estimation or during the rendering process (figure 10(b) and 11). This is why we propose to smooth normals according to neighbor voxels triangles. In our application, a parameter called *smoothing distance* is fixed by the user. Practically, with an octree depth equal to 7, our best results have been obtained with a smoothing distance set to 3 or 4 voxels (depending on the object geometry).

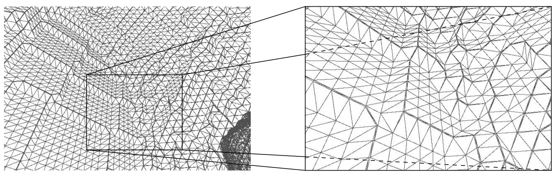


Figure 11: For curved surfaces, due to discrete representation, the marching cubes algorithm does not provide smooth surfaces.

Discrete Normal

Intuitively, a normal estimated from marching cubes should be quite representative of the object surface. However, it is also possible to define a normal in each voxel directly with the discrete surface. Particularly, if a surface mesh is not needed, acceptable results can be obtained. Normal is estimated according to out-voxels in the given neighborhood (figure 12): $\overrightarrow{N} = (\sum_{i=1}^{n} \overrightarrow{V_i})/n$, where $\overrightarrow{V_i}$ corresponds to the unit vector going from the current voxel center to the i^{th} neighbor voxel center; n is the number of voxels used. With this method, normals are defined by a fixed number of directions which could be useful for compression algorithms.

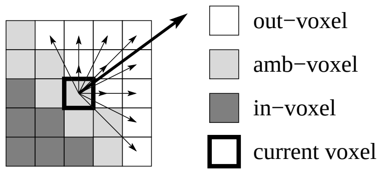


Figure 12: Normal estimation from voxels.

7. RESULTS

Object Shape

Before actually using our method with real objects, validation has been done with known geometric objects (a sphere and a cube). As shown in table 1, using image pixels with marching cubes (MC in the figure) improves consequently shape precision. For the experiments we made, the error is noticeably reduced compared to a marching cubes algorithm with edges centers. Note that when hierarchy depth increases, the two methods tend to provide the same results because the number of pixel-rays becomes lower. Our method will obviously be more accurate with a low-depth hierarchy.

For a 1-meter diameter Sphere			
Octree depth	5	6	7
Edges center MC	*10.8 mm*	*9.3 mm*	*8.9 mm*
Pixel-rays MC	*6.4 mm*	*6.7 mm*	*6.7 mm*
For a 1-meter width Cube			
Octree depth	5	6	7
Edges center MC	*26.4 mm*	*19.4 mm*	*16.7 mm*
Pixel-rays MC	*21.2 mm*	*19.1 mm*	*18.6 mm*

Table 1: Average distance between reconstructed triangles and actual object surface.

Note that a cube is the worst example. A polygon is difficult to reconstruct with a shape from silhouette approach (as any flat surface) since the camera viewpoint is never perfectly located on the polygon plane.

Normal Estimation

This paper describes two different methods for estimating normal inside each voxel. The first one is based on marching cubes triangles while the second one only relies on voxels. For each case, it is possible to smooth normal values according to a user-specified smoothing distance. For estimating normal quality, we compared the estimated normal with the actual (known) surface normal (table 2)

As for surface accuracy, a surface normal obtained with the help of pixel-rays is sensibly more precise (20-25%) than with using edges centers marching cubes or

For a 1-meter diameter Sphere			
Octree depth	5	7	
Smoothing distance	1	1	3
Discrete normal	*11.1°*	*12.1°*	*2.3°*
Edges center MC	*5.7°*	*6.3°*	*2.1°*
Pixel-rays MC	*3.3°*	*4.9°*	*1.8°*

Table 2: Average angles difference (degrees) between estimated normal and actual object normal.

the discrete surface. Our method provides a precision with an error less than 5° even without any smoothing. When smoothing, normal precision is about one degree (with a smoothing distance of 5 voxels). Note that with the discrete surface, the smoothing distance should not be less than 3 voxels.

Rendering using Normals

From geometry and normal we have generated new views. Triangles can be directly with Graphics Hardware (OpenGL). For example, figures 13, 15 and 14 show new images.

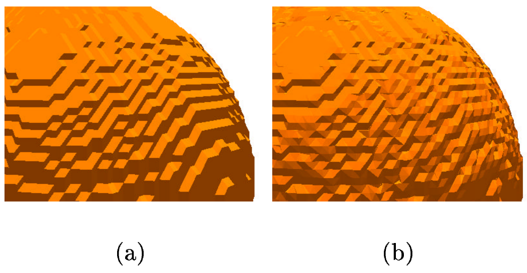


Figure 13: Rendering using triangles; a. with *edges-centers* marching cubes; b. with pixel-rays marching cubes.

