ISSN 1213-6972 *Volume 22, Number 1, 2014*

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, visual human interaction and virtual reality, animation, multimedia systems and applications in parallel, distributed and mobile environment.

EDITOR – IN – CHIEF

Václav Skala

Vaclav Skala – Union Agency

Journal of WSCG

Editor-in-Chief: Vaclav Skala
c/o University of West Bohemia
Faculty of Applied Sciences
Univerzitni 8
CZ 306 14 Plzen
Czech Republic
http://www.VaclavSkala.eu

Managing Editor: Vaclav Skala

Printed and Published by:
Vaclav Skala - Union Agency
Na Mazinach 9
CZ 322 00 Plzen
Czech Republic

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

Journal of WSCG

Editor-in-Chief

Vaclav Skala

c/o University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering
Univerzitni 8
CZ 306 14 Plzen
Czech Republic

http://www.VaclavSkala.eu
Journal of WSCG URLs: http://www.wscg.eu or http://wscg.zcu.cz/jwscg

WSCG 2014

Board of Reviewers

Muller,Heinrich
Munoz,Adolfo
Murtagh,Fionn
Okabe,Makoto
Oliveira,Joao
Oliveira,Manuel M.
Oyarzun,Cristina Laura
Pan,Rongjiang
Papaioannou,Georgios
Paquette,Eric
Pasko,Galina
Patane,Giuseppe
Patow,Gustavo
Pedrini,Helio
Pereira,Joao Madeiras
Peters,Jorg
Pina,Jose Luis
Platis,Nikos
Post,Frits,H.
Puig,Anna
Puppo,Enrico
Puppo,Enrico
Rafferty,Karen
Raffin,Romain
Renaud,Christophe
Reshetouski,Ilya
Reshetov,Alexander
Ribardiere,Mickael
Ribeiro,Roberto
Richardson,John
Ritter,Marcel
Rojas-Sola,Jose Ignacio
Rokita,Przemyslaw
Rudomin,Isaac
Sacco,Marco
Sadlo,Filip
Salvetti,Ovidio
Sanna,Andrea
Santos,Luis Paulo
Sapidis,Nickolas,S.
Savchenko,Vladimir
Segura,Rafael
Seipel,Stefan
Sellent,Anita
Semwal,Sudhanshu
Sheng,Bin
Sheng,Yu
Shesh,Amit
Schmidt,Johanna
Sik-Lanyi,Cecilia
Sintorn,Erik
Sirakov,Nikolay Metodiev
Sourin,Alexei
Sousa,A.Augusto
Sramek,Milos
Stroud,Ian
Subsol,Gerard
Sundstedt,Veronica
Svoboda,Tomas
Szecsi,Laszlo
Tang,Min
Teschner,Matthias
Theussl,Thomas
Tian,Feng
Tokuta,Alade
Torrens,Francisco
Trapp,Matthias
Tytkowski,Krzysztof
Umlauf,Georg
Vergeest,Joris
Vitulano,Domenico
Vosinakis,Spyros
Walczak,Krzysztof
Wan,Liang
Wu,Shin-Ting
Wuensche,Burkhard,C.
Wuethrich,Charles
Xin,Shi-Qing
Xu,Fei Dai Dongrong
Yoshizawa,Shin
Yue,Yonghao
Yue,Yonghao
Zalik,Borut
Zemcik,Pavel
Zhang,Xinyu
Zhao,Qiang
Zheng,Youyi
Zitova,Barbara
Zwettler,Gerald

Journal of WSCG Vol. 22, 2014

No.1

Contents

City Sketching

James Gain
University of Cape Town
jgain@cs.uct.ac.za

Patrick Marais
University of Cape Town
patrick@cs.uct.ac.za

Rudolph Neeser
University of Cape Town
rudy.neeser@gmail.com

ABSTRACT

Procedural methods offer an automated means of generating complex cityscapes, incorporating the placement of park areas and the layout of roads, plots and buildings. Unfortunately, existing interfaces to procedural city systems tend to either focus on a single aspect of city layout (such as the road network) ignoring interaction with other elements (such as building dimensions) or expect numeric input with little visual feedback, short of the completed city, which may take up to several minutes to generate.

In this paper we present an interface to procedural city generation, which, through a combination of sketching and gestural input, enables users to specify different land usage (parkland, commercial, residential and industrial), and control the geometric attributes of roads, plots and buildings. Importantly, the inter-relationship of these elements is pre-visualized so that their impact on the final city layout can be predicted. Once generated, further editing, for instance shaping the city skyline or redrawing individual roads, is supported. In general, City Sketching provides a powerful and intuitive interface for designing complex urban layouts.

Keywords
sketching interfaces, procedural modeling

1 INTRODUCTION

Procedural methods for simulating terrain, buildings, plants and cities have been used effectively in computer games, visual effects and virtual environments for training and simulation. This loose family of computational methods includes L-systems [1], noise functions [2], shape grammars [3] and other algorithms characterised by recursive self-affine behaviour. Their benefit is that with little manual effort, complex and realistic content can be generated. In the most extreme instance an entire virtual world can burgeon from a single pseudo-random seed.

However, there is a tension between the productivity that arises from extensive automation and the control provided by user involvement. Ideally, users should be able to dictate key aspects of a scene with as fine a granularity as desired, and the attendant procedural method should follow these specifications. There are several strategies for achieving this, some more successful than others.

The most prosaic procedural interface is a set of numeric control parameters. Unfortunately, this type of interface tends to expose the internals of the procedural method and fails to foster any geometric intuition on the part of the user. More effective are image maps, where regions of a landscape are painted with different properties, such as landforms with particular frequencies [4], the distribution and density of plants or categories of land usage in cities [5]. Here there is a direct visual mapping to the domain but the context of use is somewhat limited. Another option is a textual approach in which adjectives are used to describe a scene and then mapped via machine learning to procedural parameters. This approach is generally preferred to parameter specification by inexperienced users [6] but it does require an extensive training phase. Recently, procedural techniques have begun to borrow from real-world data using example-based synthesis. As long as such real-world data-sets are readily available, as is the case for city [7] data, and the intended results do not diverge significantly from the exemplars, this can lead to unprecedented realism.

Recent work has resulted in some early direct manipulation of procedural models for buildings and trees, which has been extended to the direct manipulation of road networks [8].

Another alternative is to develop a sketching interface. Inspired by pencil-and-paper illustration, these interfaces are accessible to non-experts and enable rapid iterative design, particularly where absolute precision is not required. They have been applied with success to the procedural modeling of terrain, trees, flowers and clothing.

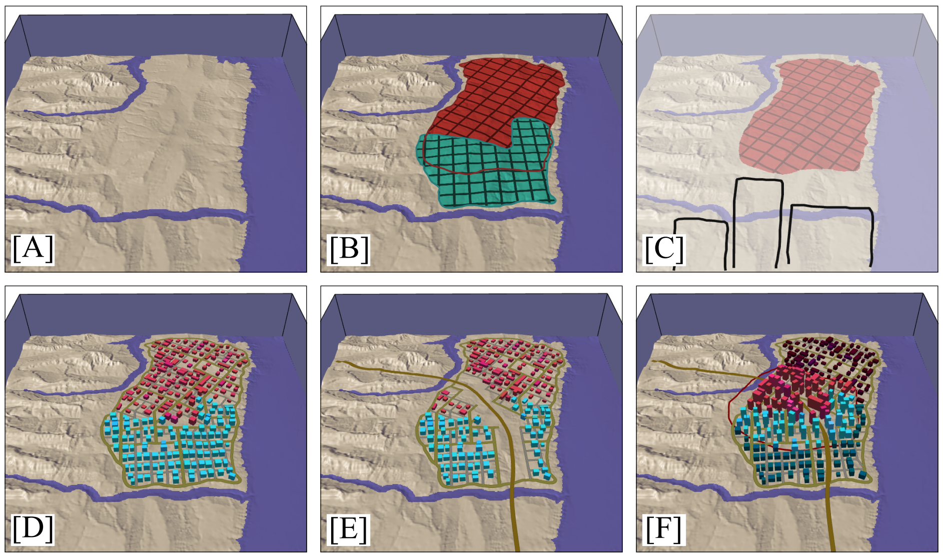


Figure 1: Generating urban layouts by City Sketching: [A] Given an existing landscape; [B] the user sketches road pattern and land usage zones, and [C] a gesture interface is used to interpret the type of zone and its statistics; [D] a corresponding city is generated; further iterative editing of roads [E] and buildings [F] is supported.

Turning to the particular case of procedural cities, the purpose here is to create an urban layout composed of a road network, plot subdivisions and building envelopes. This may extend to procedurally modeling the facades of individual buildings, but, of late, this has devolved to a separate subfield [3]. While seminal work [5] uses broad proxy parameters (road pattern type and population density) coupled to image map inputs, the field has since adopted both example-based methods [7] and simulation [10] in the interest of increased variety and realism. With some exceptions (as noted in Section 2) interfaces have not kept pace with the increasing sophistication of city simulation and are generally still limited to direct parameter specification and image maps.

Motivated by the failings of existing procedural city interfaces and the ease-of-use, familiarity and speed of sketching, we have developed *City Sketching*. The fundamental operation in our system is marking out regions on a landscape (see Figure 1[A, B]) to represent road patterns and land usage zones. There are two important considerations here: firstly, a full set of statistics is automatically calculated for each region (for example, land usage will not only have a type — commercial, industrial, residential or parklands — but also a mean, minimum and maximum for plot size, building base and height, or appropriate equivalents), and, secondly, the final city is determined by the interaction between the different regions (roads conform to the landscape, plot sizes determine separation between roads, building heights influence road widths). We provide visual feedback of these interrelationships using procedural textures projected onto the landscape. Further to the high-level control provided by such region drawing, users are also able to directly sketch individual roads at a low-level.

Once a user is satisfied with their initial layout, the region statistics and explicit road constraints are passed to a procedural system that creates road networks, plot subdivisions and individual building placements (Figure 1[D]). After the city has been created it can be iteratively modified by sketching new roads and regions (Figure 1[E]), followed by a localized procedural update. Furthermore, the characteristic skyline of the city can be shaped by raising and lowering buildings to fit a side-view sketch (Figure 1[F]), or more directly by interactively manipulating building heights in a region.

To summarize, the key contribution of this work is a holistic approach to procedural city generation that allows users to specify and visualize not only road networks, but also categories of land usage, dimensions of plots and buildings and their interrelationships.

The system has a number of novel components: (a) A gesture-based component that extracts both zone information and statistics for roads and buildings from sketched exemplars. (b) Interactive pre-visualization, which shows the interrelationship between road patterns and region usage and their effect on the final city. (c) A procedural engine that grows the entire city and supports interaction between all elements, and the inclusion of user specified roads as constraints.

The remainder of the paper is structured as follows: Section 2 provides more detailed coverage of previous interfaces to procedural city systems; Sections 3 and 4 address the interface design and associated procedural engine; Section 5 examines the system's performance in terms of usability and versatility; and, finally, Section 6 provides a summary and recommendations for future work.

2 RELATED WORK

As previously mentioned, most procedural city systems employ a combination of numeric parameter input and image maps [5, 12, 10]. These image maps are often generated using a raster paint application and supplied as separate input files, making it difficult for a user to create an integrated mental map of the final layout. Some systems do support the drawing of simple road constraints [10] but, in any event, rarely go far enough in allowing interactive specification and previsualization of city layout. This is less problematic for fast system, where the generated city serves as its own visualization, but can cause frustrating design cycles for simulation-heavy methods, which typically take several minutes to execute.

It is also worth considering other procedural domains, such as plant and terrain modelling, be they painting [4] or sketching [9] interfaces. The key issue is how regions are specified in these interfaces. Sketching interfaces generally allow the user to draw a closed loop, which works well for large contiguous regions without holes. In contrast, painting interfaces require longer to demarcate a region but allow complex topology. While these interfaces are a source of inspiration, our system goes beyond marking out regions and visualizes how regions overlap and interact, the specification of constrained linear features such as rivers and roads, and the sketch-based input of geometric region parameters for roads and buildings.

There are two recent techniques that counter this trend. Aliaga et al. [7] develop an example-based approach that allows a city to be assembled from fragments consisting of vector data (attributed point sets representing the junctions of road) and aligned images (georeferenced aerial photography of city blocks). From an interface perspective, fragments can be copied and pasted and then synthesized into a coherent urban layout via *join*, *blend* or *expand* operations depending on their spatial arrangement. Although the technique focuses on mimicking aerial imagery, there is no reason, in principal, that it could not be extended to create a fully geometric realization of a virtual city. The only weakness with this kind of structural synthesis is the reliance on existing annotated and registered data.

We were also inspired by the road network design of Chen et al. [13]. In their system a 2D tensor field is shaped by boundary, pattern, heightfield and density constraints, and edited with smoothing, noise, rotation and brush operations. Finally, roads are traced along the streamlines of the major and minor eigenvector fields. These results are passed to a separate engine for splitting into block, plots and buildings, which precludes any real feedback between building data and the road network, as is achieved by our system.

Lipp et al. [8] show how road networks can be edited using direct manipulation, including copy and paste functionality. These operations preserve road network validity (such as preventing unwanted intersections), and persist even after the road network is regenerated.

Galin et al. [11] address road networks between cities using a hierarchical approach, starting from highways and progressing to secondary roads, with graph optimization that respects terrain and water features. Their interface relies on the now familiar image map painting.

For a more complete overview of approaches to the modelling and simulation of urban environments, see Vanegas et al. [14].

3 INTERFACE

There are two aspects of city design that receive separate consideration in our sketching interface: (a) closed region strokes are drawn onto the landscape to represent *usage zones and road patterns* with a gestural interface for capturing various region statistics; and (b), once an initial city has been generated, further editing is supported, such as drawing in extra roads, constraining skylines from separate viewpoints using sketched polylines, and altering building heights. Distinct forms of sketching are appropriate in each case. Care is taken in the interface to provide support for iterative refinement. For instance, users are able to refine a particular portion of a curve or loop by oversketching or direct manipulation, with the option of shifting to a more favorable viewpoint during the process. Furthermore, users can "undo" previously committed sketches using a scratch gesture. In our system, city and landscape features are generally sketched from an aerial viewpoint with strokes and points in the 2D image plane projected onto the 3D terrain. The resulting features are then variously interpreted as usage zones, road patterns or road constraints, depending on the selected interface tool.

3.1 Usage Zones and Road Patterns

City Sketching, in common with other procedural city systems, controls city attributes, such as zoning and road arrangement, through the shaping and placement of contiguous regions. However, we diverge from previous approaches in several respects. First, our system exposes a broad spectrum of detailed geometric parameters for roads, buildings and plots. We favor a geometric approach because it promotes visual depiction and thus has greater predictability than abstract quantities such as population density and wealth. Second, these parameters are controlled through a combination

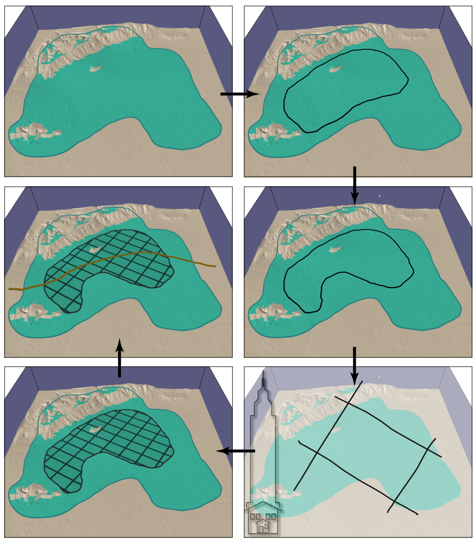


Figure 2: Sketching of zones and roads: inside a previously sketched commercial usage zone (blue-green) the user draws a closed region stroke onto the terrain; this can be modified by oversketching; the road pattern (a grid) and statistics (orientation of major and minor gridlines) are inferred from gestures; the zones and their interactions are visualized using procedural textures; roads (such as a highway) can also be sketched directly.

of sketching (to specify the outlines of regions) and gestures (to capture specific region information) — Figure 2.

City Sketching users mark out two types of regions on the landscape. *Usage Zones* represent typical urban zoning classifications (commercial, industrial, residential or parklands) and also capture the size and density of buildings (or trees in the case of parklands); specifically the minimum, maximum and mean for building height, building length and plot length. *Road Patterns*, on the other hand, represent the dominant layout of roads in a region (grid, radial or random) and also encode various style parameters, such as the angular separation between radiating avenues in the case of the radial style, or the aspect ratio of blocks in the grid style.

The final city layout is determined by the interaction of usage zones and road patterns. Thus, inter-road spacing is determined by plot size, road widths by building volume and density (approximating traffic intensity), and the incidence at road junctions by the road pattern. This enables nuanced control over an urban layout. For example, a rich residential suburb with relatively narrow roads and large single-story buildings on larger plots can be specified with our system — a level of control difficult to achieve using less detailed parameters such as population density.

However, two challenges arise: the need for a visual depiction of the interaction between regions, and for a simple, fast and effective means of entering region statistics. The problem of visualization is overcome by creating procedural textures that are stencilled appropriately to the intersection of a usage zone and road pattern region and mapped onto the landscape. While these textures cannot capture the full complexity of a procedural system, as presented in Section 4.2, they are sufficient to convey the directionality, distribution and style of the final road layout. We use gestures to address the problem of capturing region information. While the field of gestural input is broad and encompasses aspects such as communication through hand motion and body language, one commonality is the compact encoding of symbolic information. In our context a set of sketched symbols is used to indicate the type of each region (see Figure 3). If that was the limit of our gesture functionality then a simple button or menu interface would suffice. However, our gestures also serve as exemplars from which region statistics are derived.

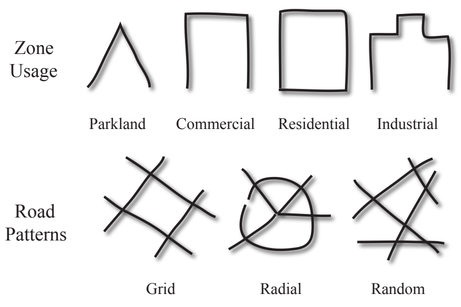


Figure 3: Gestures: simple gestures are used to indicate zone usage (park, residential, commercial, or industrial) or road pattern (grid, radial, or random). Variation in the gestures, such as spacing and aspect ratio are also used to capture zone statistics.

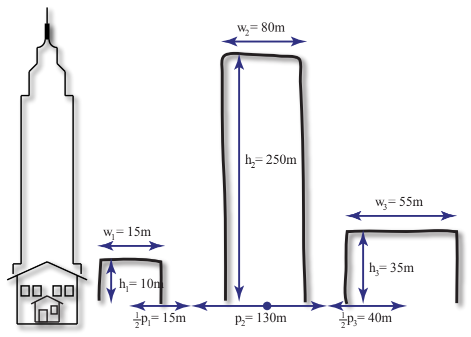


Figure 4: Extracting statistics from gestures: three exemplars are drawn for each land usage zone (in this case commercial) and their aspect ratio and relative separation are used to derive statistics (minimum, maximum and mean) for building height (h_1, h_2, h_3), building width (w_1, w_2, w_3) and plot width (p_1, p_2, p_3). Scaling in the vertical dimension is piecewise linear in order to allow both skyscrapers and single-story buildings on the same gesture canvas.

For usage zones we expect users to draw three gestures of the same type. As shown in Figure 4, the dimensions of the bounding boxes around each gestures and their relative separation are then used to derive statistics for building length, height and plot length. Specifically, the values for a particular parameter (e.g., plot length) are sorted from smallest to largest and then mapped to the minimum, mean and maximum, respectively. One complication is that the full range of building heights (from 5m to 400m) cannot reasonably be drawn to a linear scale on a single canvas. Instead, we utilize a piecewise linear mapping with a relatively small gradient up to 20m and a steeper slope thereafter. The key on the left of Figure 4 serves users as a visual indicator of this scale.

Road pattern layout statistics are derived as follows: for grids, the direction of major/minor axes and the aspect ratio of city blocks; for radial, the angular distribution of the radial spokes and their center; for random, the range of angles between intersecting roads. Of course, the exemplars could be followed more explicitly, but this would require a more deliberate and less natural style of interaction.

Finally, while road pattern gestures provide the template for an entire region, sometimes users wish to precisely control the placement of roads. For this purpose, City Sketching provides the capability to draw primary, secondary or tertiary roads directly onto the terrain, as demonstrated in Figure 2. When the procedural city is generated road constraints, corresponding to the user-drawn roads, are stitched into the prevailing road pattern.

3.2 Building Editing

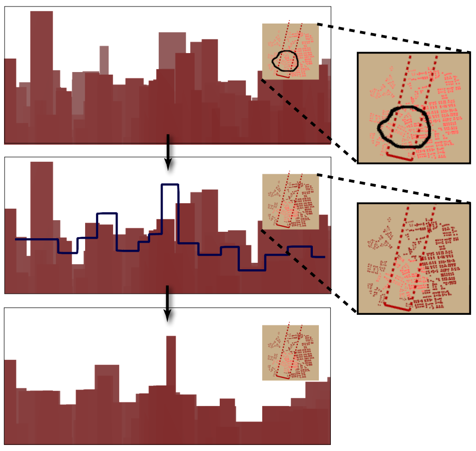


Figure 5: Sketching skylines: from an orthographic side-on view the user selects a subset of buildings on an inset mini-map; then draws a characteristic skyline, which is automatically converted into a piecewise representation; finally, building heights are adjusted to conform to the skyline.

Roads can be drawn directly onto the terrain and then passed as explicit constraints to the procedural engine, and it is desirable to have analogous control over buildings. This is provided in two ways: in *skyline* mode where the characteristic skyline of the city can be sketched from a side-on orthographic view, with building heights conforming to the resulting constraint envelope (see Figure 5), and in *elevation* mode, by demarcating collections of buildings from an aerial perspective and then directly raising or lowering them.

In skyline mode, users first select a group of buildings to edit by drawing an enclosing loop on an inset contextual map; they can then draw and refine a silhouette, which is automatically converted into a step-wise representation (effectively, a piecewise constant function). Once committed, building heights are adjusted to the minimal extent necessary to meet the skyline constraint, while taking cognizance of the region statistics (as discussed further in section 4.3). Note that building footprints remain unaltered, because to do otherwise would require re-organising the street layout.

From a rendering perspective we chose flat-shading, so that users would focus predominantly on building silhouettes, and a fog effect to provide some depth context. We found an orthographic projection to be essential in preserving the relative scale of buildings.

While this approach works well for moderate changes in terrain altitude, it fails for cities built on or around hills and mountains (such as San Francisco). In extreme cases buildings might be entirely culled, since their foundations lie above the sketched skyline. We considered a number of alternatives — projecting buildings onto a plane for the purposes of Skyline editing or allowing users to select buildings in bands of altitude — but ultimately we opted to simply filter out buildings with a base situated above a user-defined threshold. Skyline mode is primarily intended for shaping the city vista from a particular view direction and this is damaged by alternative implementations where there is only an indirect correspondence between the sketched skyline and the final results.

In elevation mode, the user selects a set of buildings either by radial distance from a pick point or by drawing a region loop (using the same process as land usages and road patterns). The selected buildings can then be raised or lowered by a constant amount, with a smooth tailing away of elevation change towards the edges of the selection, so that the alterations blend with the surrounding city (see Figure 1[E]). Any building whose new height falls outside the bounds specified by the region statistics is highlighted.

4 IMPLEMENTATION

4.1 Gesture Recognition

The gesture recognition subsystem is responsible for categorizing one or more user-sketched input strokes and extracting appropriate parameters. For our purposes, each stroke is a sequence of point samples drawn in one continuous motion. Gesture classification is usually based on a set of extracted features, with machine learning applied to deal with inevitable variability. Extensive training on example gestures is usually required and, although the recognition rates are generally high (95% or greater), these systems can also produce non-negligible false positive rates [15].

However, since our interface need only differentiate between a limited set of 4 zone usage gestures, methods based on machine learning were deemed an unnecessary overhead. Instead, we have developed a simple and robust ad-hoc approach with satisfactory performance (see Section 5). Specifically, we assume that gesture strokes are piecewise linear curves defined on the unit square, and that the interface designer will supply a set of *gesture exemplars* for each zone usage type. To determine if a set of point-sampled strokes matches a valid usage exemplar, we proceed as follows: First, we filter the point set to remove drawing jitter. We then apply a simple line aggregation algorithm to collapse multiple consecutive line segments, which results in either a single larger line segment or a poly-line, depending on the curvature of the stroke. Finally, we compute a simple *feature vector*: a list of angles between adjacent line segments. A similar vector is also computed for each exemplar during initialization. Classification entails first matching the number of segments (since each usage zone gesture is unique in this respect) and then testing the absolute difference between components of the feature vector to see if the gesture falls within the broad range of permitted angular variation. If a gesture is not recognized as a usage zone we test it against the set of road patterns.

A road pattern template also consists of a series of strokes arranged in specific geometric relationships. For example, a Manhattan grid has two sets of roughly parallel lines, which are approximately orthogonal, while a Paris-style radial road network has concentric ring-roads with spoke-avenues radiating from a central area.

In order to classify a road pattern, each input stroke is first categorised as a line, polyline or (approximate) circle. We then compare the set of input strokes against each road pattern template in turn, until a match is found. If no match is found, the collection of strokes is flagged as unclassifiable. The tests required for a match depend on the template under consideration. So, for the radial pattern, the stroke set must contain both lines and at least one circle, but no polylines. If multiple circles are present, the circles are tested to see if they are roughly concentric. Finally, lines are examined to see if they start within the innermost circle and cross outwards. At least two lines are required to estimate angular separation. If all these tests are passed, the set of strokes is matched and the appropriate parameters extracted. Otherwise we attempt to match against the next pattern template.

4.2 Procedural City Generation

Our goal is to create a simple and efficient means of generating road networks, parcel allotments and building envelopes, based on the statistical information provided by a city sketch. The system outlined here is just one of several possible procedural back-ends that could be fed by our sketching front-end. Our approach is inspired by the principles of L-system road generation [5], including context awareness and non-determinism. However, in practice, we found L-systems to be very slow for large cities, because they require production matching over the entire string, which grows exponentially. Instead, we use *growth seeds* to represent potential road segments. These are placed initially at the centroids of each unique combination of road pattern and usage zone. Should the centroid not lie inside the corresponding region, we use the center of the largest inset square instead. A growth seed derives its direction, length and road width from the underlying region statistics. For example, road length within a grid pattern depends on average plot width, block aspect ratio, and alignment with the major or minor axis. As a further example, road width varies according to the average building volume normalized by plot area as a proxy for population density. Road growth also adapts to the terrain, either avoiding or conforming to slope contours depending on the severity of the incline. To lessen the impact of high frequency fluctuations we first smooth the terrain before calculating slope. In regions with a slope greater than 2.86 degrees from horizontal, roads are aligned with the terrain contours (the so-called San Francisco pattern [5]) and we prevent road placement altogether where the slope is greater than 5.7 degrees, corresponding to a 10% grade limit used in international road codes.

Growth seeds are reoriented as they approach existing roads, using the heuristics outlined by Parish and Müller [5], in order to connect at junctions and avoid malformed blocks. As a seed lays down roads it may, depending on the needs of the road network, spawn further growth seeds at junctions.

There are two issues with this approach: handling user-drawn road constraints and merging road networks at region boundaries. It is important that individual roads sketched by users do not unduly distort the underlying road pattern. Unfortunately, the obvious strategy of growing patterns outwards from seeds placed along a sketched road is foiled by any form of curvature in the sketch. Instead, we stitch in user-drawn roads by first growing the road network to completion, then deleting existing roads within an offset distance of the user constraint, followed by laying a new road along the constraint curve, and finally initiating growth seeds at the newly formed deadends in the existing network, with parameters copied from incident deleted roads. A memory of the prior road pattern is thus retained, so that it is not unduly disrupted during regrowth. In this way the network regrows with a similar pattern that adapts where necessary by snapping to the road constraint. A similar process is used for filling new regions in an existing city.

The second challenge lies in correctly switching between road patterns at region boundaries. It is not sufficient to reorient seeds as they transition between regions because, even if their directions align with existing roads in the current region, this does nothing to match road position and spacing. Rather, we delay growth seeds as they cross region boundaries. In this way existing growth within a region can continue undistorted and the approaching fronts of two road patterns will meet correctly at the boundary. We also place different categories of road into separate queues and process each to completion in priority order (highways, arterials, then streets), ensuring that higher priority roads can extend to completion without being blocked by those with lower priority.

Once the road network is in place, we extract blocks by a simple winding process. We then generate buildings by first subdividing an input block into parcels and the placing a building bounding box within each parcel. The bounding box is a cuboid which can be replaced with the appropriate building geometry.

The parcel splitting procedure must run quickly and produce sub-plots which span the range of plot sizes specified by the usage zone sketches (see Figure 4). This provides a single axis for building width and spacing (and thus, implicitly, plot size), from which infer a range of values for plot size that constrain the allowable size of bounding boxes generated by parcel splitting. No explicit provision is made to produce or control irregular sub-plots; they arise naturally along curved boundaries. The plot parameters do not obey a specific distribution: they are merely bounds which constrain how large or small a plot should be. Since we employ a recursive splitting procedure, explained below, and this will tend to split plots as much as possible, we allow a random chance (50%) of terminating splitting early, as long as the current plot under scrutiny is not bigger than the largest specified plot size.

Our parcel generation phase is a modification of the method used by Parish and Müller [5]. First, we build an oriented bounding box for the input polygon. We then split the input polygon in the direction of the longest axis of the bounding box, allowing a small random perturbation in the range (0.4,0.6) for the origin of the split line along the shortest box axis. The splitting process continues recursively: each new polygon is split orthogonally to an edge of the polygon which maximizes our splitting criterion. This criterion gives high weight to long edges as well as edges which have immediately adjacent edges which are near orthogonal to the edge under consideration. The origin of the splitting line is jittered to ensure that we do not always split from the centre of an edge. We effectively ignore concavity by taking the first two line intersection points with the polygon loop. This is guaranteed to yield two polygons, regardless of boundary complexity. This allows concave plots if the initial polygons is concave, but since these occur in reality this is not an issue. We disallow splits which would generate new polygons with oriented bounding boxes that fall below our minimum parcel size requirement. A new parcel must be large enough to accommodate the smallest building as well as a prescribed margin around each building. We also disallow splits which would generate parcels with no street access.

For each we estimate the largest interior box, using a sampling approach: we look inwards from each polygon edge and try to find the largest contained box using a number of ray intersections centred on each edge. While this is a rather crude approximation, it is reasonably cheap (for small numbers of rays) and yield boxes oriented along edges, promoting street access. We then shrink the box to fit the constraints supplied by the sketch system. We do this by generating a random plot margin for each box axis and determining whether the resulting (reduced) box obeys the constraints on building size. If it does, we accept the reduced envelope as the building base; otherwise we try with the minimal margin constraint. If this fails, we flag that parcel as being too small to contain a valid building. Once we have generated a suitable building, we impose a global aspect ratio check to shrink plots which have exploited the maximal plot size to aggressively. Currently we do not allow buildings with an aspect ratio above 3. Finally, we generate a uniform random height between the height constraints supplied for that input block.

4.3 Building Editing

The skyline subsystem takes a sketched skyline curve and must then ensure that the final orthogonal projections of all selected buildings conform to the implied envelope. Our solution obeys the following principles: (a) the fewest possible buildings are altered, (b) heights after modification remain in the min/max range attached to buildings, (c) buildings above the skyline constraint are lowered, while those lying below are only raised as necessary to cover the skyline, and (d) the procedure executes at interactive rates.

To enforce the skyline constraint, we decompose the piece-wise constant skyline envelope into a sequence of zones on the projection plane. Each zone is simply a disjoint rectangular region in the plane with a constant height constraint that impacts all buildings projecting into that zone. Of course, buildings often project across adjacent zones. In keeping with principle 3, such a building is lowered, if necessary, to the minimum height of the zones it spans.

In order to satisfy principle 1, we develop the idea of *zone coverage*. This refers to the proportion of a zone's width that is covered by the union of the bases of buildings whose heights have been fitted to that zone's skyline. Thus the potential coverage of a building is the intersection of its projected extent with the zone's extent, but this coverage is only realized if its height is altered to fit the zone. To keep building edits to a minimum, we first raise those buildings which contribute a large proportion of unique coverage to any given zone. To do this we order the height edits for each zone by potential coverage. We then select legal edits for each zone until either we have no edits remaining or some prescribed coverage fraction has been attained. In practice the piece-wise constant height constraint can lead to skylines that are too sharply defined to be natural. We address this by allowing a certain "fuzziness" in associating buildings with zones. More precisely, we shrink the building extent by an ε in either direction for the purpose of computing zone associations. The resulting skyline has a rougher, more approximate appearance while, in essence, still conforming to the sketched constraint. To further enhance variation, we add a small random offset to building height edits, which allows them some variation below the zone height threshold.

By default, the skyline is enforced on all buildings in the selected region. Sometimes, however, a user may wish to edit only those buildings in the front-line closest to the view. We find these buildings using a 1D z-buffer. The base extent of each building (in the XZ-plane) is rasterized into a 1D depth buffer such that the identity of the closest buildings along the base line can be retrieved.

For elevation mode we need to ensure that height changes fall off smoothly towards the boundary of the selection. To achieve this the change in height is scaled by a weighting factor:

$$w = \begin{cases} 1 & \text{if } r > r_1 \\ f(1-\frac{r}{r_1}) & \text{if } 0 \le r \le r_1 \\ 0 & \text{if } r < 0 \end{cases} \quad (1)$$

where $f(x) = (x^2-1)^2$, $x \in [0,1]$; r is the shortest distance to the boundary of the selection (with points inside having positive distance and those outside negative), and r_1 is the distance after which a constant height change is applied.

5 EXPERIMENTS AND RESULTS

Since the primary focus of this work is the development of a sketching and gestural interface, we concentrate in this section on the utility and ease-of-use of the final system. In this regard, we administered a number of usability experiments:

Comparing Sketching and Input Maps: We tested an early version of our sketching interface against the dominant alternative of image map input created with a paint program. In the painting interface each layer (usage, road pattern, parkland) is displayed separately and created using a typical image painting toolset with variable width paintbrushes, erasers and bucket fill. This represents a common paradigm in procedural city modelling.

The experiment consisted of a between-groups study with 20 subjects. Users were asked to reproduce 2 previously created cities (drawn randomly from a sample of 5) and were assessed on the basis of speed, accuracy and a post-test usability questionnaire. While sketching was faster than image maps, this was not significant ($p = 0.15$ on a t-test). We attribute this to the disparity in experience with comparable interfaces (users scored a mean of 3.2 for painting and 1.4 for sketching on a 5-point Liekert scale of self-reported experience). With training we expect sketching to be significantly faster. Accuracy was assessed using a scoring scheme which considered the approximate shape of region boundaries, placement of road constraints and the heights of buildings. The sketching interface was found to be significantly more accurate ($p = 0.008$). Finally, users scored the usability of sketching more highly than image maps, with a mean score of 4.35 against 2.10 on a simple 5-point Liekert scale. In summary, we found that sketching wins in general over image maps with respect to accuracy and usability, with a non-significant trend towards improved speed.

Gesture Calibration: In order to deal with the wide range in building scales (from 5m high single-story homes to 400m tall skyscrapers) and the limited size of the gesture canvas, we originally intended employing an exponential scale. After running an exploratory experiment with 10 subjects, however, we discovered that most users found this to be confusing and counter-intuitive. Given drawings of various building gestures along with a scaling key (as shown on the left of Figure 4) and the task of attaching heights and widths, the majority of users (8 out of 10) used a piecewise linear scale for the vertical, with a change in slope at about the $3-4$ story level, and a width dictated by the aspect ratio of the building. These findings motivated our final approach.

Gesture Recognition: We undertook an experiment to test the robustness of our simple gesture recognition scheme, since it can be frustrating for a user to have to undo false positives or redo false negatives. Our experiment was something of an acid test: the 12 subjects were given no training except a briefing sheet, received no feedback as to the success or failure of gestures (since we wanted to minimize learning effects) and they used a mouse, which, although ubiquitous, is rather deficient as a drawing device. Subjects were presented with 3 examples of each gesture, in a randomized order. The results were encouraging, with a mean recognition rate of 84.1% ($s = 7.4\%$) on the first attempt. There were no misclassified gestures (false positives) and the mean unclassified gesture rate (false negatives) was thus 15.9%. The gesture that failed most often was the simple Manhattan road pattern: two sets of near orthogonal lines that cross each other. Users sometimes drew overly long lines with high curvature, which were incorrectly classified as polylines, an issue that is easily corrected with remedial training. Overall, this experimental setup represents worst-case performance, since, in practice, users are given feedback, are allowed to practise, and are not restricted to using a mouse, leading to substantially higher first-attempt recognition rates.

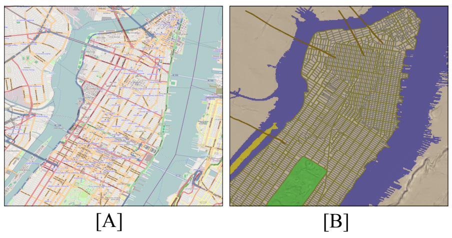


Figure 6: Results: comparison of real and virtual Manhattan - [A] real street map (©OpenStreetMap contributors), [B] virtual recreation. City Sketching is able to reproduce the alignment and spacing of road networks and the placement of features, such as Central Park, Broadway and Brooklyn Bridge but is not suited to exactly reproducing local road layouts.

We have also undertaken some informal validation tests to see whether an existing cityscape can be roughly reproduced by an experienced designer within a fixed time limit (45 minutes). A procedural version of Manhattan (New York City) with 7660 buildings and 3006 roads is shown in Figure 7 and alongside a road network from the corresponding real city in Figure 6. Also shown is a hypothetical city on the edge of the Grand Canyon (11794 buildings and 5605 roads). The procedural city generation completed in under 30 seconds for both examples on a 2.4 GHz Intel Core 2 Duo with 2 Gb RAM.

We have found that sketched gestures are an effective input mechanism when absolute precision is not required and parameters have a direct geometric interpretation. In such circumstances they can be both meaningful and compact. For instance, our usage zone gesture captures a selection and 9 float values in a single sketch. However, gestures are less useful for indirect parameter specification. For example, our system assumes usage zones with a dominant type, but in many cases a mixture is more realistic. Using a gesture to encode the exact proportion of industrial, residential and commercial buildings within a zone would likely be clumsy in the extreme. In cases like this, traditional GUI elements, such as slider bars, are more appropriate.

6 CONCLUSION

This paper presented City Sketching, a procedural system that employs a hybrid sketch and gesture interface, enabling users to control, in detail, the generation of a virtual city, including the layout and interaction between road networks, categories of land usage, and the dimensions of plots and buildings. Our system can be applied to improve scene modelling for visual effects, computer games and simulation. Our results show that it is possible to have a single procedural framework, controlled through sketch-based interaction alone, which draws together all the necessary components for high-level city design. Furthermore, our approach is both more usable and more accurate when compared to conventional image map or numeric constraint specification.

There are a number of directions in which the system could usefully be extended. First, it would be interesting to connect our interface to a separate simulation-oriented city generator that is capable of exploiting the statistics and constraints that we produce. Second, there is scope to extend detailed editing of the city. For instance, the placement of individual "hero" buildings with specific dimensions is possible but not currently supported. Finally, the example-based aspect of our system could be extended by substituting real-world road map images for road pattern gestures. However, deriving statistics or even explicit road constraints in this case would require extensive image processing.

ACKNOWLEDGEMENTS

This research was supported by an NRF/THRIP grant and the Centre for High Performance Computing.

7 REFERENCES

[1] Prusinkiewicz, P., and Lindenmayer, A. The algorithmic beauty of plants. Springer-Verlag, 1996.

[2] Ebert, D., Musgrave, F., Peachey, D., Perlin, K., and Worley, S. Texturing and Modeling: A Procedural Approach. Morgan Kaufmann; 3 ed., 2003.

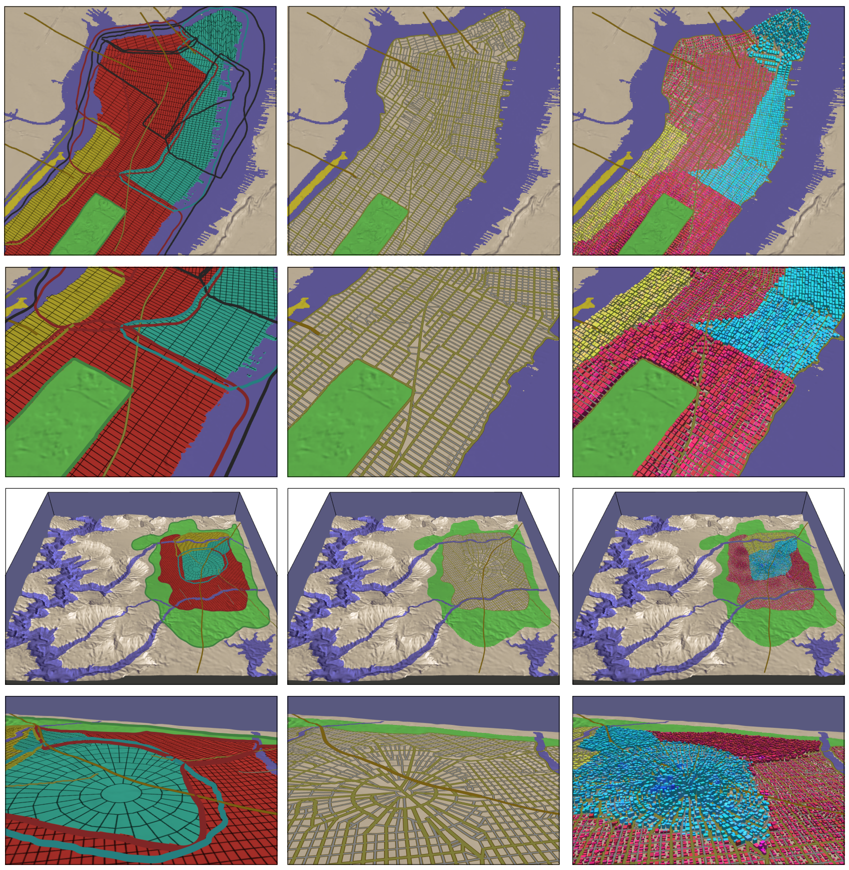


Figure 7: Results: two procedural city sketches. By row from top to bottom - Manhattan, Manhattan in close up, A hypothetical city on the edge of the Grand Canyon and its close up. By column from left to right - sketch, road network, procedural city. Both examples were designed in under 45 minutes by an experienced user.