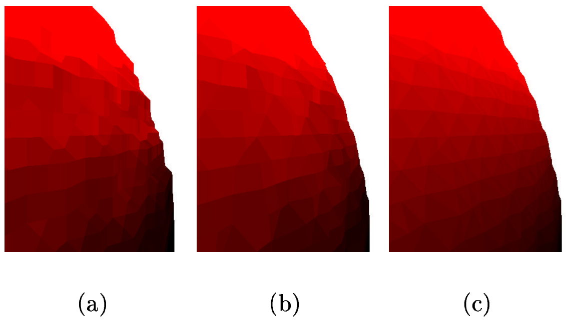


Figure 14: Rendering using voxel normal with smoothing; a. with *edges-centers* marching cubes note the bumpy surface on object silhouette; b. with pixel-rays, marching cubes contour is smoother; c. with actual sphere normal.

Rendering using Voxel Radiances

It is also possible with pixel-rays (in-rays) to estimate an average radiance emitted by the voxel (figures 16).

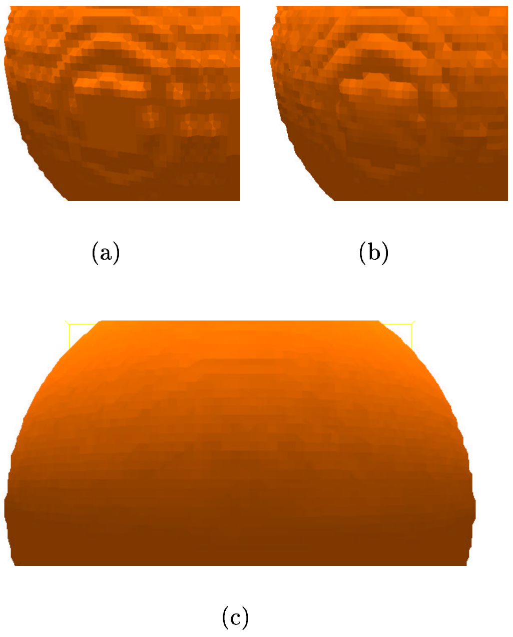


Figure 15: Rendering using voxel normal: in a voxel, all the triangles normals are replaced by the voxel normal. a. with *edges-centers* Marching cubes; b. with pixel-rays; c. with smoothed normal (distance of 5 voxels).

8. CONCLUSION

This paper presents a method for using image pixels together with marching cubes for a shape from silhouette application. Our work relies on a 6-connected discrete surface obtained with the method proposed by Szeliski [18]. We also propose two methods for estimating the normal inside voxels, using either triangular mesh or discrete surface. As seen in the results, our method proves robust even for a cheap acquisition system with usual camera and turntable.

In the future, we aim at combining the reconstructed information with image-based techniques such as ligh-fields/lumigraphs [10, 6] for integrating (real) objects into virtual environments with realistic relighting and global illumination. This method has already been used for estimating light sources positions from images [14].

9. ACKNOWLEDGMENTS

We would like to pay a tribute to Alain Fournier (Imager Lab, University of British Columbia, Vancouver) with whom the overall framework around this paper has been initiated in 1999. The quad has been designed by Stewart Cowley who kindly accepted to let us use this model for our work.

10. REFERENCES

[1] P. Callet. Rendering of binary alloys. In *ICCVG 2004*, sep 2004.
[2] Q. Chen and G. Medioni. A volumetric stereo matching method: Application to image-based

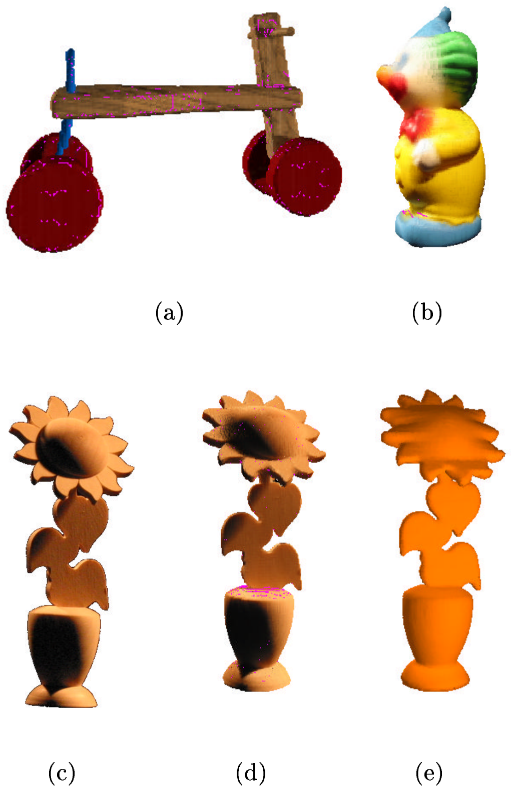


Figure 16: Triangle color corresponds to the average radiance for each voxel; a. for a virtual quad; b. for a real clown (toy); c. Actual photograph; d. for a real wood-flower; e. wood-flower with modified lighting.

modeling. In *CVPR*, pages 29–34. IEEE Computer Society, 1999.

[3] C H Chien and J K Aggarwal. Volume/surface octrees for the representation of three-dimensional objects. *Comput. Vision Graph. Image Process.*, 36(1):100–113, 1986.

[4] C. Hernández Esteban and F. Schmitt. Silhouette and stereo fusion for 3d object modeling. In *3DIM 2003*, pages 46–53, 2003.

[5] Olivier Faugeras and Renaud Keriven. Complete dense stereovision using level set methods. *Lecture Notes in Computer Science*, 1406:379+, 1998.