[3] Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. Procedural modeling of buildings. ACM Trans Graph, 25(3): p. 614-23, 2006.

[4] Perlin, K., and Velho, L. Live paint: painting with procedural multiscale textures. In: SIGGRAPH '95, p. 153-60, 1995.

[5] Parish, Y., and Müller, P. Procedural modeling of cities. In: SIGGRAPH '01, p. 301-8, 2001.

[6] Hultquist, C., Gain, J., and Cairns, D. An adjectival interface for procedural content generation. In: Intelligent Computer Graphics 2009; v. 240, p. 143-65, 2009.

[7] Aliaga, D., Vanegas, C., and Benes, B. Interactive example-based urban layout synthesis. In: SIGGRAPH Asia '08, p. 1-10, 2008.

[8] Lipp, M., Scherzer, D., Wonka, P., and Wimmer, M. Interactive modeling of city layouts using layers of procedural content. Computer Graphics Forum, 30(2), p. 345-54, 2011.

[9] Gain, J., Marais, P., and Strasser, W. Terrain sketching. In: I3D '09, p. 31-8, 2009.

[10] Vanegas, C., Aliaga, D., Benes, B., and Waddell, P. Interactive design of urban spaces using geometrical and behavioral modeling. In: SIGGRAPH Asia '09, p. 1-10, 2009.

[11] Galin, E., Peytavie, A., Guerin, E., and Benes, B. Authoring hierarchical road networks. Computer Graphics Forum, 30(7), p. 2021-2030, 2011.

[12] da Silveira, L., and Musse, S.. Real-time generation of populated virtual cities. In: VRST '06, p. 155-64, 2006.

[13] Chen, G., Esch, G., Wonka, P., Müller, P., and Zhang, E. Interactive procedural street modeling. In: SIGGRAPH '08, p. 1-10, 2008.

[14] Vanegas, C., Aliaga, D., Wonka, P., Müller, P., Waddell, P., and Watson, B. Modelling the appearance and behaviour of urban spaces. Computer Graphics Forum, 29(1), p. 25-42, 2010.

[15] Dadgostar, F., Sarrafzadeh, A., Fan, C., Silva, L., and Messom, C. Modeling and recognition of gesture signals in 2d space: A comparison of nn and svm approaches. IEEE Conference on Tools with Artificial Intelligence, p. 701-4, 2006.

Intelligent Prioritization and Filtering of Labels in Navigation Maps

Mikael Vaaraniemi
BMW Forschung und Technik GmbH
München, Germany
mikael.vaaraniemi@bmw.de

Markus Goerlich
BMW Forschung und Technik GmbH
München, Germany
markusgoerlich@mac.com

Aick in der Au
TEXTURE-EDITOR GbR
München, Germany
aick@texture-editor.com

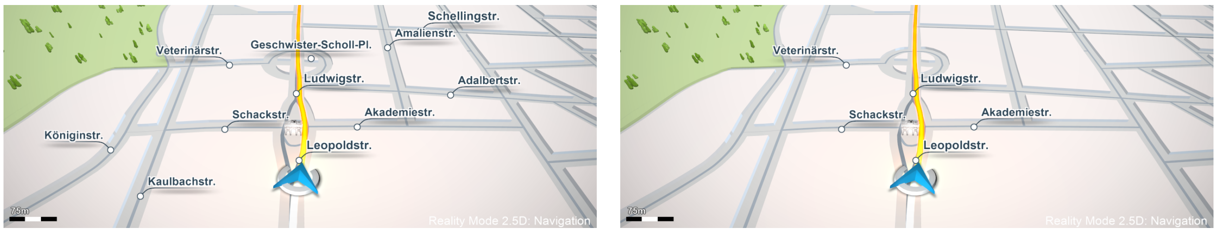


Figure 1: Filtering of labels in a 3D navigation map: (left) without filter (right) with filter.

ABSTRACT

The description of objects in navigation maps by textual annotations provides a powerful means for orientation and visual data exploration. However, displaying labels for all features leads to a cluttered map with unreadable labels and occluded information. Therefore, the overall goal is to display the most important and filter out the less important labels. In this paper, we present a general approach for filtering labels. We use the navigation in automotive maps as an application to test our approach. This involves the creation of a priority metric for ranking labels in maps. Our flexible system allows runtime configuration of the priority. Moreover, we keep the temporal coherency of label filtering; hence, jittering of labels does not occur. The system is predictable, modular, and can easily be adapted to new applications. On medium-class hardware, our real-time system is capable of filtering on average 1000 labels within 12 ms. A concluding expert study validates our approach for navigation purposes. All candidates approve the resulting clear labeling layout.

Keywords
Annotation, Labeling, Ranking, Filtering, Navigation, 3D maps, GIS.

1. INTRODUCTION

The description of objects by textual annotations provides a powerful means for visual data exploration. Compared to images, they have to be actively read, but can precisely describe objects with selected information. In Geographic information systems (GIS) labels are used to describe geospatial data, such as road networks and demographic data. In these systems, labels are important for the user's orientation and understanding of map elements. Unfortunately, the finite screen-space of desktop PCs limits the maximum amount of labels that can be displayed at the same time. This becomes even more problematic when we have a high amount of information on a small spatial area, e.g. when displaying road names of dense cities like Tokyo. The 3D view of a map accentuates this problem; Tilting the view condenses the projected spatial area onto the screen. Hence, the main problem when annotating all map features is a cluttered map with unreadable labels. Fig. 2 displays the problem, that occurs when showing all available labels of Munich in Germany at the same time.

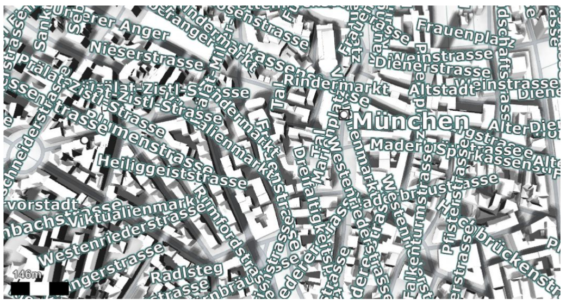


Figure 2: Display of all labels in Munich, Germany.

Therefore, to create a usable annotated map we have to prioritize all existing labels to select and display only the most important ones. Of course, removing labels is only feasible in non-critical applications.

The filtering system should follow some prerequisites. We limit the number to approximately 4 to 8 labels to stay consistent with the findings of Alvarez [AF07] and the Miller's law [Mil56]. Alvarez states in his first experiment, that a user can track between 4 fast objects and 8 slow objects on average. Additionally, more and more GIS applications have to cope with dynamic content, requiring runtime prioritization of labels without prior knowledge, e.g. when loading KML files into Google Earth. Finally, we follow the labeling rules set by Vaaraniemi et al. [VTW12]. They state that labels should behave in a temporally coherent way: they should not flicker or jump around the map.

Advantages of a label filtering approach are at hand. First, because we render less labels, we lower the overall performance impact, e.g. for placement, collision detection and rendering [VTW12]. Second, less collision between labels occur, which results in an enhanced readability. Third, the recognition and tracking of individual labels becomes easier. Fourth, the graphical association of a label with its map feature is enhanced. This is consistent with Imhof's statement [Imh75], who names legibility and the graphical association of a label with its feature as characteristics of good lettering.

1.1. Goals and Contributions

Based on the aforementioned requirements and our intended application area, we defined the goals of such a system as follows: We want to (a) display most important and filter out less important labels, (b) select approximately 4 to 8 labels, (c) create a flexible filtering system, that is configurable at runtime, (d) create a predictable, reproducible and understandable system, and (e) maintain temporal coherency where jittering of labels does not occur.

Prioritization and selection of labels is a very application specific topic. In this paper, we use navigation in 3D maps as an application to test our approach. Our proposed approach depicted in Fig. 1 has the following contributions:

- General approach for filtering a set of labels
- Priority metric for labels in navigation maps
- Temporal coherency of label filtering

In Section 2 we will present related work and survey filtering in existing commercial navigation systems. In Section 3, we will propose a system composed of 3 stages: configuration, ranking of labels and a rule-based selection. The implementation details are stated in Section 6. In Section 7, we evaluate the performance of our approach and present the findings of a conducted expert study. Finally, we conclude this paper with future research areas.

2. RELATED WORK

To assess the current state-of-the art, we first evaluated existing filtering and prioritization schemes for labels.

Simple Label Prioritization. Tatemura [Tat00] introduces fisheye maps, an approach for information exploration based on features with priorities. In his application, he uses a priority metric based on image similarities. Been [BDY06] presents an approach to label 2D maps in real-time. However, filtering is only slightly addressed: labels are selected on the basis of a geographic region and / or scale. For instance, he drops local road names when zoomed out. Bertini et al. [BRL09] introduce an excentric labeling where lenses focusses information. However, manual input for filtering labels is required: the user selects with checkboxes which labels should be displayed.

Assumed Prioritization. In most cases, the approaches describing placement and collision detection of labels are based on a ranked list [BF00, AHS05, MD06], [VTW12, VFW12]. They usually assume that the prioritization was already done in a preprocessing step. For instance, Luboschik et al. [LSC08] mention a ranking for GIS based on the feature type. Mote [Mot07] mentions a ranking based on the population of cities or the Google page ranking when displaying web searches. Moreover, Stein and Décoret [SD08] present a GPU-based real-time approach for labeling a 3D scenery. However, to compensate the drawbacks of their greedy approach, an Appolonius diagram defines the label placement order.

Conclusion. Most research on interactive labeling systems is based on the optimal placement and collision detection of labels (see Fig. 3, (violet)). However, there is almost no literature about the prioritization schemes used in existing systems. In most approaches, the ranking is defined by the importance of the map feature or set with simple characteristics, e.g. the distance to the camera. Hence, to better grasp related filtering systems, we decided to survey commercial navigation systems.

2.1. Survey: Filtering of Labels in Commercial Navigation Systems

Filtering of labels is an essential component of current navigation systems. However, in most cases it is unknown how the filtering of labels works and which algorithms are internally used. Hence, we conducted a survey with several navigation devices, namely, the TomTom Go 940 Live, Garmin nüvi 765T, Falk F10, Becker Z108 and the Navigon Select App for iOS.

2.1.1. *Quality: Readability*

In our first test, we evaluated the quality of the labeling. For every device, we counted in two different cities at eight locations all labels displayed in the map. We counted the labels at different zoom levels: the default zoom while driving and the maximum zoom level. Then, for each label, we checked it's readability. This survey can be see in Table 1.

Readability of Labels. As can be seen in Table 1, only the TomTom device achieved a readable labeling at every zoom level. All the other devices rendered labels, which collided or became clipped by the outer edges of the screen.

Navigation System	Default zoom level		Maximum zoom level	
	Total #	Readable #	Total #	Readable #
TomTom Go 940	9	9	4	4
Garmin nüvi 765T	9	5	2	2
Falk F10	12	7	5	4
Becker Z108	11	7	6	4
Navigon select	14	10	6	3

Table 1: Results of our survey of the readability of labels at different zoom levels in commercial navigation systems. This table shows the total number of labels and the number thereof of readable labels. The numbers are averaged over eight different locations.

2.1.2. *Filtering Criterias*

In the following tests, we tried to comprehend which criteria are used in the filtering of labels. This was done by following the same navigation route in the city of Munich in Germany with each device.

Repetitions of Road Labels. Not a single device repeats the road names, even if there would be enough screen-space.

Filtering: Horizontal Roads. All devices seem to prioritize roads which are rendered horizontally in screen-space. Garmin tries to show only intersecting roads.

Current Road Label. TomTom and Falk always show the current road on the map. The other devices display the current road as a separate layer at the bottom of the screen.

Filtering: Route vicinity. Only TomTom and Garmin use sometimes the route as a filtering criteria. On the other hand, Falk, Becker and Navigon try to squeeze as many labels as possible on the screen.

Filtering: Distance to user. Only Garmin and Becker prioritize labels near to the current user's position.

2.1.3. *Conclusion*

Every system uses it's own combination of filtering criteria. However, not a single one uses all criteria to create a good labeling layout with all important labels.

3. 3-STAGE FILTERING SYSTEM

The key idea of our approach is to create a ranking of all labels by computing a score for each label. This score depends on the label's type and the spatial relationship to the map environment. Using this priority list, we can intelligently select and filter the displayed labels. The result can be seen in Fig. 1 and Fig. 4.

Our filtering system depicted in Fig. 3 is composed of 3 stages:

1. *Configuration* – The first stage configures the filtering system depending on the current situation. For example, we can configure the system to output the four most important road labels if we drive along a route. Moreover, when we are searching the map for restaurants, we could restrict the labels to restaurant POIs and the surrounding roads.
2. *Label Ranking* – The second stage creates a ranking of all existing labels using a scoring system. It is computed using scoring modules called *evaluators*. The following Section 4 describes all types of modules with their score computation.
3. *Rule-Based Selection* – The last stage is a rule-based system which uses the ranking from last stage and predefined rules to select the labels to display. This stage enforces rules to ensure a specific behavior, e.g. limiting the maximum amount of labels.

Our approach distinguishes the following categories:

- City Labels, point features.
- Point Of Interest (POI), point features.
- Road Labels, line features.

City and POI labels describe a point feature and have a fixed 2D position on the map. Road labels describe a line feature stored as a 2D polyline [VTW11]. The road's label can be placed along the entire polyline composed of linear street segments.

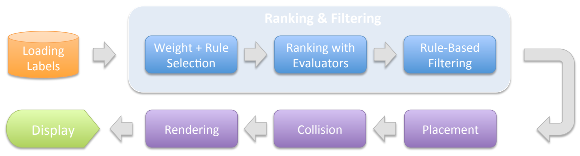


Figure 3: Overview of the entire labeling pipeline: After loading the labels from the map database (orange), the 3-stage filtering system (blue) presented in this paper filters the labels. Then, the selected labels are sent for placement onto the screen (violet). At runtime, we compute if any collision between labels occurs to finally render them to the display (green).

4. LABEL RANKING: EVALUATORS

We evaluate the score of a single label with a ranking system composed of evaluator modules. Each label is sent to each evaluator to compute an arbitrary score. Finally, the scores of a label i are summed up to create a final score $score(i)$.

4.1. Evaluators

We define evaluators based on the following criteria:

- Category Evaluator: score is based on the label's road or city or POI class
- Placement Evaluator: score is based on the past label's visibility to the user
- Distance Evaluator: score is based on the distance between the label and the user's position
- Route Evaluator: score is based on the vicinity of the label to the route
- Driving Direction Evaluator: score relates to the driving direction
- Road Angle Evaluator: score is based on the relationship between the driving direction and road direction
- Road Length Evaluator: score is based on the ratio between the road segment and the label's length

Furthermore, each evaluator specifies separately if higher or lower scores are better.

The presented evaluators were created mainly to rank labels of a 3D navigation map. Transferring our system to other application areas should be easily possible but would require a re-evaluation of each evaluator and possibly requires adding new ones.

4.1.1. Category Evaluator

This basic evaluator scores labels based on the category of their corresponding map element, i.e. the road class or the city's importance. These categories are precompiled into the map database and fetched at runtime. Hence, the resulting score is:

$$score(i)_{cat} = score(i)_{db} \tag{1}$$

Usually, highways have the highest score, followed by federal roads and main roads. Streets of lower importance, such as ordinary urban roads or walking paths, receive a lower score. The larger a city is, the more likely it is probable that it is known to the driver and thus contributes to a better orientation. Therefore, capitals and major cities return high scores. The less important a city is, e.g. in terms of population number, the lower the score gets.

4.1.2. Placement Evaluator

The placement evaluator is the key element to achieve a temporally coherent label filtering. The idea of this evaluator is, that already displayed labels are boosted through a higher score. Such a scoring prevents jittering, e.g. when labels become visible for a brief moment and disappear when another label achieves a slightly higher score.

$$score(i)_{place} = \begin{cases} 1 & \text{if label } i \text{ visible} \\ 0 & \text{else} \end{cases} \tag{2}$$

4.1.3. Distance Evaluator

The distance evaluator prioritizes labels around the user's location, e.g. the current car position (CCP). Therefore, the score is given by the distance between the label and the user's current position.

For the user's position p_{ccp} and the label i with position p_i the score is computed as follows:

$$score(i)_{dist} = \|p_i - p_{ccp}\| \tag{3}$$

4.1.4. Route Evaluator

This evaluator is used when an active route guides the user to his destination. The goal is to achieve a greater focus on labels positioned closer to the route. Therefore, for each candidate position p_i of a label i, the shortest distance d to the route is calculated. This distance does not map linearly to a score, since a label that is twice as far away of the route is not necessarily half as important. Instead we use a logarithmic function to create a focus tube around the route.

For label i with position p_i and the shortest distance d of label i to the route we get the following score:

$$d = \|p_i - \text{route}\| \tag{4}$$

$$score(i)_{route} = log_d * \sqrt{d} \tag{5}$$

4.1.5. *Driving Direction Evaluator*

The driving direction evaluator focuses on the labels ahead of the user. This is achieved by computing the deviation of a label's position from the direction of travel.

The driving direction is given by a 2D angle α_{ccp}. Moreover, we compute the vector between the user's position p_{ccp} and the label's position p_i. This is transformed to an angle α_{label_i}. We compute the differential angle spanned between the driving direction and the label's position.

With the normalized deviation $\hat{p}$ and the corresponding angle α_{label_i} we get:

$$\hat{\mathbf{p}} = (p_{ccp} - p_i)/\|p_{ccp} - p_i\| \tag{6}$$

$$\alpha_{label_i} = atan2(p_y, p_x) \tag{7}$$

$$score(i)_{direction} = \|\alpha_{ccp} - \alpha_{label_i}\| \tag{8}$$

We define the angle to be 0° when the label is in front of the user, 90° states that it's on either side and 180° if it's directly opposite to the direction of travel.

4.1.6. *Road Angle Evaluator*

The goal of this evaluator is to increase the score of labels from roads stretching vertically to the driving direction. Such roads mainly include crossroads and intersections that are especially relevant for navigation maneuvers. Even if this is a coarse simplification, we assume that most of these roads can be reached directly from the current route. In contrast, parallel roads can usually not be reached without entering another road first.

To compute the score, the direction of travel α_{ccp} is determined. Then, for each label candidate position p_i, the angle α_{road} of the corresponding road segment is computed. The smallest differential angle α_{angle} is used to create the score $score(i)_{angle}$ as follows:

$$\alpha_{angle} = \|\alpha_{road} - \alpha_{ccp}\| \tag{9}$$

$$score(i)_{angle} = \begin{cases} cos(\alpha_{angle}) & \text{if } \alpha_{angle} \leq 90° \\ 0 & \text{if } \alpha_{angle} > 90° \end{cases} \tag{10}$$

The advantage of this approach is that we do not need a complex graph-based structure of the road network to find intersecting roads. However, this evaluator still needs the road's geometry, i.e. a polyline. This means that neither POIs or city labels can be scored.

4.1.7. *Road Length Evaluator*

The road length evaluator prevents the placement of labels which names are longer than their corresponding road in screen-space. If the label would still be shown, it would hide the road, suggest a longer road and even encompass on neighboring roads.

The score $score(i)_{length}$ is based on the screen-space ratio between the length of the name compared to the projected road's length. First, we project the road's polyline into screen-space and compute the length l_{road}. Second, we compute the screen-space length of the rendered label's name l_{label}. If the label is shorter than the projected road segment, it receives a perfect score of 1. If the name is longer, the score is given by the ratio of the road's length and the label length.

$$score(i)_{length} = \begin{cases} 1 & \text{if } l_{label} \leq l_{road} \\ l_{road}/l_{label} & \text{else} \end{cases} \tag{11}$$

Again, this evaluator works only for roads and no other label categories can be considered.

4.2. Final Score

We have computed for every label i and each evaluator k the $score(i)_k$. This score is normalized to $score(i)'_k$ as described in Section 6.2. Moreover, we introduce weights w_k to configure the influence of each evaluator k. Each normalized score $score(i)'_k$ is factored with his weight w_k and summed up to a final score for label i:

$$score(i)'_k = \frac{score(i)_k}{\max\limits_{1\leq j\leq n}(score(j)_k)} \quad \text{(n: #labels)} \tag{12}$$

$$score(i)_{global} = \sum_k score(i)'_k * w_k \quad (k \in \text{evaluator}) \tag{13}$$

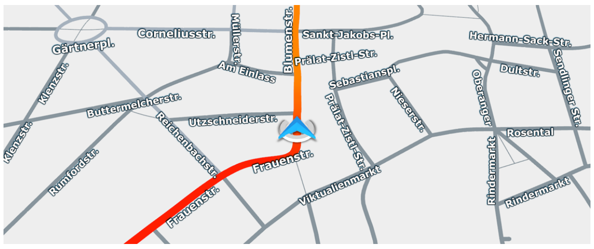
Computing this for every label results in a global ranking needed for the intelligent selection described in the following Section.

5. CONFIGURATION & RULE-BASED SELECTION

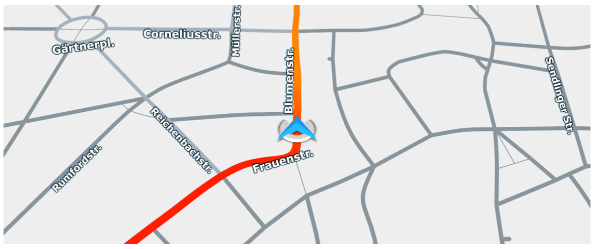
Selecting labels solely based on the ranking can not ensure a desired result. We solve this problem by introducing two methods: Based on the current context and the desired result we (a) configure the evaluators' weights and (b) instate predefined rules to select labels.

5.1. Configuration: Weights

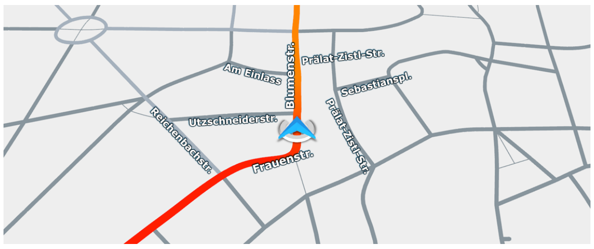
It is necessary to change the effect of the individual filters in specific situations. Using weights, the influence of a filter on the overall score can be changed. In different modes, different weights are needed. For instance, it makes no sense to use the route evaluator when there is no route. Analogously, when the user is exploring the map, the distance and direction evaluators become irrelevant. In the following Table 2, a configuration example of the weights w_k for the evaluators k can be found.



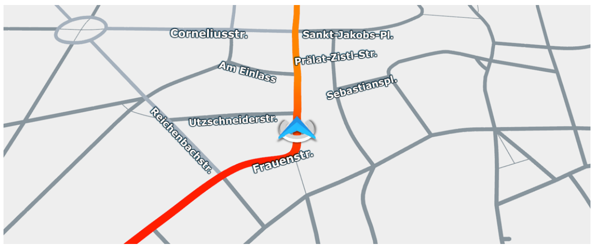
(a) No filtering.



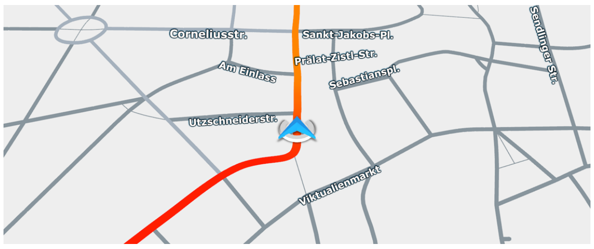
(b) Category Evaluator: Filtering based on the label's type.



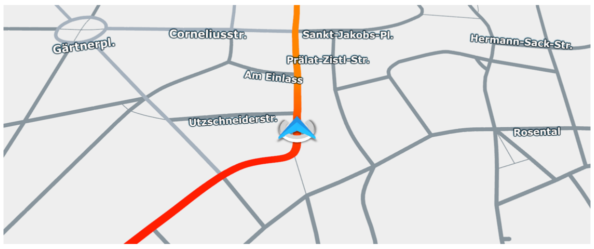
(c) Distance Evaluator: Filtering based on the distance to the user.



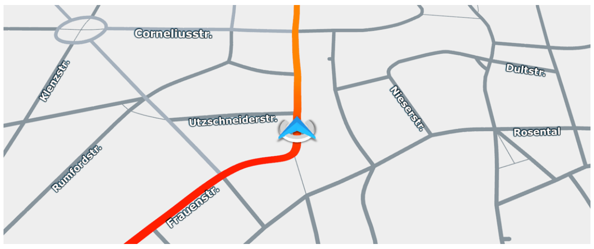
(d) Route Evaluator: Filtering based on the route's vicinity.



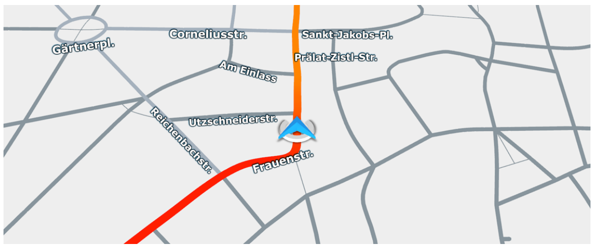
(e) Driving Direction Evaluator: Filtering based on driving direction.



(f) Road Angle Evaluator: Filtering based on driving vs. road direction.



(g) Road Length Evaluator: Filtering based on name vs. road length.



(h) All: Filtering based on all evaluators.

Figure 4: Our approach is based on evaluators ranking the labels with the help of map and user properties. This ranking is then used to filter the set of labels. In this figure, the current user's position is marked with the blue arrow and the user's navigation route is colored in orange. To display the impact of every evaluator, we limit the label number to eight and let a single evaluator process all labels.

5.2. Rule-Based Selection

The last stage of our system is a rule-based selection of labels based on the overall configuration and the ranking generated by the evaluators. These rules can overrule the ranking and ensure a specific behavior. We instate the following rules:

Repetition of Labels. At predefined zoom thresholds, we allow the repetition of labels for longer line features, e.g. motorways, rivers and country borders.

Maximum Number of Labels. We limit the maximum number of labels to be displayed.

Zoom-dependent Rule. Depending on the current map zoom level we hide or display certain label categories. Generally speaking, we remove predefined road or city classes when zooming out. For instance, when zooming out, we stop showing road names. In contrast, when zooming closely into the map, we stop showing country names.

Special Labels. We also instate a special rule to ensure the display of special labels. In a navigation system, some labels should always be shown, e.g. the start, the destination, and the current road.

Evaluator	Information Mode Weight	Route Mode Weight
Category	0.8	0.5
Placement	0.6	0.7
Distance	0.0	1.0
Route	0.0	1.0
Driving Dir.	0.0	0.5
Road Angle	0.7	0.5
Road Length	0.9	0.8

Table 2: Two configuration examples of the evaluator's weights. In the route mode, we have an active route and a current user's position. Hence, all labels around the user, the route and intersecting roads are important. In the information mode, we do not have a route and the exploration of the map with all it's labels becomes the most important task. Setting weights to 0.0 disables irrelevant evaluators.

6. THE LABEL SCORING SYSTEM

Until now, we assumed that labels have a single position. However, line features like roads are composed of several line segments. Their annotation can be placed at any position along these segments.

6.1. Road Labels: Candidate Scores

As can be seen from the given examples, most evaluators require a fixed position for a label, e.g. to determine if a label is near the route. For point features like city and POI labels, each evaluator returns a single score because they are stored with a single position in the map database. However, road labels could be placed at any position along the road's geometry. We simplify this to achieve real-time computation: for each road label we store all its linear road segments. Such road segments have a limited resolution and do not always match the curvature of the road as illustrated in Fig. 5. We allow the placement of road labels at every mid-point of a road segment.

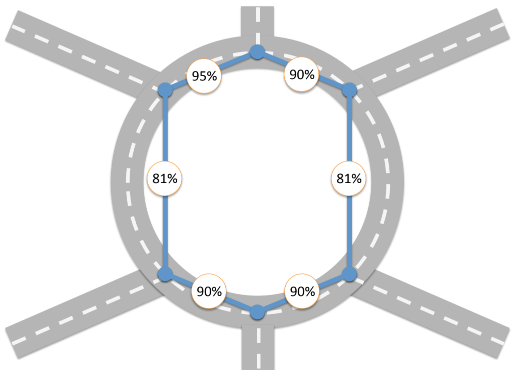


Figure 5: Road segments as candidates for the placement of a roundabout's label. Possible placement positions for road labels exists only at segment mid-points.

Candidate Scores. For each road label, we store one total score that determines the priority of the entire road and a second set of candidate scores, one for each segment, that determines the best placement position. Both score types are independent and cannot be compared with each other. The total score for road labels is calculated by taking the maximum score of all it's candidates.

As a result, the label placement system needs to check for a given label if it can be placed at a candidate position. It checks all candidate positions in descending sequence of its candidate scores. With both score types the label filtering system can be kept independent from other parts of the label placement system. Information about label collisions are not needed and all labels can be processed independently from each other.

Hence, for road labels and if the evaluator requires a position, we need to compute multiple candidate scores – one for each segment's mid-point. Evaluators which are unrelated to the label's position, i.e. the placement evaluator, just distribute their score to all segments.

Example. Consider the case depicted in Fig. 6 where labels for Street 1 and Street 2 need to be placed: Label *Street 1* (belonging to the left road) has the highest priority with a score of 95%. Therefore, it's the first label to be placed at C1 (green) into the screen. Label *Street 2* with a score of 93% has three possible candidate positions (blue): The left segment with the highest candidate score (C1) of 99%, C2 with 50% and C3 with 0%. We would like to place the label *Street 2* at C1 but it fails due to collision with the label *Street 1*. Therefore, because C2 (blue) represents the second best score, the label *Street 2* places itself onto C2.

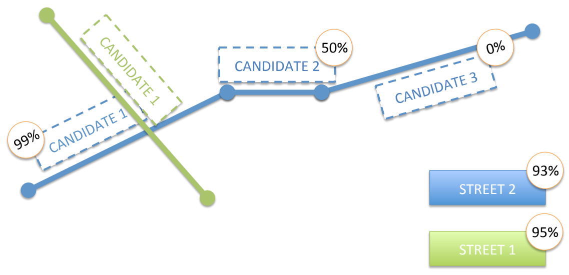


Figure 6: Road labels store their overall score (bottom, right) and additional candidate scores to determine the best possible placement along a road.

6.2. Score Normalization and Weighting

In a second step, we normalize all scores. The total score for each label is calculated by taking the weighted sum of all resulting scores as shown in Table 3(a). Normalization is done by dividing by the maximum score over all labels of each evaluator. For road labels consisting of multiple segments, we take the score from the best segment, i.e. depending on the evaluator either the minimum or maximum (see Sec. 4.1). Candidate scores for road labels are evaluated in a very similar fashion: Normalization is done by dividing by the maximum candidate score over all segments for each evaluator (see Table 3(b)).

7. RESULTS

The implementation of this filtering system was programmed with C++ and OpenGL 3.2 and runs on Windows 7 (x86). In this section we analyze the runtime behavior of the system followed by the results from the conducted expert study.

7.1. Performance

In our application, the number of labels n the system is processing varies between ≈ 500 in sparely and ≈ 9000 in densely populated areas. This high variation of n is caused by several properties: (a) level-of-detail of the map, (b) zoom level, and (c) the map database vendor. On average, we observed a range of ≈ 1000 to ≈ 4000 labels to be filtered.

The Route Evaluator is the most demanding evaluator of our approach because it requires nearest neighbor queries to the route. These queries can be computed in $O(\log m)$ time [AMN+98] with m being the number of positions of the route's polyline. In our implementation reasonable m with $m < 10.000$ do not impact the performance. Overall, the filtering system's complexity is $O(n \log m)$.

The performance results may vary depending on the activated evaluators, configuration settings and applied rules. The timing in ms for two different configurations can be seen in Fig. 7. Assuming a medium-class hardware Intel Core 2 Quad CPU clocked at 2.66 GHz, our system is capable of filtering 1000 labels within 12 ms on average. In our implementation, this is computed in

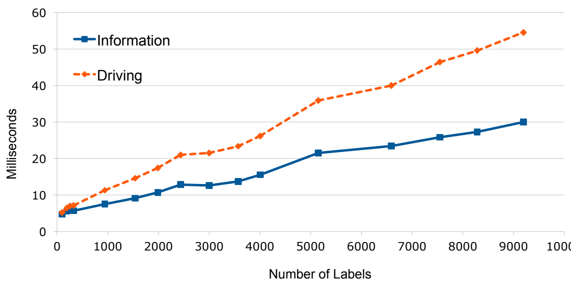


Figure 7: Performance measurements: We measured the computation time of the labeling pipeline in two different configuration: (stippled) driving mode with all evaluators enabled and (solid) information mode where the route and distance evaluator are disabled.

the rendering thread. In extreme scenarios (> 2000 labels), this could impact the overall rendering framerate. Therefore, for an enhanced performance, this system could be executed in a separate thread. This is possible because our approach is completely decoupled from the loading, placement, collision and rendering parts (see Fig. 3, violet steps).

7.2. Expert Study

We conducted an expert study at a research facility to evaluate the resulting filtering layout.

Candidates. We selected five candidates aged between 30 and 40 years with a strong background in automotive navigation: three male engineers, one person with a design and user experience background, and a psychologist specialized in user experience. Everyone had experience with commercial 3D navigation systems.

Study Design. As an introduction, we showed movies with a camera following a preset navigation route and our active label filtering. We requested the candidates to think about the shown selection of labels.

The first part of the study was designed to evaluate and compare different configurations of the filtering system. We showed four printed maps with the results of our system. Each time, the configuration of the filtering system was changed. Then, the candidates had to score the labeling quality depicted in each map on a scale 1..10 (10=best). In the second part of the study, they had to write labels manually on a printed map depicting a road network with a navigation route and a current car position icon (CCP). In the last part, we asked them which map and user criteria should influence the label filtering of a navigation map.

7.2.1. Results

Part 1 – Scoring the Filtering. On average, our filtering approach depicted in the printed maps was scored 6.62 (all scores: 6.67, 6.83, 7.33, 7.0, 5.25). The first map with 8 labels in route mode had an average score of 7.2. The second map with 14 labels in information mode scored 5.8. When comparing the route and the information mode filtering, the route mode always received higher averaged scores (first map 6.8 to 6.4, second map 7.75 to 6.5).

We deduce that our filtering approach was well received and that maps with less labels are preferable.

Part 2 – Writing Labels. Fig. 8 depicts the results of the manual placement of labels by our experts. We encoded the number of votes to different colors: At least three of the candidates voted for green labels, one of five voted for the white labels, and red labels did not receive any votes.

We deduce that the experts prefer labels around the user's position and on crossing roads.

Table 3: Normalization of the label's scores computed by the evaluators.

(a) Scores for k evaluators and n labels.

Evaluator	Label 1	$\cdots$	Label n
$1: Category$	$score(1)_1$	$\cdots$	$score(n)_1$
$\vdots$			
$k: Length$	$score(1)_k$		$score(n)_k$
$\mathbf{score(i)_{global}}$	$\sum_{j=1}^{k}\left(\frac{score(1)_k}{\max\limits_{1\le j\le n}\left(score(j)_k\right)}w_k\right)$	$\cdots$	$\sum_{j=1}^{k}\left(\frac{score(n)_k}{\max\limits_{1\le j\le n}\left(score(j)_k\right)}w_k\right)$

(b) Candidate scores for a road label for k evaluators over m segments.

Evaluator	Segment 1	$\cdots$	Segment m
$1: Category$	$score(1)_1$	$\cdots$	$score(m)_1$
$\vdots$			
$k: Length$	$score(1)_k$		$score(m)_k$
$\mathbf{score(i)_{seg}}$	$\sum_{j=1}^{k}\left(\frac{score(1)_k}{\max\limits_{1\le j\le m}\left(score(j)_k\right)}w_k\right)$	$\cdots$	$\sum_{j=1}^{k}\left(\frac{score(n)_k}{\max\limits_{1\le j\le m}\left(score(j)_k\right)}w_k\right)$

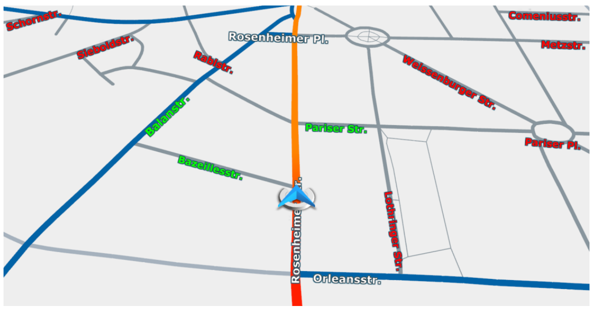


Figure 8: Printed map area we showed to our candidates. The current navigation route is displayed in orange. Each expert had to place labels on an empty map (without labels). The most voted labels are green.

Part 3 – Questions. First, we asked how many labels should be placed in a map. All experts agreed on a very low number of labels, (1 of 5) to display the current road only, and (1 of 5) to display at most 6 road labels.

Then we asked, which criteria are important for a good label placement. All experts defined the vicinity of the route as important and (4 of 5) the next crossroads. (2 of 5) mentioned the driving direction, (1 of 5) the road class, and (1 of 5) the road's length. Then, (2 of 5) mentioned landmarks and (1 of 5) well-known roads to help orientation. Districts (1 of 5), the current road (1 of 5), distance to destination area (1 of 5) were also mentioned.

Moreover, we asked how important are road labels behind the current user's position. The majority of experts (3 of 5) stated they are not important.

Finally, we asked if repetitions of road labels would be useful at higher zoom levels. Only (2 of 5) experts stated they are useful if the road branches or the course becomes unclear.

Conclusion. The results of this study shows the importance of the distance to the driving route and the current user's position on the label selection strategy. Moreover, roads crossing the current road are an important factor. All statements indicate a preference to display a minimum amount of labels. On top, the filtering system should be modular to satisfy even more criteria.

Finally, even if our filtering system was well received, we increased the weights of the route and distance evaluators to better fit the new findings. The resulting weights can be seen in Table 2.

8. CONCLUSION

In this paper, we presented an approach to filter labels in navigation maps. We introduced a 3-stage label filtering pipeline. First, it computes a score using multiple evaluators based on user's and map properties. This creates a ranking for all existing labels. Then, based on these results, predefined rules selects which labels should be displayed. With this system, we can achieve all our goals set in Section 1: The ranking creates a metric for the importance of labels in navigation maps. Then, we can limit the number of labels to a fixed amount depending on the current navigation context using the rules from Section 5. Temporal coherency is achieved with a placement evaluator (see Section 3). Moreover, our system is highly flexible as we can change the prioritization metric at runtime. Finally, the filtering system can easily be adapted to new applications.

On an Intel Core 2 Quad CPU at 2.66 GHz, our system is capable of filtering on average 1000 labels within 12 ms. Finally, the candidates of our expert study approved the clear labeling layout created by our approach.

In further research, we would like to find heuristics to detect unclear road courses where label repetions would be useful. Then, we plan to evaluate our approach while driving in a real world scenario. This should give more insights on the quality of the priority metric. Furthermore, we would like to include the user's preferences as an additional evaluator module.

9. REFERENCES

[AF07] Alvarez, G. A. and Franconeri, S. L. How many objects can you track?: Evidence for a resource-limited attentive tracking mechanism. *Journal of Vision*, 7(13):14, 2007.

[AHS05] Ali, K., Hartmann, K., and Strothotte, T. Label layout for interactive 3d illustrations. *Journal of the WSCG*, 13(1):1–8, 2005.

[AMN$^+$98] Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998.

[BDY06] Been, K., Daiches, E., and Yap, C. Dynamic map labeling. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):773–780, 2006.

[BF00] Bell, B. A. and Feiner, S. K. Dynamic space management for user interfaces. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, UIST '00, pages 239–248. ACM, 2000.

[BRL09] Bertini, E., Rigamonti, M., and Lalanne, D. Extended excentric labeling. In *Proceedings of the 11th Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'09, pages 927–934. Eurographics Association, 2009.

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

[LSC08] Luboschik, M., Schumann, H., and Cords, H. Particle-based labeling: Fast point-feature labeling without obscuring other visual features. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1237–1244, November-December 2008.

[MD06] Maass, S. and Döllner, J. Efficient view management for dynamic annotation placement in virtual landscapes. 4073:1–12, 2006.

[Mil56] Miller, G. A. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[Mot07] Mote, K. Fast point-feature label placement for dynamic visualizations. *Information Visualization*, 6(4):249–260, 2007.

[SD08] Stein, T. and Décoret, X. Dynamic label placement for improved interactive exploration. In *Proceedings of the 6th international symposium on Non-photorealistic animation and rendering*, NPAR '08, pages 15–21. ACM, 2008.

[Tat00] Tatemura, J. Dynamic label sampling on fisheye maps for information exploration. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, pages 238–241. ACM, 2000.

[VFW12] Vaaraniemi, M., Freidank, M., and Westermann, R. Enhancing the Visibility of Labels in 3D Navigation Maps. In *Progress and New Trends in 3D Geoinformation Sciences*, Lecture Notes in Geoinformation and Cartography, pages 23–40. Springer, 2012.

[VTW11] Vaaraniemi, M., Treib, M., and Westermann, R. High-Quality Cartographic Roads on High-Resolution DEMs. *Journal of WSCG*, 19(2):41–48, 2011.

[VTW12] Vaaraniemi, M., Treib, M., and Westermann, R. Temporally Coherent Real-Time Labeling of Dynamic Scenes. In *Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications*, COM.Geo '12, pages 17:1–17:10. ACM, 2012.

Detecting Dominant Motion Flows and People Counting in High Density Crowds

Sultan Daud Khan, Giuseppe Vizzari, Stefania Bandini
Complex System and Artificial Intelligence,
University of Milano-Bicocca
name.surname@disco.unimib.it

Saleh Basalamah
GIS Technology Innovation Center,
Umm Al Qura University
smbasalamah@uqu.edu.sa

ABSTRACT

Urbanisation is growingly generating crowding situations which generate potential issues for planning and public safety. This paper proposes new techniques of crowd analysis and precisely crowd flow segmentation and crowd counting framework for estimating the number of people in each flow segment. We use two foreground masks, one generated by Horn-Schunck optical flow, used by crowd flow segmentation, and another by Gaussian background subtraction, used by crowd counting framework. For crowd flow segmentation, we adopt K-means clustering algorithm which segments the crowd in different flows. After clustering, some small blobs can appear which are removed by blob absorption method. After blob absorption, crowd flow is segmented into different dominant flows. Finally, we estimate the number of people in each flow segment by using blob analysis and blob size optimization methods. Our experimental results demonstrate the effectiveness of the proposed method comparing to other state-of-the-art approaches and our proposed crowd counting framework estimates the number of people with about 90% accuracy.