[6] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *ACM Computer Graphics*, 30(Annual Conference Series):43–54, August 1996.

[7] Jean-Marc Hasenfratz, Marc Lapierre, Jean-Dominique Gascuel, and Edmond Boyer. Real-time capture, reconstruction and insertion into virtual world of human actors. In *Vision, Video and Graphics*, pages 49–56. Eurographics, Elsevier, 2003.

[8] D. R. Hougen and N. Ahuja. Adaptive polynomial modelling of the reflectance map for shape estimation from stereo and shading. In *CVPR*, pages 991–994, 1994.

[9] Kiriakos N. Kutulakos and Steven M. Seitz. A theory of shape by space carving. Technical Report TR692, , 1998.

[10] Marc Levoy and Pat Hanrahan. Lightfield rendering. *Computer Graphics*, 30(Annual Conference Series):31–42, August 1996.

[11] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3D scanning of large statues. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 131–144. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[12] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM Computer Graphics*, 21(Annual Conference Series):163–169, July 1987.

[13] W. N. Martin and J. K. Aggarwal. Volumetric descriptions of objects from multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):150–158, March 1983.

[14] B. Mercier and D. Meneveaux. Joint estimation of multiple light sources and reflectance from images. In *ICCVG 2004*, sep 2004.

[15] S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring, 1997.

[16] Hemant Singh and Rama Chellappa. An improved shape from shading algorithm. Technical Report CS-TR-3218, Department of Computer Science, University of Maryland Center for Automation Research, College Park, MD, February 1994.

[17] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafer. A survey of methods for volumetric scene reconstruction from photographs. In *VG01*, pages 81–100, 2001.

[18] Richard Szeliski. Rapid octree construction from image sequences. In *CVGIP: Image Understanding*, volume 1, pages 23–32, July 1993.

Texture Painting from Video

Jinhui Hu
University of Southern California
3737 Watt Way, PHE 404
USA(90089), Los Angeles, California
jinhuihu@graphics.usc.edu

Suya You
University of Southern California
3737 Watt Way, PHE 404
USA(90089), Los Angeles, California
suyay@graphics.usc.edu

Ulrich Neumann
University of Southern California
3737 Watt Way, PHE 404
USA(90089), Los Angeles, California
uneumann@graphics.usc.edu

ABSTRACT

Texture mapping is an important research topic in computer graphics. Traditional static texture-maps are limiting for capturing a dynamic and up-to-date picture of the environment. This paper presents a new technique called texture painting from video. By employing live video as the texture resource, we are not only able to create an accurate and photo-realistic rendering of the scene, but also can support dynamic spatio-temporal update in the structure of texture model, database, and rendering system. We present our approaches towards the system requirements and experimental results for both simulation and real datasets.

Keywords

Texture Painting, Image Warping, Image Registration

1. INTRODUCTION

Texture is a crucial element in today's graphics oriented applications. In many cases, the value of the applications is increased if both the geometric information and the appearance of the generated images are accurate and realistic analogues of the real world. Texture mapping is a relatively efficient means to create the appearance of realism without the tedium of modeling and rendering every 3-D detail of a surface.

Effective generation of textures has been becoming an important research issue in computer graphics and image processing. There are many ways to acquire data for creating scene textures, such as texture synthesis and direct texture mapping from imagery. Much research has been conducted on the texture synthesis, a technology inspired by research in texture analysis and statistics. While the results from the stochastic textures are useful, this class of approach is unable to deal well with more complicated textures and hard to achieve photo-realistic effects.

To create an accurate and realistic appearance of rendering scene, real world images are often captured and used for texture creation. While most graphics systems support high-quality texture mapping from real imagery, they are limited to static texture resources that must be created prior to use. Static textures are usually derived from fixed cameras at known or computed transformations relative to the modeled objects. The creation and management of such texture databases is also time consuming since it includes image capture and the creation of mapping functions for each segmented image and model patch Once their relationships are established, the texture images are mapped to the geometric models during scene rendering. Therefore, such static texture-maps are limited for applications requiring a dynamic and up-to-date picture of the environment.

This paper presents a novel technique, called *texture painting from video*, to cope with the aforementioned limitations of the static texture mapping and visualizations. Live video is used as texture resource and mapped dynamically onto the 3D models of scenes to reflect the most recent changes of the environments. The video streams can be acquired from stationary or moving cameras such as handheld camcorder, and their projections onto the 3D model are achieved in real-time. Unlike the traditional texture mapping in which each texture image is a *priori* associated with, and mapped onto, patches of the geometric models, our approach dynamically creates the associations between the model and image as a result of image projection during the rendering process. In this case, we can automate the texture mapping process. As new images arrive, or as the scene changes, we simply update the camera pose and image information, rather than repeat the time consuming process of finding mapping transformations, hence make it possible to handle live video streams in real-time.

Related Work

Texture synthesis is a popular technique for texture creation. Such an approach is able to take a sample texture and generate an amount of images, while not exactly like the original, will be perceived by human beings to be the same texture. The parametric model based approach uses a number of parameters to describe a variety of textures ([Hee95, Por00]). The non-parameterized texture, or example based methods generate textures by directly copying pixels from input textures [Efr99]. Recently, [Ash01] suggested an approach to synthesize textures using whole patches of input images. While the texture synthesis technique has been demonstrated successfully in certain applications to be a useful tool for texture generation, the results of the synthesized textures are not photo-realistic and lack of texture details.