Keywords
Crowd analysis, clustering, crowd counting, crowd flow segmentation, crowd counting

1 INTRODUCTION

As the population of world is increasing and ever more located in urban areas, public safety is becoming a problem in most crowded areas of the big cities. Mass events like those related to sports, festivals, concerts, and carnivals attract thousands of people in constrained environments, therefore adequate safety measures must be adopted. Despite all safety measures, crowd disasters still occur frequently. The reasons of these disasters is mostly the presence of different and conflicting motion patterns that influence the crowd. A crowd is composed of small groups of people, for instance due to social relationships (families or friends) or a common goals, like reaching a certain point of the environment. The latter groups can be called short term coherent groups because they discontinue their cohesion after completing the goals (e.g. reaching an exit, completing a movement). Detecting the second kind of group, essentially associated to a certain flow of pedestrians in the environment, can be important to be able to prevent conflict situations.

Due to the complex dynamics of the crowd, crowd management is becoming a daunting job where huge effort from the security staff is required to manage the potentially problematic situations. In such high density crowded areas, surveillance cameras are generally installed in different locations that can even cover the whole scene. Detecting specific activities in real-time videos is the task of analysts sitting in surveillance room and watching over multiple Tv screens. Such manual analysis of high density crowds is a tedious job and usually prone to errors. Therefore we need automatic analysis of the crowd which can reliably estimate the density and detect specific activities. Creating such kind of virtual analyst has become the focus of many researchers. This research has a wide range of application domain in crowd management, public space design, underwater fishes analysis (and animal behavior studies in general), and cell population analysis. In video surveillance, “detection and tracking” are the core technologies but these technologies are likely to fail in high density crowded scenarios. In this paper, we propose a framework that tackles problems of crowd flow segmentation, crowd counting and consists of three parts: foreground extraction, crowd flow segmentation, and crowd counting. The paper is organized as follows: the following section will briefly introduce related works. In Sect. 3 we shall describe propose framework. In

Sect. 4 we shall discuss experimental results. Conclusions and future development will end the paper.

2 RELATED WORKS

For more than 40 years researchers have been studying pedestrian dynamics with the aim of supporting the design of pedestrian facilities. Since the population of the world is increasing and concentrating in urban areas, and due to the growing relevance of mass events (like sports contests, concerts and festivals periodically arranged and attracting growing number of people from different parts of the world), adequate safety measures are becoming ever more important. More recently, researchers are focusing on studying crowd dynamics in order to improve the evacuation strategies in emergency situations.

2.1 Motion Flow Segmentation

An important contribution that automated analysis tools can give to pedestrians and crowd safety is the detection of conflicting large pedestrian flows: this kind of movement pattern, in fact, may lead to dangerous situations and potential threats to pedestrians' safety. Therefore, segmenting typical flow patterns of crowd and estimating the number of people in crowd are important steps to understand overall crowd dynamics. Crowd flow segmentation has multiple benefits: (1) enables clutter free visualization of moving groups; (2) it is independent from "detection and tracking"; (3) provides input for the pedestrian simulation models (in terms of data for simulation initialization or validation). Automatic analysis of the crowd has become the center of focus for most of researchers in computer vision. Detecting pedestrians and tracking are traditional ways of crowd analysis. Most algorithms developed for object detection and tracking work well with pedestrians in low density crowds where the number of people is generally less than twenty individuals in a single frame, but with higher densities (where the number of people in a frame can be in the order of hundreds), detection and tracking of individuals are almost impossible due to multiple occlusions.

Therefore, the research has focused on gathering *global motion information* at higher scale. Global analysis of dense group of moving people is often based on *optical flow analysis*. [AS07] proposed particle dynamic segmentation of crowd flows by detecting the lagrangian coherent structures over the phase space. But their proposed method is computationally expensive because of the calculation of FTLE and also could not detect small flows. [OYA10] used SIFT features to detect dominant motion flows. Flow vectors of SIFT features are calculated and then motion flow map is divided into small regions of equal size. In each region, dominant motion flows are estimated by clustering flow vectors. [EB08] proposed spectral clustering method for crowd flow segmentation by computing sparse optical flow field. Crowd flow is estimated using multiple visual features reported by [SND11] where flow is estimated by the number of persons passing through a virtual trip wire and accumulate the total number of foreground pixels. Min-cut/max flow algorithm is used by Ullah et al. [UC12] for crowd flow segmentation. In all of above four methods, we can not find clear boundaries among different flows. Crowd flow segmentation by using histogram curves is reported by [LRZ12] where angle matrix of foreground pixels is segmented instead of optical flow foreground. The derivative curve of histogram is used to segment the flow. Since this method only looks to the peaks of histogram curve, therefore it loses information about the crowd flow. Our proposed approach for detecting dominant flows is similar to spectral clustering in [EB08]. The difference is that we carry out segmentation by employing K-means clustering on dense optical flow field. After K-means clustering, small blobs can appear especially at the boundaries of conflicting flows, generated by the optical flow computation but not really associated to actual pedestrian movements; these small blobs are removed by our blob absorption method. Comparing to other approaches, our approach can detect large as well as small flows, and by employing blob absorption method, we generate clearer boundaries between different flow segments. We show the effectiveness of our proposed motion segmentation approach by comparing with state-of-the-art approaches in Sect. 4.

2.2 Crowd Counting

Most of the literature in field of crowd density estimation has focused on segmentation of *people or head counts*. Some of the work focused on texture analysis or wavelet descriptor for estimating crowd density. Bayesian based segmentation proposed by [ZN03] to segment and count the people but this method fails in high density scenarios as because of severe occlusions. [YST10] extract blob features of moving objects and neural network is trained to estimate number of pedestrians in each blob. [XLH06] classified crowd density into four classes by using wavelet descriptors. [MHL08] used texture descriptors to estimate crowd density. [TYOY99] count the number of people as they cross some virtual line. [HMY^{+}97] used infra-red imaging to count the number of people in crowd. Simple background subtraction from static images to estimate the crowd density was proposed by [RP07]. Background removal concept is used to estimate the crowd area by [VYD^{+}93, VYD^{+}94]. [RMAS04] used a forward facing camera mounted on the car to detect crowd of pedestrians. Trained support vector machines using HAAR transform is used by [LCC01] to identify heads of people. Median background computing technique

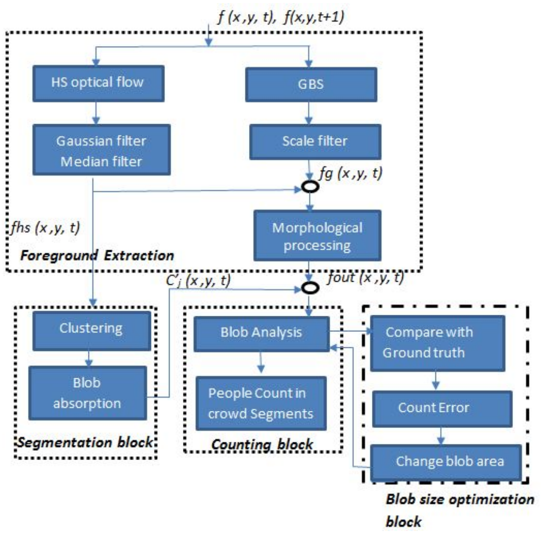


Figure 1: Overview of proposed framework

is used by [RP07] to extract foreground pixels. Support vector machine, K-nearest neighbor, PNN, BPNN are used to classify images in two categories (zero persons, one or more persons). As the sensor are becoming cheap, therefore, recently many researchers count the people using infra-red sensors. [TS07] proposed lightweight camera sensor nodes to count the people in the indoor environment based on motion histogram. Recently many infra-red sensors specifically designed for people counting are available in the market[1]. Our approach starts from the results of motion segmentation to perform people counting, estimating the number of pedestrians in each dominant flow by blob analysis and blob size optimization methods. This allows having a more selective and informative information on where the counted pedestrians are headed. Moreover, compared to the above methods, our approach is aimed at supporting people estimation even in high density crowds. As we will show in Sect. 4 we achieve satisfactory performance on people counting in experiments performed adopting different videos.

3 PROPOSED FRAMEWORK

Our proposed framework is composed of four processing blocks, Foreground extraction, segmentation, counting and blob size optimization block, but this block only executes in the beginning for few initial frames. In this section, we will discuss each processing block in detail. For sake of description of the proposed approach we will employ videos taken from a crowd related data set from UCF [AS07].

[1] See e.g. `http://www.sensourceinc.com/thermal-video-imaging-people-counters.htm` or `http://www.irisys.co.uk/people-counting/our-products/`.

3.1 Foreground Extraction

Foreground extraction is the most important preprocessing step for detecting the moving objects from the video and therefore forms the basis of our framework. Foreground extraction is useful for detection, tracking and understanding the behavior of the object. A survey on motion detection techniques can be found in [MG01]. Traditionally, in video surveillance with a fixed camera, researchers use background subtraction method, where foreground objects are extracted from video if the pixels in the current frame deviate significantly from the background. In this paper, we use two foreground masks as in [LWMZ10], one generated by optical flow, $f_{hs(x,y,t)}$ and will be used by crowd flow segmentation framework and other is Gaussian background subtraction, $f_{g(x,y,t)}$ used by counting framework as shown in Figure 1. Two consecutive frames $f_{(x,y,t)}$ and $f_{(x,y,t+1)}$ are applied to foreground extraction block. First, we compute Horn and Schunk (HS from now on) optical flow between adjacent frames, then Median filter and Gaussian filter are used to remove noises. We then set a threshold to get foreground mask $f_{hs(x,y,t)}$. In the same way, Gaussian Background Subtraction (GBS from now on) is used to get another foreground mask $f_{g(x,y,t)}$, after applying scale filter. Usually crowded objects move in wide areas, and for crowd flow segmentation, we need to detect change in every pixel, so optical flow methods reported in literature to compute sparse optical flow using the interest points (Lucas-Kanade optical flow) [LK$^+$81] or dense optical flow for all pixels (HS optical flow) [HS81] in each frame can be used. Since, we want to detect change in every pixel, we compute dense optical flow. Since the optical flow vector of each pixel has the magnitude and direction values, we use magnitude information to extract foreground, all the pixels which have higher magnitude than T_{th} will be classified as foreground. Direction information of optical flow vectors can be used in crowd flow segmentation by clustering all optical flow vectors having similar orientations. Such methods are usually prone to errors due to unpredictable behavior of the pixels which change due to fast/slow moving objects and illumination. A small change in illumination can be detected as foreground objects even in the static background. Such methods can be useful in extracting region of interest (ROI) in the scene but can not be used in separating individuals in high density scenarios. As shown in Figure 2, $f_{hs(x,y,t)}$ can not provide information about the group of foreground pixels (blobs) related to the people in the crowd. Therefore, for counting framework, we generate another foreground mask $f_{g(x,y,t)}$ by Gaussian background subtraction method. GBS is a kind of background subtraction method [SG99] and is very good in separating objects from the background. GBS method is effective in suppressing

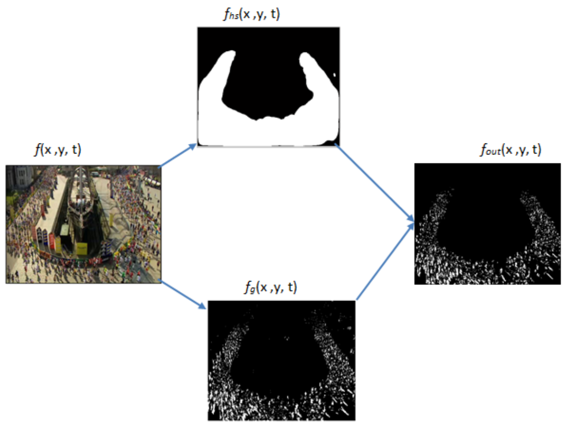


Figure 2: Foreground extraction framework

noise and robust to change in illumination. $f_{g(x,y,t)}$ is also a binary image, where blobs represents the objects of different sizes. Small blobs are related to parts of object, medium blobs related to objects and large blobs represent group of objects, appeared due to occlusions. Optimal foreground mask $f_{out(x,y,t)}$ is obtained by logical product of $f_{g(x,y,t)}$ and $f_{hs(x,y,t)}$. Later on, we apply morphological processes like morphological opening and closing on the binary image $f_{out(x,y,t)}$. The morphological open operation is erosion followed by dilation, eliminates smooth contours and protrusions. While morphological close is dilation followed by erosion, smooths the section of contours, eliminates small holes and fills gaps in contours. These operations are dual to each other. Segmentation block segments the crowd flows into different clusters, $C'_{j(x,y,t)}$, by employing K-means clustering followed by blob absorption method. To estimate the number of people in each flow segment, we take logical product of each cluster $C'_{j(x,y,t)}$ and foreground mask $f_{out(x,y,t)}$ and count the number of people by blob analysis and blob size optimization methods.

3.2 Motion Flow Field Computation

After foreground extraction, the objects in the foreground move in different directions as shown in first row of Figure 3. It can be seen that in each video, foreground objects have multiple flows. Since we use dense HS optical flow that computes movement of every pixel, we call it motion flow field. The motion flow field is a set of independent flow vectors in each frame and each flow vector is associated with its respective spatial location. This instantaneous motion field of the video contains temporal information and can be used for the learning motion pattern of the video. Consider a feature point i in F_t, its flow vector Z_i includes its location $X_i = (x_i, y_i)$ and its velocity vector $V_i = (v_{x_i}, v_{y_i})$, i. e. $Z_i = (X_i, V_i)$ where θ_i is the angle or direction of V_i, where $0° \leq \theta \leq 360°$.Then $\{Z_1, Z_2, \ldots, Z_k\}$ is the motion flow field of all the foreground points of an image.

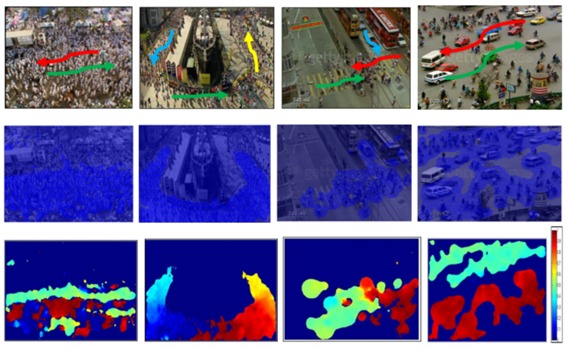


Figure 3: First Row: sample frames from videos of the Hajj, a marathon, pedestrian crossing, and road section; second row: corresponding optical flow; third row: corresponding direction map

3.2.1 Motion Flow Field Segmentation

The motion flow field $\{Z_1, Z_2, \ldots, Z_n\}$ is a n x 4 matrix where each row represents flow vector i and columns represents its spatial location X_i and velocity vector V_i. n represents total number of flow vectors (foreground points). Each flow vector represents motion in specific direction as shown in Figure 3, third row. Figure 3, (third row) does not show dominant motion patterns, so we can not infer any meaningful information about flows. Therefore, we need a method that automatically analyses the similarity among the flow vectors and cluster them in multiple groups. We use K-means clustering algorithm(widely used in data analysis and image segmentation) to segment motion flow field into different groups. This process of grouping vectors that represent specific motion pattern is called segmentation. After segmentation process, motion field is divided into multiple segments. We denote K as the initial number of cluster centroids. Commonly used initialization methods are Forgy and Random Partition [HE02]. We initialize cluster centroids as $(K-1)$ x $360°/K$. let C = $\{1, 2, ..j\}$ is the set of initial cluster centroids. ε = 360° /2K and δ = 360°/K.

```
Step 1 Clustering with initial K-centroids
  for $1 \leq i \leq n$ do
    for $1 \leq j \leq K$ do
      if $\| \theta_i - c_j \| \leq \varepsilon$ then,where $c_j \in C$
        $z_i(x_i, v_i) \rightarrow c_j$
        $n_j \leftarrow n_j + 1$
      end if
    end for
  end for

Step 2 New centroids calculation
  for $1 \leq j \leq K$ do
    $c'_j = \sum_{i=1}^{n_j} \theta_i / n_j$, Update $C$ with new centroids $c'_j$
  end for

Step 3 Clustering of similar clusters
  if $\| c'_l - c'_m \| \leq \delta$ then
```

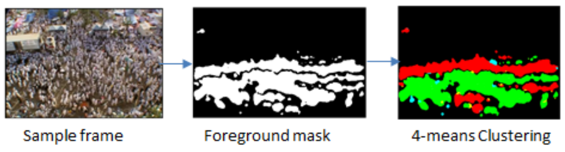


Figure 4: Results of 4-means clustering in a Hajj video frame

```
        $c'_l = \sum_{i=1}^{n_l+n_m} \theta_i / n_l + n_m$
        $c'_m \leftarrow c'_l$
    end if
```

Step 4 Return to step 1

This approach can be applied to the images where the objects moves in every direction. For such kind of complex movements in images, we assign larger value of K while we assign lower value to the images where objects move in regular directions. In this paper, we assign lower value of $K = 4$ because in our benchmark videos, objects move in regular directions. Figure 4, shows that the objects in sample frame are clustered into different groups by applying 4-means clustering. We use different colors to differentiate clusters. Let C = $\{1, 2, ..., K\}$ is the set of clusters found in sample frame.

3.2.2 Blob Absorption

We noticed that after K-means clustering, some small blobs appear: these small blobs represent small clusters as shown in Figure 4 and resulted due to following reasons. First, if the objects move slowly, the inside and outside flow vectors of the objects are not same and as a result are classified into two different flows. Second, if the two opposite optical flow intersect, the optical flow at the boundaries is ambiguous. Third, small blobs represents small groups of people and are not the part of dominant motion flows and they are not relevant to the aims of our analysis. Therefore, we adopt blob absorption approach (mimicking a "big fish eats small fish" process), where these blobs are either absorbed by dominant cluster or by the background. The algorithm is as follows:

1. Compute weights for all clusters, i.e. $C_{wj} = \sum_{j=1}^{K} n_j \; / \; T$. where n_j is number of features points $z(x, v)$ in cluster C_j and T is total number of foreground points.

2. Select cluster C_j and perform blob analysis and find area of each blob in C_j.

3. Use threshold area L and find blobs whose area $A \leq L$. Let $B = \{b_1, b_2, ...b_n\}$ set of blobs represents small clusters and needs to be absorbed.

4. Select blob b_i from set B, find its edges points by using canny edge detector [Can86].

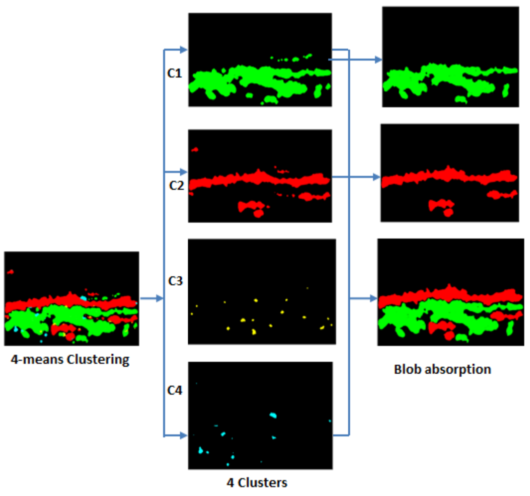


Figure 5: Results of the Blob Absorption method applied to a frame of the Hajj video

5. For each edge point, look at its neighborhood points, find neighborhood cluster ids and store ids of neighborhood points in array S. Remove those points from S that have same cluster id j, because b_i can not be absorbed by itself.

6. From remaining points in S, compute blob weight $b_{wi} = \sum_{j=1}^{N} n_j \; / \; T_s$. where N is the total number of cluster ids found in S. n_j is total number of points with cluster id j and T_s is total number of points in S.

7. Compute w_t= c_{wi} + b_{wi} and cluster id j with maximum weight wt is selected and id j is assigned to all points of blob b_j.Hence blob is absorbed.

8. Repeat steps 4 to 7 until B is empty.

9. Repeat step 2. Here background is also considered as cluster with id and cluster weight $c_w = 0$.

After blob absorption, as shown in Figure 5, small clusters (C_3 and C_4) are removed leaving behind large clusters (C_1 and C_2) representing dominant flows with clear boundaries, by setting up threshold area $L = 500$. Let C'=$\{1, 2, ..j\}$ is set of large clusters.

3.3 Counting People in High Density Crowds

This section describes the methodology for counting people in high density crowds. In this step, we count the number of people in each cluster C'_j. In low density crowds, due to clear visibility of individual with little occlusions, we can detect, track and count the number of individuals in crowd, but in high density crowds, it is hard to extract and count the individuals due to (i) with increasing density, the number of pixels/individual decreases (ii) severe occlusions result in the loss of observation of the target individual (iii) discerning individuals from one another is caused by constant interaction among individuals in a crowd. Therefore, as a solution, we perform blob analysis and blob size optimization techniques on foreground image and estimate the number of people in high density crowds.

3.3.1 Blob Analysis and Blob Size Optimization

For extracting foreground, belonging to each dominant flow (or cluster C'_j), we take logical conjunction of each cluster C'_j and foreground mask $f_{g(x,y,t)}$, generated by Gaussian background subtraction and shown in Figure 7. First row of Figure 7, shows that sample frame of marathon video is segmented into three dominant flows while second row shows foreground elements belonging to each of three segments. After foreground extraction, small blobs appear which represent moving objects. Blobs are the connected regions of variables “areas” in the binary image. Since there are many blobs of different areas representing different moving objects we need to find an optimal area that will serve as a threshold. The blob with areas above this threshold will not be taken into account (for instance, when counting pedestrians in road videos, these large blobs might be related to cars). For computing threshold area we devised blob size optimization algorithm discussed below.

1. Select the blob’s size randomly.lets blob’s size is A.
2. c_i = blobAnalysis(A); will return count of blobs whose size $\leq A$ for frame i.
3. $error_j$ = $\| c_i - gth_i \|$. where gth_i is the ground truth count for frame i.
4. Vary the blob size A by some constant k and repeat step 2 to 4 for N iterations.
5. Select blob’s size A for which $error_j$ is minimum.

Note that for finding optimum blob size, we used only four or five initial frames whose ground truth is available. These frames are selected randomly. For each initial frame we compute optimum blob size by using the method discussed above. We take the mean A' of all four or five optimum sizes computed for each initial frame and use A' for counting people in rest of frames. Average and standard deviation of the error between people count using the blob area and the actual number of people (Ground Truth) is plotted in Figure 6 versus blob area. In Figure 6, mean and standard deviation of the counting error is plotted for a road video. It can be seen from the figure that the error is minimum for the blob area 17, resulting therefore context dependent. It must be stressed that the optimal blob size depends on the video, especially on the point of vantage determining the size in pixels of people to be counted (in other

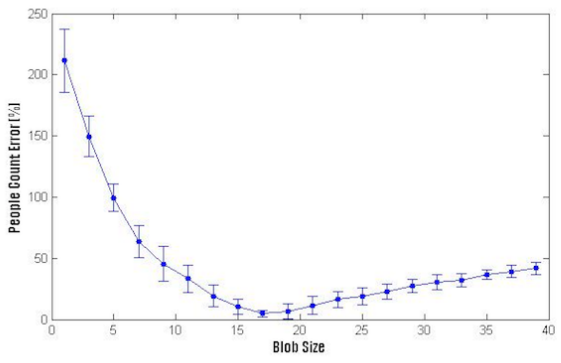


Figure 6: Blob size optimization for Road video: notice that the optimal blob size for error minimization is different for different videos.

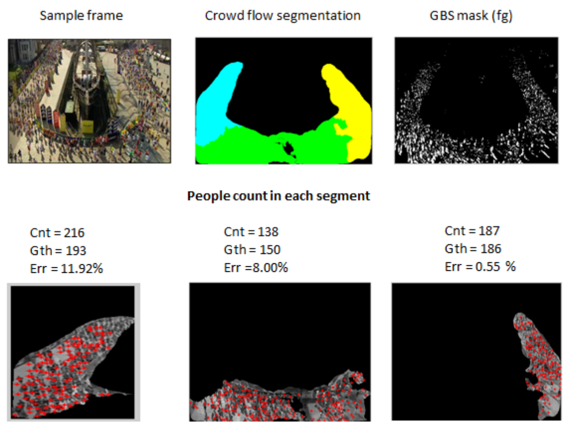


Figure 7: People Counting Framework highlighting results of intermediate steps in one frame of the marathon video

videos analysed in Sect. 4 the optimal blob size is as small as 2 pixels). Through experiments, we observed that for small blob areas, the count of people will be higher as the noise will also be counted as people. For large blob areas, instead, some people might be missed in the count. Hence selection of optimal blob size is very important to minimize the error in people count.

4 EXPERIMENTAL RESULTS

This section presents the quantitative analysis of the results obtained from experiments. We carried out our experiments on a PC of 2.6 GHz (Core i5) with 4.0 GB memory and data set from UCF [AS07]. The data set covers two types of crowded scenarios: the first scenario consists of videos involving high density crowds i.e. videos from Hajj and a marathon, where the number of people is higher than 150 in a single frame. The second scenario covers low density crowds where the number of people in a frame is lower than 70, i.e. road crossing video, where people are moving over zebra crossing in different directions, and road video, where vehicles and people are moving in different directions on road. Since our framework consists of two major

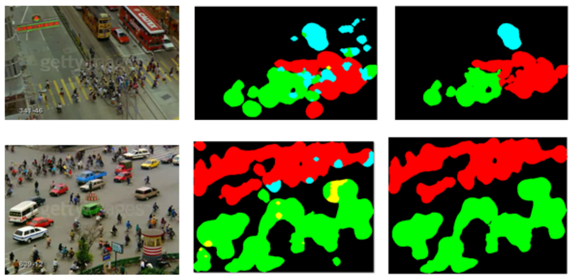


Figure 8: First column: sample frames; Second Column: K-means clustering results; Third Column: Blob absorption results

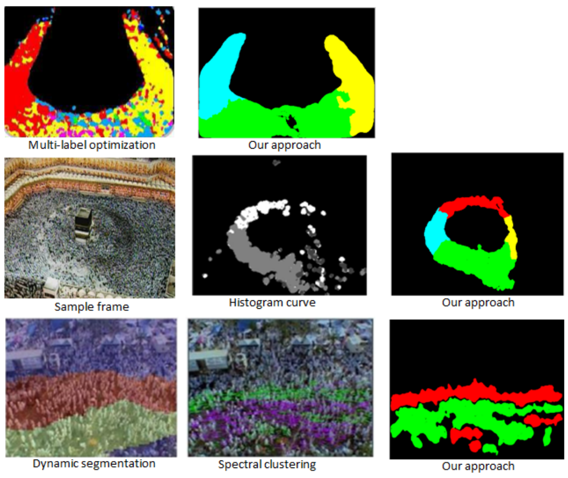


Figure 9: Comparing Results

Table 1: Hajj Video people counting in sequence of frames

F.n.	G.T.(E)	G.T(W)	Cnt.(E)	Cnt.(W)	Err(E)	Err(W)
12	151	159	170	154	12,58%	3,14%
20	153	161	167	154	9,15%	4,35%
29	185	185	195	194	5,41%	4,86%
37	176	187	192	201	9,09%	7,49%
45	187	186	200	191	6,95%	2,69%
55	187	187	195	188	4,28%	0,53%
63	189	185	194	194	2,65%	4,86%
		Average Error			**7,16%**	**3,99%**

Table 2: Crossing video people counting in sequence of frames

F.n.	G.T.(E)	G.T.(W)	Cnt.(E)	Cnt.(W)	Err(E)	Err(W)
10	30	30	30	29	0,00%	3,33%
16	34	35	30	39	11,76%	11,43%
22	37	36	25	38	32,43%	5,56%
28	35	33	29	32	17,14%	3,03%
30	38	35	37	43	2,63%	22,86%
35	38	34	35	41	7,89%	20,59%
40	37	36	36	39	2,70%	8,33%
47	35	36	35	30	0,00%	16,67%
55	37	38	38	34	2,70%	10,53%
64	37	40	31	28	16,22%	30,00%
		Average Error			**9,35%**	**13,23%**

Table 3: Road Video people counting in sequence of frames

F.n.	G.T.(E)	G.T.(W)	Cnt.(E)	Cnt.(W)	Err(E)	Err(W)
11	45	67	33	44	26,67%	34,33%
20	38	65	45	58	18,42%	10,77%
30	42	62	46	69	9,52%	11,29%
35	41	61	40	62	2,44%	1,64%
43	39	64	36	53	7,69%	17,19%
50	40	65	48	67	20,00%	3,08%
55	40	65	36	55	10,00%	15,38%
62	39	63	39	67	0,00%	6,35%
		Average Error			**11,84%**	**12,50%**

parts,crowd flow segmentation and crowd counting, our experiments are carried out in two steps.

4.1 Segmentation Results

We selected 65 frames from each video. After computing optical flow, we apply K-means clustering algorithm that cluster all the similar flow vectors.In this paper, we use $K = 4$ for all the videos,so after segmentation, we detect four different flows in video frame as shown in second column of Figure 8. We then apply blob absorption method to remove small clusters as shown in third coumn of Figure 8. For blob absorption we use different threshold L values. Small clusters can not be aborbed completely by using smaller values of L while we lost some portions of dominant cluster by using larger values of L. Therefore, we determined value of L experimentally and is different for different videos. After blob absorption, image of cross video is segmented into three flows, red(west),green(east) and cyan(south). While image of road video is segmented into two flows, red and green as shown in third column of Figure 8.

We compared our approach in Figure 9 with multi-label optimization [UC12], histogram curve [LRZ12], dynamic segmentation [AS07] and spectral clustering [EB08]. In the first row of Figure 9, we compare our method with multi-label optimization method. We see that crowd flow segmentation using multi-label optimization could not segment the crowd into dominant flows. Moreover, it could not find clear boundary due to small blobs appeared after segmentation. In the second row of Figure 9, we compare our results with histogram curve method. Segmentation by using histogram curve is fastest than existing methods but it lost much information about the crowd flows, since this method only looks to the peaks of histogram curves. In the third row of Figure 9, we compare our results with dynamic segmentation and spectral clustering approach. Dynamic segmentation is not able to detect small flows in the crowd, while spectral clustering carries out segmentation on sparse optical flow and give the approximate segmentation where we can not find clear boundaries between flows. All the above shortcomings are resloved by our proposed approach. Our proposed approach not only detects dominant flows but can also detects small flows without the loss of crowd flow information. Moreover, our proposed approach finds clear boundaries among different flows.

4.2 Crowd Counting Results

After crowd flow segmentation, we count the number of people in each flow segment. Each video consists

Table 4: Marathon Video people counting in sequence of frames

F.n.	**G.T.(E)**	**G.T.(N)**	**G.T.(S)**	**Cnt.(E)**	**Cnt.(N)**	**Cnt.(S)**	**Err(E)**	**Err(N)**	**Err(S)**
11	145	192	187	134	176	199	7,59%	8,33%	6,42%
15	150	186	193	138	187	216	8,00%	0,54%	11,92%
20	148	193	200	126	178	190	14,86%	7,77%	5,00%
27	155	200	211	151	244	225	2,58%	22,00%	6,64%
33	150	195	220	145	223	219	3,33%	14,36%	0,45%
39	160	205	210	151	199	222	5,63%	2,93%	5,71%
45	158	210	205	145	215	210	8,23%	2,38%	2,44%
49	156	207	210	145	189	197	7,05%	8,70%	6,19%
55	162	215	215	164	210	196	1,23%	2,33%	8,84%
59	158	220	220	162	210	202	2,53%	4,55%	8,18%
62	167	225	224	158	185	198	5,39%	17,78%	11,61%
Average Error							**6,04%**	**8,33%**	**6,67**

of sequence of 65 frames and our proposed method automatically counts the number of people in each frame as shown in Tables 1, 2, 3, 4. Tables show counting results of random frames taken from each analysed video, where F.n. represents frame number of the analysed sequence. The rise and fall in people count in different frames represents the fact that people are entering or leaving the scene affecting people count at different time. To check the counting accuracy of the proposed framework, ground truth (G.T) for each direction (East(E), West(W), North(N), South(S)) is found for the frames after random intervals and count error (Err) is computed by comparing results with the ground truth data. Count error is shown in details in tables 1, 4, 2, 3 for all analyzed video sequences. The first column of each table shows the frame number, G.T. shows grouth truth found for each direction and Cnt. is counting results of our proposed approach. Average error is less than 12% for all analyzed video sequences. For some frames, however, count error is higher due to the fact that some people in that frame are missed in count or noise (resulted after motion segmentation) is counted as people. As obvious from tables, our proposed framework works better in high density scenarios like Hajj and marathon. It is matter of the fact, that in high density scenarios, people covers much of the scene's area in comparison to low density scenarios. After motion segmentation, foreground extracted in high density scenarios contains less background noise (foreground noise generally moves with people and it is not causing significant errors) in comparison to foreground extracted in low density scenarios. From the experimental results, it is clear that our proposed approach count the people in each video sequence with 90% accuracy.

To study the time complexity of our proposed framework, we utilize 65 frames of each of four analysed videos and time is recorded as average frame processing time and recorded in Table 5. The latter shows time complexity of crowd flow segmentation and crowd counting frameworks. Rows of table shows the analysed videos and column represents time complexity of each of processing block. It is obvious that clustering takes much time as compare to blob absorption method and crowd counting framwork. It is matter of the fact that K means clustering is computationally expensive and can be very slow to converge in worst case scenarios, i.e. high resolution videos, and high ratio of foreground to background pixels. In this paper, we use videos of the same resolution, 360x480. Although the resolution of all analysed videos is same, yet time complexity is different. The ratio of foreground to background pixels of different videos is different and usually the ratio is higher if the large part of the scene is covered by foreground pixels. It is also obvious from table that Hajj video takes more computational time than other videos. It is matter of the fact that most of scene of a Hajj video frame is covered by foreground pixels than background pixels. The computational time can be reduced and proposed framework can be employed in real time, if implemented in openCV. The current implementation is in Matlab.

5 CONCLUSIONS

In this paper, we have considered both high and low density crowds and proposed a framework that automatically detects dominant motion flows and counts the number of people in each flow. Such kind of analysis provides a useful input to pedestrian simulation models. A first employment of the our analysis is related to the actual initial configuration of the simulation scenario. Second way to exploit data resulting from automated video analysis is represented by pedestrian counting and density estimation: the indication of the average number of pedestrians present in the simulated portion of the environment is important in configuring the start areas. Finally, we can use the above analysis in the validation of simulation results. Our approach is applicable in many different situations and it is independent of local conditions and camera viewpoints. Our method does not require detection and tracking of people, hence preserving the privacy of the people. Future works are aimed at improving the precision, especially in low density situations, but also aim at using these techniques to more comprehensively characterize the movement patterns in the analyzed frame by identifying and quantitatively describe points of entrance of pedestrians, points of interest and exits, essentially to derive a so called origin-destination matrix.

Table 5: Time Complexity of our proposed framework in (seconds)

Videos	Crowd Flow Segmentation		Crowd Counting		
	Clustering	Blob Absorption	Seg # 1	Seg # 2	Seg # 3
Marathon	6	2.77	0.006	0.007	0.005
Hajj	9.88	2.93	0.009	0.008	NIL
Road	7.02	1.67	0.005	0.004	NIL
Crossing	5.12	1.03	0.003	0.005	NIL

6 REFERENCES

[AS07] Saad Ali and Mubarak Shah. A lagrangian particle dynamics approach for crowd flow segmentation and stability analysis. In *CVPR*. IEEE Computer Society, 2007.

[Can86] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[EB08] Günther Eibl and N Brandle. Evaluation of clustering methods for finding dominant optical flow fields in crowded scenes. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.

[HE02] Greg Hamerly and Charles Elkan. Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 600–607. ACM, 2002.

[HMY$^+$97] Kazuhiko Hashimoto, Katsuya Morinaka, Nobuyuki Yoshiike, Chjihiro Kawaguchi, and Satoshi Matsueda. People count system using multi-sensing application. In *Solid State Sensors and Actuators, 1997. TRANSDUCERS'97 Chicago., 1997 International Conference on*, volume 2, pages 1291–1294. IEEE, 1997.

[HS81] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1):185–203, 1981.

[LCC01] Sheng-Fuu Lin, Jaw-Yeh Chen, and Hung-Xin Chao. Estimation of number of people in crowded scenes using perspective transformation. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 31(6):645–654, 2001.

[LK$^+$81] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[LRZ12] Wei Li, Jiu-Hong Ruan, and Hua-An Zhao. Crowd movement segmentation using velocity field histogram curve. In *Wavelet Analysis and Pattern Recognition (ICWAPR), 2012 International Conference on*, pages 191–195. IEEE, 2012.

[LWMZ10] Wei Li, Xiaojuan Wu, K Matsumoto, and Hua-An Zhao. Crowd foreground detection and density estimation based on moment. In *Wavelet Analysis and Pattern Recognition (ICWAPR), 2010 International Conference on*, pages 130–135. IEEE, 2010.

[MG01] Thomas B Moeslund and Erik Granum. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding*, 81(3):231–268, 2001.

[MHL08] Wenhua Ma, Lei Huang, and Changping Liu. Advanced local binary pattern descriptors for crowd estimation. In *Computational Intelligence and Industrial Application, 2008. PACIIA'08. Pacific-Asia Workshop on*, volume 2, pages 958–962. IEEE, 2008.

[OYA10] Ovgu Ozturk, Toshihiko Yamasaki, and Kiyoharu Aizawa. Detecting dominant motion flows in unstructured/structured crowd scenes. In *ICPR*, pages 3533–3536. IEEE, 2010.

[RMAS04] Pini Reisman, Ofer Mano, Shai Avidan, and Amnon Shashua. Crowd detection in video sequences. In *Intelligent Vehicles Symposium, 2004 IEEE*, pages 66–71. IEEE, 2004.

[RP07] Damian Roqueiro and Valery A Petrushin. Counting people using video cameras. *The International Journal of Parallel, Emergent and Distributed Systems*, 22(3):193–209, 2007.

[SG99] Chris Stauffer and W Eric L Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. IEEE, 1999.

[SND11] Satyam Srivastava, Ka Ki Ng, and Edward J. Delp. Crowd flow estimation using multiple visual features for scenes with changing crowd densities. In *AVSS*, pages 60–65. IEEE Computer Society, 2011.

[TS07] Thiago Teixeira and Andreas Savvides. Lightweight people counting and localizing in indoor spaces using camera sensor nodes. In *Distributed Smart Cameras, 2007. ICDSC'07. First ACM/IEEE International Conference on*, pages 36–43. IEEE, 2007.

[TYOY99] K Terada, D Yoshida, Shunichiro Oe, and J Yamaguchi. A method of counting the passing people by using the stereo images. In *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, volume 2, pages 338–342. IEEE, 1999.

[UC12] Habib Ullah and Nicola Conci. Crowd motion segmentation and anomaly detection via multi-label optimization. In *ICPR workshop on Pattern Recognition and Crowd Analysis*, 2012.

[VYD$^+$93] SA Velastin, JH Yin, AC Davies, MA Vicencio-Silva, RE Allsop, and A Penn. Analysis of crowd movements and densities in built-up environments using image processing. In *Image Processing for Transport Applications, IEE Colloquium on*, pages 8–1. IET, 1993.

[VYD$^+$94] SA Velastin, JH Yin, AC Davies, MA Vicencio-Silva, RE Allsop, and A Penn. Automated measurement of crowd density and motion using image processing. In *Road Traffic Monitoring and Control, 1994., Seventh International Conference on*, pages 127–132. IET, 1994.

[XLH06] Li Xiaohua, Shen Lansun, and Li Huanqin. Estimation of crowd density based on wavelet and support vector machine. *Transactions of the Institute of Measurement and Control*, 28(3):299–308, 2006.

[YST10] Satoshi Yoshinaga, Atsushi Shimada, and Rin-ichiro Taniguchi. Real-time people counting using blob descriptor. *Procedia-Social and Behavioral Sciences*, 2(1):143–152, 2010.

[ZN03] Tao Zhao and Ramakant Nevatia. Bayesian human segmentation in crowded situations. In *CVPR (2)*, pages 459–466. IEEE Computer Society, 2003.

Efficient Procedural Generation of Forests

Julian Kenwood
University of Cape Town
nomad010@gmail.com

James Gain
University of Cape Town
jgain@cs.uct.ac.za

Patrick Marais
University of Cape Town
patrick@cs.uct.ac.za

ABSTRACT

Forested landscapes are an important component of many large virtual environments in games and film. In order to reduce modelling time, procedural methods are often used. Unfortunately, procedural tree generation tends to be slow and resource-intensive for large forests.

The main contribution of this paper is the development of an efficient procedural generation system for the creation of large forests. Our system uses L-systems, a grammar-based procedural technique, to generate each tree. We algorithmically modify L-system tree grammars to intelligently use an instance cache for tree branches. Our instancing approach not only makes efficient use of memory but also reduces the visual repetition artifacts which can arise due to the granularity of the instances. Instances can represent a range of structures, from a single branch to multiple branches or even an entire tree.

Our system improves the speed and memory requirements for forest generation by 3–4 orders of magnitude over naïve methods: we generate over $1,000,000$ trees in 4.5 seconds, while using only 350MB of memory.

Keywords

procedural tree generation, L-systems, instancing

1 INTRODUCTION

When large forests are used in CGI they are often created using procedural methods, due to their inherent geometric complexity. Unfortunately, the memory requirements of a procedural approach can be prohibitive. For example, some tree generation methods require as much as 10MB per tree [1]. Using such schemes, even a relatively small forest of $1,000$ trees would require much more memory than most commodity computer systems support. In addition to large memory requirements, procedurally creating a large forest from scratch could take minutes or even hours. Forests are thus usually created in an off-line process, which limits their use in games and interactive media.

We explore the problem of procedurally generating complete forests, focussing on algorithms and optimisations that facilitate the creation of very large forests, in the range of $10,000$ trees or more, in a few seconds.

We propose a new system for generating large numbers of trees with a fixed memory budget. We use L-systems to generate trees and introduce an optimisation to the L-system grammar that enables efficient caching of tree sub-branches. This allows the creation of subsequent trees to be accelerated, whilst also saving memory.

The focus of our work is not on the rendering of realistic trees, but rather on the often expensive procedural methods that underpin such systems. Consequently we illustrate our optimizations on basic branching tree structures and make no use of billboards, complex texturing and so on.

Our optimisations allow the generation of very large forests in a few seconds and with low memory overhead. This work is applicable to a broad range of L-systems and can thus supplement systems which currently make use of such a procedural approach.

The remainder of this paper is laid out as follows. Section 2 presents relevant background. Basic L-system formalism is introduced in Section 3. The optimisations that we have developed are presented in Section 4 and Section 5. The creation of tree geometry is discussed in Section 6. Section 7 presents our results and discusses the performance of the system. Finally, Section 8 summarises our findings and contributions and provides suggestions for future work.