Texture painting is an alternative way to create textures. A number of interactive texture painting systems have been suggested. [Iga01] presented a 3D painting system that allows users to directly paint texture images on a 3D model without predefined UV-mapping. [Ber94] employed the Haar wavelet decomposition of image for multi-resolution texture painting. [Per95] painted multi-scale procedural textures on 3D models. Most of current texture painting systems are interactive, allowing users to easily design and edit textures to achieve desired effects. The results can be aesthetically pleasing, but it is hard to make them photo-realistic.

A straightforward way to produce realistic textures is to use real world images as texture resources. To create a complete texture map covering the entire scene being textured, multiple images are used. [Ber01] used high resolution images captured from multiple viewpoints to create high quality textures. [Roc99] stitched and blended multiple textures for creation of textures. [Ofe97] suggested a quadtree approach to represent multi-resolution textures extracted from image sequences. This method requires user to mark the texture area to be extracted, and then manually track the area for a short image sequence (less than 16 images).

Recently several works suggested using video clips as texture resources. [Sch00] proposed the idea of "video textures". From the input clip of limited length, they can generate an infinitely long image sequence by rearranging and blending the original sequence. [Soa01] presented the idea of dynamic textures that are sequences of images with a certain stationary property in time. By learning a model from the input sequence, they synthesize new dynamic textures. Both above methods use the textures in a non-traditional way to achieve desired effects and goals of applications. Their systems deal only with the special textures of repeated patterns, such as sea waves, smoke plumes, etc. In our approach, however, we are dealing with general image textures, i.e. a multidimensional image that is mapped to a multidimensional space. Rather than synthesize new textures, we directly use the original video captured from any real world scene to produce an accurate and realistic appearance of the environment.

2. TEXTURE PAINTING FROM VIDEO

Using live video as the texture resource can offer us many benefits to reproduce real scene. However, we also encounter several technical barriers needed to be overcome. First, the video streams need to be acquired continually and updated, which may lead to infinite texture storage. A straightforward approach of using certain amount of texture memory is unable to keep old texture data from where the projection was a few moments ago. Such texture retention requires approaches being able to persist in the projection for each new video frame onto a surface area.

Second, since we want to paint the dynamic videos onto the surface of 3D model. Only if the camera positions and orientations are known, these data can be projected onto the scene model correctly, thereby highly precise tracking of camera pose and alignments between image frames are required.

Third, due to lack of the models for dynamic objects, those foreground moving objects need to be segmented from the input video to persist only the background textures being projected onto the surfaces of scenes.

Proposed Approach

We present our approaches towards the system requirements essential to the texture painting from video. We propose novel methodologies for rapid creation of dynamic textures from live video streams and their data retention, storage management, and texture refinement. We also implement a prototype of real-time 3D video painting system based on the methodologies we proposed.

Figure 1 illustrates the main structure of our texture painting from video system. As stated above, the main challenge of using live video as texture-maps is how to effectively handle the infinite video streams within a certain amount of texture buffers so that the rendering algorithm can persist in the texture projection for each new frame onto a surface area. Given the fact that only limited scene geometry can be visible from a viewpoint, we propose the idea of "base texture buffer", which is a texture buffer associated with a model patch or a group of neighboring patches being visible from the viewpoint (Figure 2). The base texture buffer is first initialized as white texture, and then dynamically updated with the new coming frames.

To update the base texture buffer for each group of visible patches, we transform each new frame to the base texture buffer. Let the projection matrix associated with the base texture buffer is P and the projection matrix of the new coming frame at a viewpoint is P_v. For every new frame, the transformation P_t between the new frame and the base texture buffer is

$$P_t = P * P_v^{-1} \qquad (1)$$

By using equation (1), we are about to dynamically warp every new frame from different viewpoint to the common base texture buffer, and project the updated content onto the visible surface of scene. In this case, we overcome the problems of infinite texture storage and also the time consuming process of polygon clipping for every video frame.

To achieve accurate image alignment, the 3D model is generated and refined based on LiDAR data [You03], and we recover the camera pose using a robust tracking approach proposed in [Neu03]. Then we refine the recovered alignments in 2D image domain to achieve seamless texture images. Several other core steps, including selective texture painting, base buffer selection, and occlusion detection, are also suggested and will be detailed in following sections.