2 RELATED WORK

EcoSys [1] represents one of the earliest and best-known procedural tree generation systems. EcoSys is able to generate realistic looking forests, including plants and other foliage, from a relatively small amount of input, such as a heightmap of the landscape. The system allows for interactive editing of the parameters with built-in editors. Each individual plant is procedurally created using an L-system. L-systems provide a set of match-replace rules that specify the appearance of the tree. These rules control everything from the texture and colour of the tree and its leaves to how the tree branches and how branches lengthen. L-systems allow modellers to present a set of rules that describe a particular species of tree.

Unfortunately, the amount of data generated for a single tree in EcoSys can be as much as 10MB. A small forest of $1,000$ trees results in a total of 10GB of data which cannot be rendered efficiently using current technology.

In order to reduce the amount of memory required the system uses *instancing* to generate a single tree that can be used in multiple places, which saves memory but can reduce realism. EcoSys only uses instances in cases where the resulting trees are likely to be similar.

EcoSys is able to render forests interactively using points and lines, but cannot achieve high enough visual quality and frame rates for games, even when executed on modern hardware [2].

The trees that are presented in modern games are usually made with a third-party library called *SpeedTree*[1]. SpeedTree, however, is proprietary software and companies have to pay a license fee to use it. SpeedTree generates trees using an offline process: either procedural generation or manual generation by an artist.

As with EcoSys, large forests are accommodated with instancing. The trees in the forest originate from a considerably smaller 'hero' tree set. Unfortunately, since the library of trees that an artist works with can be small, the same tree could be repeated in an unrealistic fashion, particularly when the game is intended for a very resource-limited platform. Recently, however, SpeedTree has added a WorldBuilder module which is able to export tree positions that exhibit fewer jarring repetition artifacts.

In this paper, we explore an alternative method for creating large forests. Similarly to EcoSys and SpeedTree, we use procedural methods to generate the individual trees in the forest. Unlike EcoSys and SpeedTree, we aim to use instancing to reduce the size of the forest *without* resorting to instancing entire trees. Our primary aim is to decrease the memory requirements for trees in forests without sacrificing visual quality.

3 L-SYSTEMS

Lindenmayer Systems, or L-systems, are used extensively in Procedural Graphics [5]. The rules for these systems are capable of describing complex structures such as plants [6] and buildings yet are simple enough to be created by modellers [4]. The simplest type of L-system — deterministic context-free L-systems (also called DOL-systems [7]) are simple match-replace rules that occur over a string of symbols. Each symbol has a specific meaning used later in the tree creation process. For instance, the symbol 'F' means to draw a cylinder at the current position, while a '+' symbol changes the orientation of the next cylinder.

$$\underbrace{O}_{Strict\ predecessor} \rightarrow \underbrace{F\ F\ F\ F\ F\ F}_{Derivation}$$

The strict predecessor is a single symbol that should be transformed into a (possibly empty) sequence of symbols called the derivation. All rules in the L-system are applied simultaneously to the entire string of symbols. The number of times the rules are applied is called the generation of the string, corresponding to the required age of the output tree. The initial string of symbols, also called the axiom, is denoted by generation 0.

The symbols from the final string are used as drawing instructions for a turtle-like graphics module called the interpreter. Symbols with no meaning are simply discarded.

There are several drawbacks to DOL-systems in the context of tree creation. Most importantly, the output for a given generation of an L-system is always the same. This means that the only way to add variation is to create scaled copies of each tree type. This limitation can be addressed by using stochastic L-systems [3], which allow *multiple* derivations with associated probabilities that indicate the likelihood of selection.

A second drawback to DOL-systems is the difficulty of growing branches of a desired length: each tree is made of individual cylinders of equal length. This can be solved by introducing parameters: each symbol in the string can have additional parameters, which can exactly model the desired length, width, and other attributes.

Researchers have also found it useful to modify the DOL-system to add two symbols, [and], to assist in creating trees efficiently by controlling a stack of saved turtles; pushing and popping onto the stack, respectively.

Before moving on, we need to introduce the concept of a *module*. A module is a special symbol in the L-system that can used to achieve higher order functionality: it effectively represents a callout to a 'subroutine'. Modules can be distinguished from regular L-system symbols by the inclusion of parentheses, which surround 0 or more parameters. Our modifications make extensive use of modules.

4 BRANCH OPTIMISATION

Branching in trees is a crucial aspect of realistic growth. Unfortunately, L-system branches generate a significant amount of geometry. This problem is exacerbated

[1] http://www.speedtree.com

when creating large numbers of trees. Our system, like EcoSys and SpeedTree, attempts to solve this problem through the application of *instancing*. However, unlike these methods, we choose to perform instancing at the granularity of branches. Such fine-grained instancing allows branches to be shared across multiple trees, whilst saving memory and keeping some degree of visual differentiation.

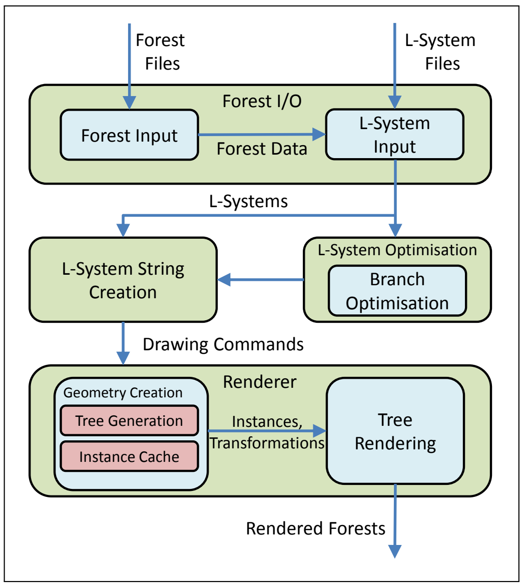


Figure 1: System structure - input L-systems are taken in together with forest generation parameters. The L-systems are then optimised to make use of instancing and geometry is generated to represent the various instanced structures. This information is then passed off to the renderer, along with positioning.

In the context of tree L-systems, branches represent additional *recursive* work that must be performed. With the help of instancing, we essentially memoise the result of this work so it can be re-used later. The basic architecture of our system is presented in Figure 1.

We use the stochastic nature of L-systems to decide which branches should be instanced and which created. A fixed *instancing probability*, P, is used to control instancing. This is a percentage represented by an *integral* number between 0 and 100, where 0 indicates no instancing and 100 indicates full instancing. We modify the L-system rules to reflect this probability. However, rules that are responsible for branching must first be detected.

4.1 Rule Detection

Each rule is examined to determine if it contributes branches to the tree. It is difficult to determine exactly which rules branch, so a heuristic is used instead. The bracket symbols, [and], are a common indicator of branching because they isolate state changes. The left bracket saves state information, such as position and orientation, which is restored at the right bracket. While this behaviour is useful in branching it is also applied to create leaves, as indicated in Table 1, as well as other non-branching phenomena.

$$L \mapsto [\,\hat{}\,\hat{}\;-f+f+f-|-f+f+f\,]$$

Table 1: A rule from an L-system used to draw a leaf. Brackets are employed but no branching occurs.

The heuristic uses brackets as an indicator of branching, but to filter out erroneous cases a further restriction is applied: the brackets must contain at least one *non-terminal* symbol. Non-terminals are the strict predecessors on the left-hand side of each rule. While this may still incorrectly identify rules as branching, it is significantly more accurate than using brackets alone.

$$\begin{aligned} &X \mapsto [\,L\,] \\ &L \mapsto \hat{}\,\hat{}\;-f+f+f-|-f+f+f \end{aligned}$$

Table 2: A modified version of the L-system in Table 1. The second rule is incorrectly identified as a branching rule by our heuristic.

Segments of the rule symbols are identified as start and end points for the branch. The brackets and the symbols between them are tagged with an identifier. Branches represented by identical symbols share an identifier, the assumption being that the resultant geometry is the same. The identifiers are global in that they may be shared across different rules; they are used later to access one of several instance caches.

Algorithm 1 shows the detection process. In the pseudocode, `seenBranch` and `branchGUID` return information about the branch currently being examined. `seenBranch` returns true if it is identical to a previously seen branch. `branchGUID` returns the unique identifier of a previously seen branch. The function `addBranch` adds a branch to the global list of previously seen branches and returns its new unique identifier. `createBranch` creates a structure that packages the unique identifier and the branch symbol information for later use in the program.

4.2 Rule Modification

The rule modification process is complicated by the existence of stochastic rules.

We begin by looking at the simpler case of modification for deterministic L-systems. For such systems, each rule's right-hand side is replaced by several right-hand sides (meaning that the resulting rule becomes stochastic) depending on the number of branches that occur. If B branches are present on the right-hand side, then 2^B

```
detectBranching(rhs, nonTerminals)
output = []
for $s = 1 \rightarrow$ length(rhs)
   if rhs[s] = [ then
      $t = s + 1$
      for $t \rightarrow$ length(rhs)
         if rhs[t] = ] then break
      if t = length(rhs) then continue
      for $u = s + 1 \rightarrow t$
         if rhs[u] in nonTerminals then
            continue s
      guid = -1
      if seenBranch(s, t) then
         guid = branchGUID(s, t)
      else seenBranch(s, t) then
         guid = addBranch(s, t)
      output += [createBranch(guid, s, t)]

return output
```

Algorithm 1: Branching RHS Detection.

$$
\begin{array}{ll}
 & \mapsto F\ A \\
A & \mapsto \overbrace{[\ \&\ F\ L\ !\ A\ ]}^{\text{Branch 1}}\ /\ /\ /\ /\ / \\
 & \overbrace{[\ \hat{}\ F\ L\ !\ A\ ]}^{\text{Branch 2}}\ /\ /\ /\ /\ /\ /\ /\ \overbrace{[\ \&\ F\ L\ !\ A\ ]}^{\text{Branch 1}} \\
F & \mapsto F\ F \\
L & \mapsto [\ \hat{}\ \hat{}\ \{\ -f\ +f\ +f\ -\ |\ - \\
 & f\ +f\ +f\ \}\ ]
\end{array}
$$

Table 3: A leafy tree L-system [5]. The A-rule has been annotated with information that marks the segments that branch. Each branch is tagged with a number that identifies the branch segment. Note that the third branch has the same identifier as the first branch due to having identical symbols.

new right-hand sides are created, representing the possibility of either instancing each branch or not.

If a branch is to be instanced, the relevant symbols are replaced by a *getInstance* module. Otherwise, other control modules, *startInstance* and *stopInstance* are inserted instead. These two modules demarcate segments of a string that correspond to branch information. Each takes two parameters: an *identifier* and an *age*. The identifier is the same as the one associated with the branch in the rule detection phase. The age is determined from the *getGeneration* function, which returns the generation at which the module was created.

Each right-hand side is given a probability, p, based on the instancing probability and the number of branches being instanced, calculated as follows:

$$p(I) = P^I \times (1-P)^{B-I}$$

where P is the probability of replacing a branch with an instance, B is the total number of branches and I is an index variable. For each right-hand side, the index variable is the number of times that the decision is made to instance a particular branch.

A simple binary number counting algorithm is used to enumerate these rules that is both efficient and easy to implement. It is only suitable if the number of branches in a rule is less than the bit-length of a machine's word size. In practice this constraint is not at all problematic. The disadvantages of this optimisation are evident from inspecting the output: the number of right-hand sides has greatly increased and each is significantly less humanly readable.

Stochastic L-systems add complexity in that the several right-hand sides may create branches. The above algorithm is performed on each original right-hand that contains branching segments. The relative probabilities of each group of newly created right-hand sides must reflect the original distribution. To enforce this, the equation for p is modified:

$$p(I) = P_{original} \times P^I \times (1-P)^{B-I}$$

where $P_{original}$ is the probability of the originating rule. Multiplying by the original probability ensures that the probabilities have the correct distribution.

The time required to apply this optimisation to a set of rules depends on the number of branches, B_i, and the length, L_i, of each right-hand side. The total computation and memory cost is bounded by $O(\sum 2^{B_i} L_i)$. The main source of the inefficiency of this algorithm is the number of right-hand sides that are created. In our investigations, B_i is rarely larger than three and is thus not problematic. It may also be possible to achieve the same effects using fewer right-hand sides. This is, however, left as future work.

The effectiveness of the branching optimisation depends on the instancing probability. If this is set too high, repetition will become evident. Conversely, setting it too low results in the memory required for a forest becoming too large. We use an instancing probability that varies as more trees are created to support instancing that becomes more aggressive as memory becomes scarcer.

The *getInstance*, *startInstance* and *stopInstance* symbols are used during interpretation to communicate with the higher levels of the system. *getInstance* indicates that the system should record the current position and orientation where an instance should be used to complete the tree. *startInstance* notifies the system that all the geometry created until the corresponding *stopInstance* symbol forms a coherent instance usable as part of other trees. The instance cache is the device used to store and retrieve instances and is introduced next.

5 INSTANCE CACHE

The end result of the interpretation phase is geometry in the form of vertex and index buffers, required for rendering. Using the symbols introduced for the L-system rule modification process, we annotate the index buffers as they are created. In this way, we can determine which ranges of the index buffers correspond to branches with different sizes and properties. Multiple instances can exist within the same index buffer. For example, a tree branch could have an annotated sub-branch. For this reason, in addition to pointers to the vertex and index buffers, two integers are stored to denote the range in the buffers that correspond to the current instance.

Each index buffer range also stores metadata about the range. The age, species type and branch identification information are stored in hash tables to allow for easy and efficient access. The hash table data contains arrays of pointers to these ranges, which allows for efficient random selection. The Instance Cache can thus retrieve a random index buffer range based on any set of age and species criteria.

In addition to storing the buffer range, the transformation of geometry is retained. Each transformation is a matrix that represents the spatial orientation of the geometry in the index buffer. This information is necessary to correctly place the branch on a renderable tree. Finally, the orientation and positions of any *getInstance* modules that occurred in the branch string are recorded. These *getInstance* modules are used to indicate exit points for the instance which must be filled with other instances in order to generate a complete tree.

6 TREES FROM INSTANCES

Trees are created in the system in two distinct phases: *Hero Creation* and *Tree Placement*. Hero Creation runs the L-systems in order to fill up the various instance caches that exist. We use one instance cache per unique identifier generated in the rule modification phase. These heroes serve as the template geometry for tree and branch instances. Although each hero created is a correctly derived tree, they are not used directly for rendering purposes: the creation of trees for rendering is left to the Tree Placement phase.

The cumulative size of all geometry buffers is limited in our system depending on the available memory. For example, if one were creating a virtual environment where forests are not important, the maximum size could be very low, perhaps as little as 16MB. On the other hand, for environments where forests are a prominent feature, one can devote upwards of 256MB to the required buffer. The ability to define a range of cache sizes allows developers to tightly control the resources used to generate a forest.

As more hero trees are created, the memory space that can be devoted to geometry decreases. In order to allow for the continued creation of new geometry, we *increase* the instancing probability for each hero tree that is created. A higher instancing probability means that more *getInstance* symbols will occur in the L-system strings. In other words, fewer new branches are created and more instances are used.

The final stage in forest creation is Tree Placement. Tree Placement creates new trees by cutting and joining parts of the hero trees together and calculating the necessary transformations to place the trees on the terrain.

A renderable tree is represented by a collection of pointers to geometry buffer ranges that are stored in the Instance Cache. Creating a tree from the instance cache is done recursively. The algorithm takes the species of the desired tree, its generation and a transformation matrix describing the desired position and orientation, as input. Given this information, an instance is selected from the instance cache to serve as the base of the tree.

An instance is not limited to representing a single generation. Instead, it can represent the entire tree, a single generation or, more likely, several generations with exit points that need to be filled with sub-branches. The exit points describe not only the desired position and orientation relative to the start of the instance but also the desired age and branch identification of the instance that should fill the gap.

The algorithm recurses for each exit point required by the current instance. The age and branch information parameters are used to constrain the subsequent search of the instance cache. The instances are re-oriented by computing a placement transformation matrix. Given the desired orientation matrix, D, and the orientation matrix of the instance within its hero tree, M, we calculate the transformation to reorient an instance, T, as:

$$T = D \times M^{-1}$$

The initial desired orientation is passed as a parameter to the recursive function so it must be updated before it is passed into the next function call. To update the orientation we use the following formula:

$$D_{new} = D_{old} \times E$$

where D_{new} is the new orientation, D_{old} is D from above, and E is the orientation of the next exit point in relation to its hero tree.

Algorithm 2 shows the tree creation process described above. The recursive function, `CREATE`, takes several

```
create(output, direction, age, cache)
instance = cache.getInstance(age)
T = direction $\times$ instance.direction.inverse
output.addTreeInstance(instance.buffers, T)
for $i = 1 \rightarrow$ length(instance.exitPoints)
    newAge = instance.exitPoints[i].exitAge
    exit = instance.exitPoints[i].direction
    newDirection = direction $\times$ exit
    create(output, newDirection, newAge, cache)
```

Algorithm 2: Renderable tree building process.

input arguments that describe the instance we wish to find. In the algorithm listed above, we only search the instance cache by generation; in practice we use other criteria as well. The desired orientation, `direction`, is used to compute the transformation, T, needed to correctly draw the geometry buffer. The `output` contains a list of geometry buffers that we should draw and their correcting transformations. The `addTreeInstance` method simply appends the buffers and transformations to this list. The recursive function terminates when all exit points have been filled. Although this function is recursive, it remains significantly faster than regular L-system creation as no geometry is created. Only geometry in the instance caches are used.

Rather than applying the transformations to the geometry data immediately, the transformation is stored so that the renderer can perform the transformation on the fly. This allows the geometry data to be efficiently re-used across multiple trees and branches. The index buffer range and the required transformation are saved to the tree object for use with the renderer. The end result of this process is a collection of pointers to index buffer ranges and the transformations necessary to correctly render the tree at a particular position.

7 RESULTS

Testing was done on an Intel Core i5 2.80GHz quad-core machine with 8GB of RAM and an NVidia 580GTX graphics card. To avoid interfering memory requests from other applications or processes, we limited the system to using 4GB of RAM.

We split our tests into two groups. For the first group, we kept the forest size constant, while testing the run-time of different cache schemes. The forest size was kept at $10,000$ trees and the following cache sizes were tested: 16MB, 32MB, 64MB, 128MB, 256MB, and 512MB. The second group of tests kept the cache size at a constant 128MB and created forests of up to $1,000,000$ trees, and measured both the memory and run-time of the forest creation process.

Figure 2 shows the results of creating forests with varying cache sizes. In most cases, the majority of the time is spent creating hero trees to fill the various caches.

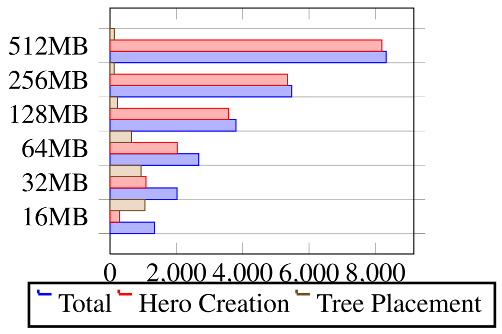


Figure 2: Time (in milliseconds) required to create $10,000$ trees, as a function of cache size. Hero tree creation accounts for the largest proportion of processing time for larger cache sizes.

From our results, hero creation time grows roughly linearly with increasing cache size. The 16MB cache takes approxmately one second to fill, while the 512MB cache takes up to eight seconds. The hero creation process requires, on average, 25 milliseconds per MB of cache.

By comparison, tree placement generally requires much less computation time, although it does scale with the number of trees being created. However, for smaller cache sizes, tree placement requires a larger proportion of total running time for a fixed number of trees. There are two reasons for this. First, a smaller cache can be filled significantly faster than a large cache. Second, each additional hero tree depletes the percentage of cache space available much more rapidly for small cache sizes. This has a knock-on effect on the instancing probability used to generate new hero tree. A hero tree created with a high instancing probability is likely to contain many instance exit points (*getInstance* symbols in the L-system string). Consequently, Algorithm 2 will require many more recursive calls on average and, thus, take longer to run. This phenomenon is not seen in the large cache sizes since the instancing probability increases much more slowly.

As the cache size increases, the time required for tree placement reduces from one second, for the 32MB cache, to 130 milliseconds for the 512MB cache. This equates to a placement time of 0.1 milliseconds per tree and 0.013 milliseconds per tree, respectively. The total time to create and place 10,000 trees ranges from 1.3 seconds for the 16 MB cache to 8.4 seconds for the 512MB cache. The cache size is an important choice that must be made by the user of the system. If the desired output is a small forest, a small cache size should be chosen and vice versa. The disadvantage to choosing larger cache sizes, however, is the significantly longer time that users must wait in order for the cache to fill.

To determine the general utility of our instancing approach, we also evaluated the running time for forest creation *without* using any instancing. Unfortunately,

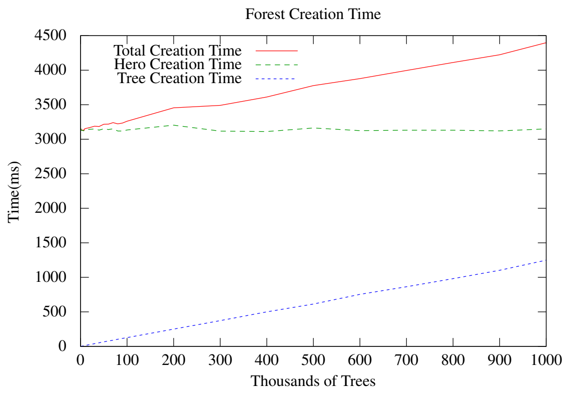


Figure 3: Time (in milliseconds) required to create forests for a cache size of 128MB. Hero tree creation accounts for most of the time, and is almost constant. Tree creation, which includes tree placement, increases linearly with the number of trees, but very slowly: creation of 1,000,000 trees requires only 1.5 seconds.