3. APPROACH DETAILS

3.1 Model Based Image Warping

A key part of texture painting from video is to dynamically update the base texture buffer, which is based on a process of model based image warping. First, we select a base texture buffer. The pose of the base buffer relative to the 3D model of the scene is computed. The correspondence of each pixel in the

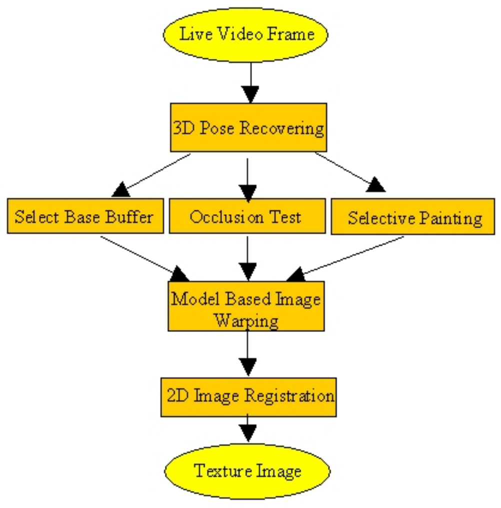


Figure 1 – Overview of the texture painting from video system.

base buffer to the 3D model is computed using equation (2), where I_b denotes the pixel in the base buffer, P_b is the projection matrix of the base buffer, and M is the corresponding point on 3D model. Next, when a new video frame comes, if the model correspondence to the base buffer is visible from current camera position, we update the base buffer with current new frame. This is done by a model based image warping operation.

$$M = I_b P_b^{-1} \qquad (2)$$

$$I_v = P_v M \qquad (3)$$

$$I_b = I_v \qquad (4)$$

As indicated in Figure 2, for each pixel I_b in the base texture, we can find its correspondence M on the 3D model. Given the tracked camera pose and the projection matrix denoted P_v, we project the 3D point M back to the image plane to find its corresponding pixel I_v using the Equation 3. We

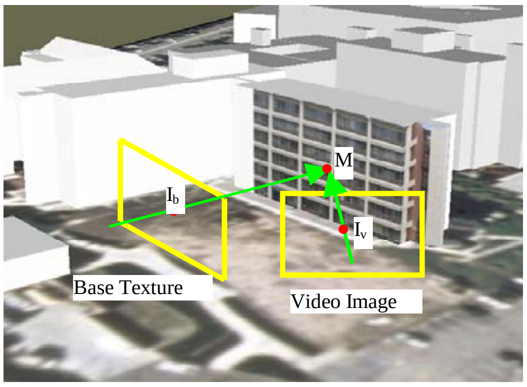


Figure 2 - Model based image warping.

then update its color information in the base texture buffer using Equation 4. This warping process is repeated for every pixel contained in the base texture buffer.

The 3D model based approach is flexible, allowing the camera moving freely in any 3D environment. It requires, however, highly precise camera tracking, which is usually hard to achieve, especially in an outdoor environment. We compensate the tracking errors by employing a 2D image registration approach in Section 3.5. Figure 3 illustrates the result of the model based warping approach.

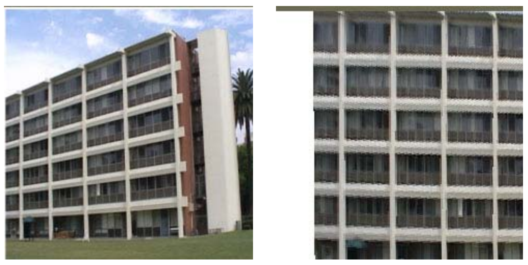


Figure 3 - Result of the model based image warping. Left: original image, right: image warped to base buffer.

3.2 Improve Warped Image Quality

Direct back-warping of the image to the base buffer may result in aliasing. To improve the final texture-maps quality, we use bilinear interpolation for anti-aliasing. As indicated in Figure 4, each pixel coordinates of *I(xb,yb)* in the base texture buffer is treated as a real number. Using the Equations (2-4), we can find its corresponding pixel in video frame, the coordinates of which, *I(xv,yv)*, are also real numbers. Usually, *I(xv, yv)* will not fall into an integer grid in the image. We then interpolate the color information using a four-neighboring bilinear interpolation. Figure 4 shows the result of applying the approach to Figure 3,which apparently improved the image quality.

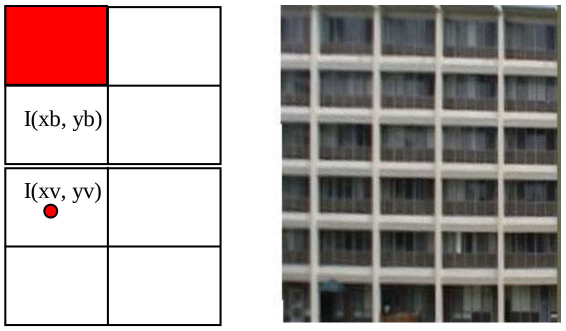


Figure 4 - Using bilinear interpolation to improve warped image quality.

3.3 Occlusion Detection

Under some circumstances, although the 3D model is visible from the base buffer, it may be occluded from current camera viewpoint. The model based warping and texturing will project part of the frame onto the occluded model areas (Figure 5 left). This occlusion problem is solved using depth maps. From each camera viewpoint we render the 3D model to obtain an estimated depth map. When doing the image warping, we first find the corresponding 3D point for an image point being warped. We then project it back to the image plane, and compare its depth value with the estimated depth map. Finally, we keep the pixel projection only if its depth value is less than the corresponding depth value in the depth map. The result is shown in Figure 5.

3.4 Selective Texture Painting

Texture painting from video has the advantage of dynamic texture updating to capture the most recent environment changes. However, if we don't select the content to be textured, it will project everything in the video sequence onto the 3D environment, and consequently give an undesired result. Figure 6 (left) shows a case of moving objects are painted onto the 3D model as part of background textures.

We can view that there are three types of texture information in a video sequence: background textures, foreground textures such as trees, and dynamic objects such as moving people or vehicles. The background textures are the only part we are interested in and intending to process, since they have corresponding 3D model to be textured.

By employing a background learning approach, we segment and remove the undesired objects from the input video. Given a number of training frames, we first learn the images to estimate a background model based on the linear average model. General average method will only work for static cameras. To deal with moving cameras, we first warp each new frame

$$I_{back} = \frac{1}{N}\sum_{i=1}^{N} I_{v,i} \qquad (5)$$

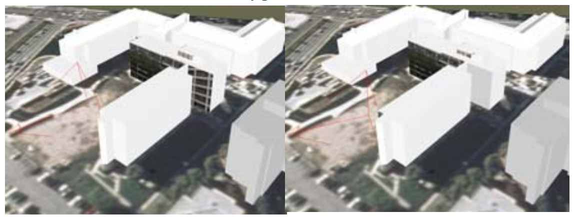


Figure 5 - Occlusion processing. Left: texture painting before depth test. Right: after depth test.

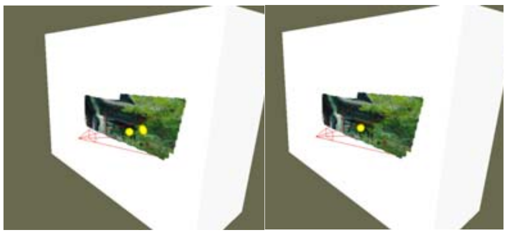


Figure 6 - Selective texture painting. Left: texture painting without background learning. Notice that the moving yellow sphere is painted as part of background texture. Right: after background learning. Only the static scene is painted as background texture.

onto the base texture buffer (which is static relative to the 3D model), then we average each warped frame over the base texture buffer. After that, we segment the objects from the background using image subtraction approach. Figure 6 (right) shows the result after the background learning.

3.5 Refine Texture Alignment

As mentioned above, inaccurate camera pose tracking will result in misalignments between the textured images as shown in Figure 7 (left). The alignment needs to be refined to improve the visualization value. Our approach to this problem is to perform the refinement in 2D image domain, i.e. we first register the video frames based on the camera tracking data and 3D model, and then re-align the registered image with the base buffer using a 2D image registration approach.

- *Motion Model*

Affine model is used for estimating the transformation parameters between the warped video frame and the base texture buffer.

$$\begin{bmatrix} I_{xb} \\ I_{yb} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} I_{xv} \\ I_{yv} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (6)$$

- *Feature Matching*

Corner feature of scenes is used as matching primitive to find the correspondences between the video frame and the base texture buffer. The Harris algorithm is employed to detect image corners, and the SSD (Sum of Squared Difference) approach is used for feature matching. Since the images have been aligned based on the camera tracking data, the SSD matching space is greatly reduced.

- *Parameter Optimization*

While three pairs of correspondences are sufficient to compute a unique solution for affine transformation, we use all matching corners to guarantee accuracy. Least square approach is used to estimate the optimal parameters.

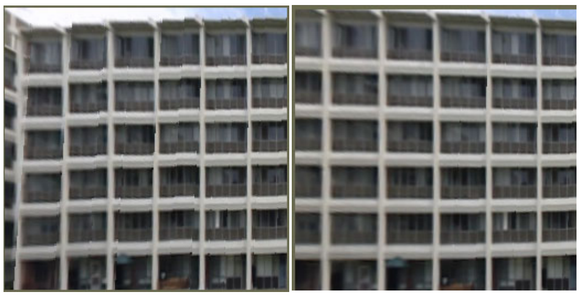


Figure 7 – Refining texture alignment. Left: texture painting using only 3D model based warping, which results in significant misalignment due to inaccurate camera pose. Right: texture painting after refining the texture alignment.

Once the affine parameters are estimated, the 3D-warped image is warped again using the affine transformation to the base texture buffer. Figure 7 (right) shows the refined result of using the approach.

3.6 Select Optimal Base Buffer

Given a 3D scene model, the selection of the base texture buffer is an optimization problem. A straightforward way is to assign one base buffer for each polygon surface. The advantage of this simple method is that it is not required to clip polygon. However, when the number of polygons becomes large, this method becomes unfeasible. So we need to minimize the number of required base texture buffers. One possible approach is to combine as many nearby polygons as possible into one base buffer. However, complex models typically have very varying surface normal, even nearby polygons. Having those polygons with very different normal shared one base buffer will lead to very poor texture reproduction. Another issue arising from the different polygons sharing one base buffer is that we need to clip the 3D model during texture mapping process. The number of clipped polygons will affect the rendering speed, so we would like also to minimize the number of clipped polygons.

The problem of selecting optimal base buffers can be proved to be a NP optimization problem. [Mat99] used a heuristic algorithm to solve the problem. In this paper, we approach to combine neighboring polygons with similar normal into the same base buffer. Our ongoing work is deeply emphasizing this problem.

4. EXPERIMENTAL RESULTS

We have tested the proposed video painting approach on both simulation and real datasets. Figure 8 shows the results from our simulation experiment. A 3D cube model covered by a real image is used to mimic a 3D environment (Figure 8a), and a synthetic camera (shown with a red frustum in Figure 8a) moving freely within the environment is used to "capture" the scene. The image captured from a viewpoint by the camera is shown in the Figure 8b, which is used to simulate input video. We then applied the proposed approach to this scenario. Figure 8e is the input image warping to the base buffer, and Figure 8d shows the images painting onto the 3D model. Since the synthetic camera pose is perfect in this simulation experiment, no further 2D image registration is needed. The whole system is in real time, achieving ~30fps on a 1.1GHZ DELL workstation. It is worth to mention that the camera motion is completely arbitrary, and the captured images are painted persistently onto the correct surface areas, as shown in Figure 8c, and Figure 8f.

We also tested the approach on real data captured by a user walking around USC campus. A portable tracking and video system was used to collect camera tracking data and video streams. The entire campus model was reconstructed using a LiDAR modeling system [You03]. We then applied the approach to the collected dataset. The video frames were first aligned to a temporary buffer based on the tracked camera pose; then the 2D affine transformation between the warped image and base frame was computed to refine the alignment; and finally the warped image is re-warped back to the base buffer using the computed 2D transformation. The whole processing for this scenario achieved ~10fps, in which the most time-consuming part is the 2D image registration. We notice, however, the fact that most part of two successive image frames is overlapped, and there is no need to update the painting for every frame. So, it is sufficient to only update the painting buffer in every certain frame (e.g. every 20th frame) to speedup the system performance. Figure 9 shows the results of the real data experiment.

5. CONCLUSION

Texture is a crucial element in today's graphics oriented applications. Traditional static texture-maps are limiting for capturing a dynamic and up-to-date picture of the environment. This paper presents a new technique of texture painting form video. By employing live video as texture resource, we are not only able to create an accurate and photo-realistic appearance of the rendering scene, but also can support dynamic spatio-temporal update in the structure of texture model, database, and rendering system. We present our approach towards the system requirements and experimental results for both simulation and real datasets.

While the proposed approach is novel, there are still several technical barriers we are addressing in our ongoing work, including

- Selection of the optimal base texture buffer for complex scene models. Currently we simply approach to combine the neighboring polygons with similar normal into the same base buffer, which works fine for most of the man-made scenes' models. We would also like to deeply explore the solution for more complex scenes.
- Real time implementation to support multiple video streams. Today's computing and graphics hardware has reached a stage where many complex real-time computations could be performed with the high-end graphics processors (GPU). How to effectively utilize the programmable GPU features to speedup the video processing is also our emphasis.

ACKNOWLEDGEMENT

This work was supported by the National Geospatial Intelligence Agency (NGA) under a NGA University Research Initiative (NURI) program, and in part by a Multidisciplinary University Research Initiative (MRUI). We thank the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, for their support and facilities. The first author would also like to thank Pamela Fox for her proof reading and comments.

REFERENCES

[Ash01] Ashikhmin, M. Synthesizing natural textures. ACM Symposium on Interactive 3D Graphics, pp. 217–226, 2001.

[Ber94] Berman, D.F. etc. Multiresolution painting and compositing. Proceedings of SIGGRPAH 94, 1994.

[Ber01] Bernardini, F., Martin, I. M. and Rushmeier, H. Hight-quality texture reconstruction from multiple scans. IEEE Visualization and Computer Graphics, Volume: 7, Issue: 4 pp. 318-332,2001.

[Efr99] Efros, A.,and Leung, T. Texture synthesis by non-parametric sampling. ICCV, pp. 1033-1038, 1999.

[Efr01] Efros, A., and Freeman W. T. Image quilting for texture synthesis and transfer. Proceedings of SIGGRAPH 2001, pp. 341–346,2001.

[Hee95] Heeger, D.J. and Bergen, J.R. Pyramid based texture analysis/synthesis. Proceedings of SIGGRPAH 95, pp.229-238, 1995.

[Iga01] Igarashi, T., and Cosgrove, D. adaptive unwrapping for interactive texture painting. ACM Symposium on Interactive 3D Graphics. 2001.

[Jia01] Jiang B., Neumann U., Extendible Tracking by Line Auto-Calibration. International Symposium on Augmented Reality, pp.97-103, New York, October 2001.

[Mat99] Matsushita K., and Kaneko, T. Efficient and handy texture mapping on 3d surfaces. In Proc. of Eurographics, pp.349-358, 1999.

[Neu03]Neumann, U., You, S., Hu, J., Jiang, B. and Lee, J. W. Augmented virtual environments (AVE): dynamic fusion of imagery and 3D models, IEEE Virtual Reality, pp. 61-67, Los Angeles California, 2003.

[Ofe97] Ofek, E. etc. Multiresolution textures from image sequences. IEEE Computer Graphics and Applications, Volume:17, Issue:2 pp.18-29, 1997.

[Per95] Perlin, K. Live paint: painting with procedural multiscale textues. Proceedings of SIGGRAPH, pp.153–160, 1995.

[Por00] Portilla, J., and simoncelli, E.P. A parametric texture model based on joint statistics of complex wavelet coefficients. IJCV 40, 1(Oct.) pp. 49-70, 2000.

[Roc99] Rocchini, C., Cignoni, P. and Montani, C. Multiple textures stitching and blending on 3D objects. In Eurographics Rendering Workshop, 1999.

[Sch00]Schodl, A., Szeliski, R., Salesin, D. H., and Essa, I. Video textures. Proceedings of SIGGRAPH, pp. 489–498, 2000.

[Soa01] Soatto, S., Doretto, G., and Wu, Y. Dynamic textures. In Proceeding of IEEE International Conference on Computer Vision, II, pp. 439–446, 2001.

[Wei00] Wei, L. Y., and Levoy, M. Fast texture synthesis using tree structured vector quantization. Proceedings of SIGGRAPH, pp.479–488, 2000.

[You03]You, S., Hu, J., Neumann, U. and Fox P. Urban Site Modeling From LiDAR, Second International Workshop on Computer Graphics and Geometric Modeling, Montreal, CANADA, 2003.

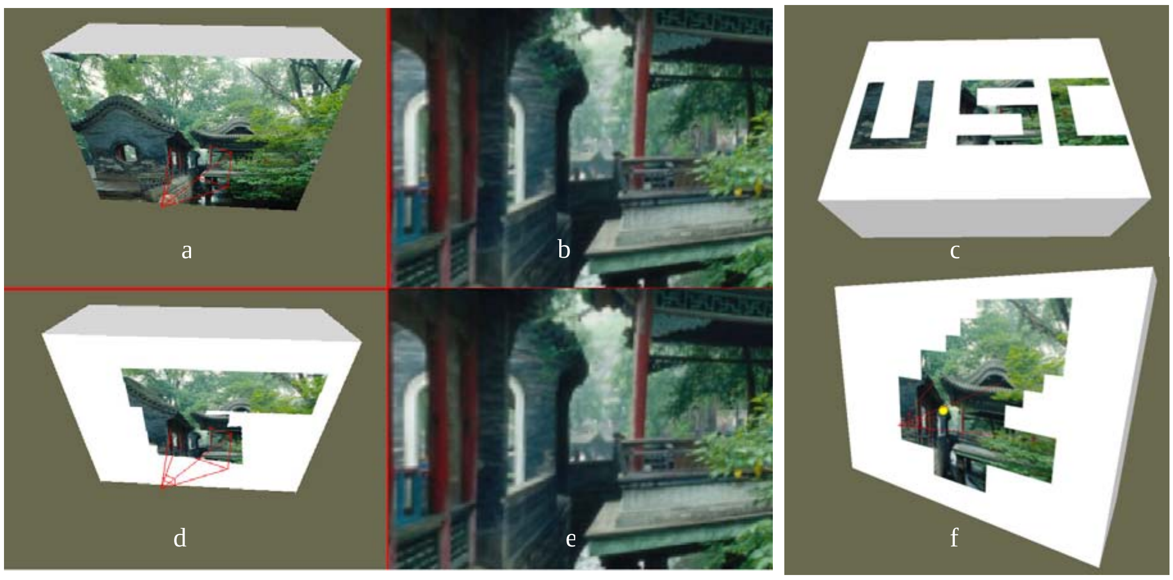


Figure 8 - Texture painting with simulation dataset. A cube model covered by a real image is used as a simulated environment (a). A synthetic camera moving inside the environment is shown in red frustum. The image viewed from the camera's viewpoint, which is used as simulation of input video (b). The video frame is warped based on 3D model onto the base buffer (e), then painted as texture onto the same 3D model (d). The (c), (f) show the painting results of simulated data.

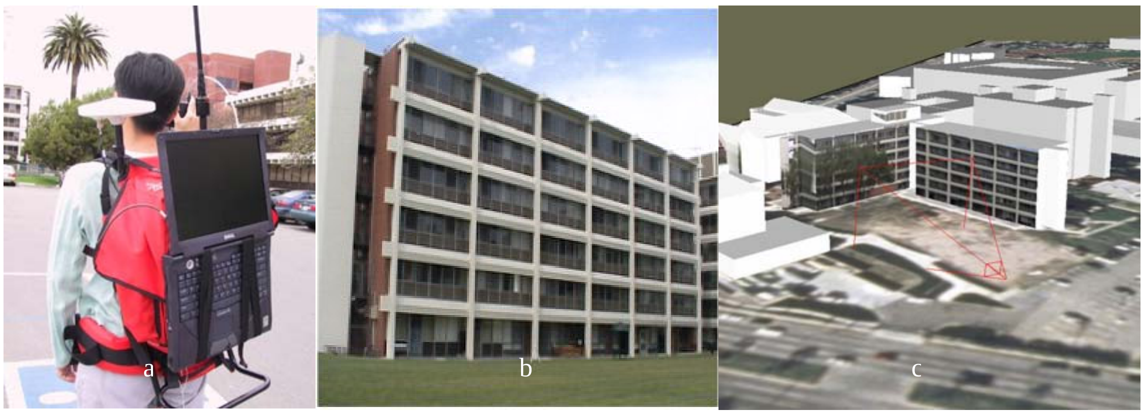


Figure 9 - Texture painting with real data. (a): Portable video acquisition and tracking system. (b): One frame of the captured video. (c): Rend [illegible] of USC campus with live video painting.