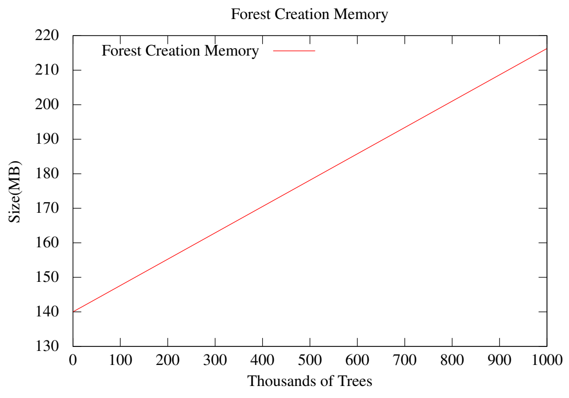


Figure 4: Memory (in megabytes) required to create a forest for a cache size of 128MB.The instance cache accounts for most of the memory usage. Each individual tree typically requires an average of only 100 bytes.

the system ran out of memory when attempting to create all 10,000 trees. The largest size that we were able to create was approximately 2,000 trees over sixty seconds. Based on our testing, we estimate that creating 10,000 trees would require at least 300 seconds to complete. It is clear that our caching system is markedly faster than the uninstanced approach.

Our next set of tests show the running time and memory consumption of our method as the forest size increases.

As can be seen in Figure 3, the time required to add new renderable trees grows very slowly. Although this graph only shows results for a 128MB cache, the other cache sizes exhibit similar behaviour. Even for a million trees, the majority of time, about 3 seconds, is taken up with hero creation. A million trees only requires 1.5 seconds to create, which is equivalent to approximately 650 trees per millisecond. The time required to place trees — a component of the creation time — grows linearly with the desired number of trees. As noted above, our system performs better than the uninstanced approach: we can create a million trees in the time that the uninstanced method is only able to create three hundred trees.

Figure 4 shows the memory requirements for each additional tree in the forest. Memory usage grows approximately linearly. The memory usage metric is the sum of the instance cache size, the size of all renderable trees and the size of all textures in the trees. The graph shows that, even up to a million trees, the memory requirements are dominated by the cache size. The memory requirements grow very slowly with the number of trees in the forest, which highlights an important advantage over the uninstanced approach. Without instancing, each additional tree consumes a large amount of graphics memory, which can be severely limiting on all but the most recent hardware. In our approach, however, each additional tree consumes less than 100 bytes, on average.

The effect of instancing changes with cache size: for a small cache far more instancing takes place. Figure 5 shows a forest with 100,000 trees and a cache size of 32MB (left) and 128MB (right). The shade of red indicates the extent to which branch instances occurred in the forest, with grey indicating that branch/part of the tree was not instanced at all. Note that the shading does not indicate spatial proximity or branch indices, but simply the degree of instancing. It is readily apparent that the larger cache size dramatically reduces the amount of instancing. For the 128MB cache, no more than 4 instances of any branch were used, while for the 32 MB cache, no more than 27 instances of any branch structure were re-used throughout the forest.

In the next section, we will discuss some of the disadvantages of our approach, in particular, the visual artifacts that can occur when using random instancing.

7.1 Limitations

Using instancing for procedural forest generation is not without its drawbacks. An important criticism is the possible reduction in visual quality due to excessive reuse of tree geometry. We attempt to reduce the effect on appearance by letting the non-deterministic nature of the L-systems decided where instances should be placed. This can still lead to problems: the L-system could randomly decide two trees that are near to each other should use the same instances, or, even worse, be constructed entirely from the same instance.

Although identical trees are unavoidable due to the random nature of L-systems, certain steps can be taken to mitigate this effect. The first is to use L-systems which are markedly non-deterministic, in that they are able to produce a large number of varied trees. The second approach is to detect when we are about to place an of-

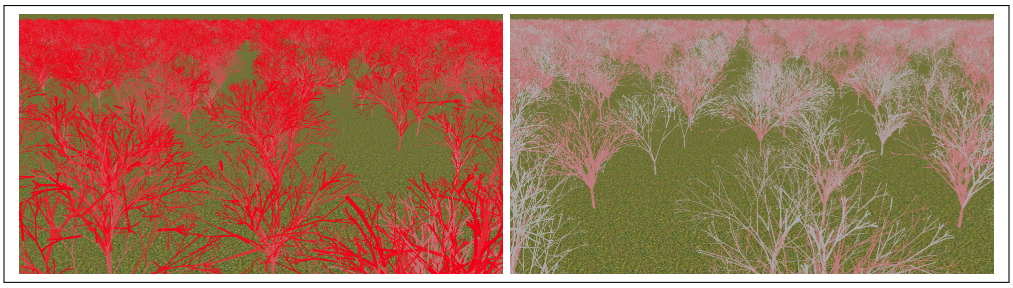


Figure 5: The effect of instancing for a forest of 100,000 trees. The shade of red indicates how frequently that branch structure was re-used in the forest. For the small cache (32MB, left) much more instancing is evident, as one would expect. However, no more than 27 instances were used for any branch structure. For the large cache (128MB, right) the number was only 4, as shown by the much lighter shading of red.

fending instance, one which is too close to another instance of the same type, and replace it with another instance. This second approach is made more difficult by the random nature of L-systems. The L-system could, for example, decide to add multiple instances to the cache which all represent the same geometry. In this case we would not even be aware that we are using the same geometry when using different instances.

Fortunately, the visual artifacts arising from instancing are not necessarily a problem. Informal user tests (a user-guided fly-through of the forest) revealed that most users do not notice that instances existed. Indeed, identical trees that are nearby often go unnoticed if oriented at a random angle and branches that are identical can appear at different levels of the tree which further masks their similarities.

8 CONCLUSIONS

We present a new scheme which significantly accelerates L-system tree creation and reduces memory overheads. By dynamically modifying the tree L-systems and making careful use of instancing, we can create large and varied forests quickly whilst using a bounded amount of memory. This is accomplished by filling a special fixed-size instance cache with sub-branch geometry derived from a much smaller set of ‘hero’ tree templates. Rendering is accomplished by looking up the appropriate geometry buffers in the instance cache and issuing draw calls using the associated transformation and texture metadata. Each new tree requires less than a 100 bytes of storage on average and takes less than 0.1 milliseconds to create. During our tests, we were able to create a forest of one million trees using approximately 350MB of memory in under 4.5 seconds. By contrast, the naïve algorithm was only able to generate 300 trees in the same amount of time.

There are several avenues for future work. Instancing improves memory requirements but may give rise to visually jarring repetition. In order to correct this behaviour we would need to scan the instance cache to detect when duplicate geometry is added so that we could ignore it. This is not an easy task since it requires complicated matching on the geometry and would likely slow the system down significantly. Alternatively, one could make the simplifying assumption that the same substring (the part of the string that represents the branch) represents the same geometry. Comparing strings instead of geometry is significantly easier (and more efficient).

Finally, while we have demonstrated our technique at work with L-systems, it is possible that other procedural tree generation algorithms could also benefit from our instance cache scheme.

9 ACKNOWLEDGEMENTS

This research was supported by an NRF/THRIP grant and the Centre for High Performance Computing.

10 REFERENCES

[1] Deussen, O.,Hanrahan, P., Lintermann, B., Měch, R., Pharr, M., and Prusinkiewicz, P. Realistic modeling and rendering of plant ecosystems, In: SIGGRAPH ’98, p. 275-86, 1998.

[2] Deussen, O., Colditz, C., Stamminger, M., and Drettakis, G. Interactive visualization of complex plant ecosystems, In: Proceedings of the conference on Visualization, p. 219-26, 2002.

[3] Eichhorst, P., and Savitch, W. Growth functions of stochastic Lindenmayer systems, Information and Control, 45(3), p.217-28, 1980.

[4] Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. Procedural modeling of buildings, In: SIGGRAPH ’06, p. 614-23, 2006.

[5] Prusinkiewicz, P., Lindenmayer, A., and Hanan, J. The algorithmic beauty of plants, The Virtual Laboratory, 1991.

[6] Prusinkiewicz, P. Graphical applications of L-systems, In: Proceedings of Graphics Interface, 1986.

[7] Salomaa, A., and Rozenberg, G. Handbook of Formal Languages, 1997.

Detecting Topologically Relevant Structures in Flows by Surface Integrals

Wieland Reich
Image and Signal Processing Group
University of Leipzig
PF 100920
04009 Leipzig, Germany
reich@informatik.uni-leipzig.de

Jens Kasten
Image and Signal Processing Group
University of Leipzig
PF 100920
04009 Leipzig, Germany
kasten@informatik.uni-leipzig.de

Gerik Scheuermann
Image and Signal Processing Group
University of Leipzig
PF 100920
04009 Leipzig, Germany
scheuermann@informatik.uni-leipzig.de

ABSTRACT

Gauss' theorem, which relates the flow through a surface to the vector field inside the surface, is an important tool in Flow Visualization. We are exploit the fact that the theorem can be further refined on polygonal cells and construct a process that encodes the particle movement through the boundary facets of these cells using transition matrices. By pure power iteration of transition matrices, various topological features, such as separation and invariant sets, can be extracted without having to rely on the classical techniques, e.g., interpolation, differentiation and numerical streamline integration. We will apply our method to steady vector fields with a focus on three dimensions.

Keywords
Surface integrals, vector field topology, flow visualization, transition matrices, stochastic processes

1 INTRODUCTION

Vector field topology reveals the basic features of a flow field, i.e., critical points, separatrices, separation surfaces and other invariant manifolds. For instance, it became a widely used tool for the feature-based analysis of stationary flow fields. For time-dependent flow fields, the finite-time Lyapunov exponent (FTLE) is often used to extract time-dependent counterparts of the structures known from vector field topology. Nevertheless, methods to extract vector field topology are based on interpolation, differentiation and numerical streamline integration.

Instead of relying on streamlines, one can use transition matrices. The entries of a transition matrix represent the probability that particles contained in one cell will enter another cell after some advection time. This leads to the theory of time-discrete Markov processes, which are a widely studied object of probabilistic theory. Here, the question is, if powers of the transition matrix will converge to a limit state. This corresponds to particles reaching a limit set, e.g., a critical point.

The generation of the transition matrices is a hard task. In [22], Reich and Scheuermann have used a combinatorial flow map, that was introduced by Chen et al. [3] to compute the outer approximation of an integration image of a cell after some advection time. Then, they were able to compute probabilities of particles leaving one cell and entering another cell. To compute the outer approximation, all edges of a cell have to be integrated adaptively.

In this paper, we present a novel algorithm to compute the transition matrices. The idea is to look at the outflow region of every cell. For two adjacent cells in 2D, the probability of the transition can be computed by relating the outflow at the edge to the outflow of the whole cell. This approach can be naturally extended to 3 dimensions by integrating the outflow along the cell surface instead of the edges.

While the approach using the method of Chen might result in transitions between cells that are not neighbored, our novel approach guarantees transitions only between neighbored cells. We therefore sample the transitions at the finest scale possible in this discrete setting. The speed of convergence of the transition matrices depends linearly on this scaling, because the computationally most costly stage is multiplying sparse matrices with vectors, which rises linearly in the number of cells, resulting in fast computations of the attracting and repelling limit sets. However, computing separation is still a computationally costly technique, because much more vectors will have to be iterated.

Our method is evaluated using data sets of different levels of complexity. Due to the extension to 3 dimensions of the existing work, we are now able to analyze more real world data with our novel approach.

The remainder of this paper is structured as follows: In section 2, we present approaches that are related to our method. The basics of surface integrals and transition matrices are explained in section 3. Afterwards, we present our method in section 4. The results are presented and discussed in section 5. The paper concludes with section 6.

2 RELATED WORK

Vector field topology has been introduced to by Helman and Hesselink [11] more than 20 years ago to the visualization community. Since then a lot of research has been devoted to the extraction of topological invariants. Initially, Helman and Hesselink extracted critical points and classified the points by linearizing the flow in the vicinity. Afterwards, the flow is segmented into basins of similar flow behavior using the separatrices that emanate from the saddle points. This method was then extended by several researchers. Scheuermann et al. [25] analyzed the boundary of the domain to extract a finer topology. Weinkauf et al. [32] extended this approach to three dimensions. Theisel et al.[29] introduced the concept of saddle connectors, which are the intersection of the separation surfaces emanating from the saddles points. Wischgoll et al. [34] extracted attracting and repelling periodic orbits in planar flows by searching for cell cycles.

Tricoche [31] studied the connection between the Poincaré index and vector field topology. Polthier et al. [20] use a discrete hodge decomposition to extract vector field singularities. Bhathia et al. [1] use edge maps as an alternative to streamlines.

The aforementioned methods analyze the vector field as a continuously given data set. This approach does not incorporate the grid and makes interpolation necessary. In contrast, discrete methods do not rely on interpolations but analyze the raw data. In particular, the work of Reininghaus [23] covers the extraction of topological structures in combinatorial vector fields. Here, the grid is represented as a graph and the vector field as an matching on this graph. Their method is based on the work of Forman [6]. Another combinatorial approach was presented by Chen et al. [3]. They use the images of triangular cells under advection to encode the flow field into a graph. Conley index theory is used to classify the strongly connected components as features. The work of Boczko et al. [5] can be seen as a special case of a Morse decomposition. Szymczak presented Morse decompositions of piecewise constant vector fields [28].

An approach to compute vector field topology in a discrete setting was presented by Reich et al. [22]. They use the theory of Markov processes [27] to extract vector field invariants.

In the last two decades, the extraction of topological structures in uncertain vector fields had come to attention in the flow visualization community, e.g., see Pang et al. [18]. Otto et al. [17] formulated convergence criteria for gaussian distributed density functions by Euler integration. Their method also uses the uncertain Poincaré index to distinguish between critical distributions. Petz et al. [19] presented an approach to analyze the probability of a critical point to be contained in a cell for uncertain vector fields. Schneider et al.[26] uses principal component analysis to detect separation in uncertain flows.

The list of the aforementioned publications related to vector field topology is naturally not complete. For a good overview that also cover topics of flow visualization, we refer to Weiskopf and Erlebacher [33], and Post et al. [21].

For time-dependent flow fields, vector field topology is not sensible to extract anymore. Here, a lot of analysis approaches search for Lagrangian coherent structures. An important approach to find these features was introduced by Haller [10] by introducing the Finite-time Lyapunov exponent (FTLE). Within the visualization community a lot of computational improvements or alternative computation methods have been proposed, see [2, 7, 13]. There is also some research done to compare vector field topology to structures extracted from the FTLE, e.g., see [24].

The stochastic processes in our work are also an important tool for image segmentation and pattern analysis [8].

3 MATHEMATICAL PRELIMINARIES

3.1 Surface Integrals

Surface integrals can be described as an observable quantity that measures the amount of leaving (entering) flow through a bounded surface in one time step. A famous theorem related to surface integrals is from C.F. Gauss. It states, that the flow f through a piecewise differentiable boundary of an area Ω is equal to divergence integral over the enclosed Ω:

$$\int_{\partial\Omega} < f, n > dA = \int_{\Omega} div\, f\, dV, \tag{1}$$

where n denotes the outer unit normal to $\partial\Omega$ and $< .,. >$ the inner product. The left side is a surface integral, while the right side is a integration over a volume.

The theorem has many applications in other sciences, e.g., in electrodynamics it implicates that there can be no electric field inside a hollow object. In one dimension, it is equivalent to the fundamental theorem of calculus.

In this chapter, we are going to introduce surface integrals in the Euclidean spaces $\mathbb{R}^2$ and $\mathbb{R}^3$. We will show that for triangular and tetrahedral cells in a piecewise linear (or piecewise constant) flow, the surface integrals reduce to (relatively) simple formulas. For interpolation schemes of higher order, there is no guarantee that there exists a closed formula, but the flow integral can still be calculated by using numerical integration techniques, like the Gaussian quadrature.

3.1.1 The 2D-Case

Since the boundary of a triangular cell in two dimensions is a closed path, the surface integral reduces to a line integral.

Lemma:
Let $f(x) : \mathbb{R}^2 \to \mathbb{R}^2$ be an affine linear field, i.e., it is of the form $A(x) + b$, with a matrix A and a constant vector b. Let p_1 and p_2 be two positions that bound an edge e of a triangle in $\mathbb{R}^2$. Then the surface integral of the flow f through e is

$$\int_e < f, n > dx = ||p_1 - p_2||_2 \cdot < \frac{f(p_1) + f(p_2)}{2}, n >, \tag{2}$$

where $||p_1 - p_2||_2$ is the length of edge e and n the outer unit normal of the edge.

3.1.2 The 3D-Case

Lemma:
Let $f(x) : \mathbb{R}^3 \to \mathbb{R}^3$ be an affine linear field again. Let T be a triangle in space, e. g. the face of a tetrahedron, with vertices positions p_1, p_2 and p_3. Then the surface integral of the flow f through T is

$$\int_T < f, n > d\sigma = \mathscr{A}(T) \cdot < \frac{f(p_1) + f(p_2) + f(p_3)}{3}, n >, \tag{3}$$

where $\mathscr{A}(T)$ is the area $\frac{1}{2} \cdot ||(p_1 - p_3) \times (p_2 - p_3)||_2$ spanned by the triangle T.

For the following sections, in particular 4.1, we rely on our gained, ready to implement formulas. Readers interested in the theory of multidimensional integration might also have a look in any vector calculus book, e.g., [16].

3.2 Transition Matrices

3.2.1 What are transition matrices?

In this section, we are going to give an insight in the basics of transition matrices, also called time-discrete Markov chains.

These matrices are linear operators that map a distribution vector v_1 to another distribution vector v_2 of the same dimension, preserving that every entry in the vector is greater or equal to zero and the sum of all entries is 1. In particular, a row-stochastic transition matrix M has the property, that all entries are greater or equal to zero and the sum of all entries in each row is 1, i.e.,

$$\sum_j m_{ij} = 1.$$

The entry m_{ij} describes the probability of the system from going from state i to state j.

Transition matrices are a stochastic processes with only a finite number of states which coincide with the size of matrix. The image of a distribution vector after k discrete time-steps is generated by multiplying the transposed distribution vector from the left n times. We have

$$v_{k+1}^T = v_k^T \cdot M. \tag{4}$$

As a sidenote, if we want to have a multiplication of the vector from the right, then our matrix M has to be column-stochastic instead. We have chosen the row-stochastic form, because it seems more intuitive when we move from state i to j than from j to i.

As one can see easily, the operator M is memoryless, i.e., the next state of the system only depends the current state, not on those before. The spectrum of M and the long term behavior of multiplication operations are of particular interest in probability theory [27].

Transition matrices may have many stationary states, these are vectors that do not change by multiplication formula (4). As a consequence, they have to be (left-) eigenvectors to the eigenvalue of 1:

$$v_k^T \cdot 1 = v_k^T \cdot M$$

The existence of at least one eigenvalue 1 is guaranteed for every M.

Though M and all distribution vectors v_k will always be bound in their norm by 1, not all multiplications involving transition matrices converge to a stationary state by power iteration. We will present a solution for this issue in section 4.

One of the most important theorems related to transition matrices is by Perron and Frobenius [27]. It states that if every entry in the matrix M is greater than zero, then there exists a unique eigenvector to the eigenvalue $\lambda = 1$ and every power iteration algorithm will converge to that eigenvector, which is the only stationary state.

Google [14] makes use of the Perron-Frobenius-Theorem to construct transition matrices that are guaranteed to converge. The states in the Google matrix are websites and the transition probabilities are determined by hyperlinks, that guide the user from one site to another. Further, there exists a very small chance, that the user chooses a completely random website, so the Google-Matrix will be densely populated. As a consequence, the unique stationary state of the Google matrix can be calculated by power iteration and delivers the page rank of each site, a measure for its importance, that can be used to order the results of search requests by the user.

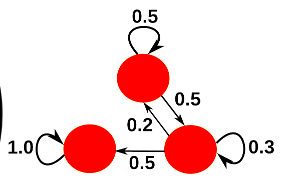
For the case of interpretation difficulties of transition matrices, it often helps to sketch a probabilistic graph that describes the movement, e.g., like in Figure 1.

3.2.2 Relation to Vector Field Topology

Transition matrices have been used before to extract topological features of a flow induced by a vector field.

$$\begin{pmatrix} 0.5 & 0.5 & 0 \\ 0.2 & 0.3 & 0.5 \\ 0 & 0 & 1.0 \end{pmatrix}$$

(a)



(b)

Figure 1: One example that expresses the duality between a (row-stochastic) transition matrix (a) and a probabilistic graph (b). The probability m_{ij} describes the likelihood to move from node i to node j.

The main idea of the preceding approaches was to encode particle movement from numerical streamline integration to a transition probability from cell to cell. Two important publications in that context are [4] and [22].
There is a duality between both dynamical systems. For a flow $\phi(t,x)$ associated with a vector field we have

$$\phi(0,x) = x$$

and

$$\phi(t_1, \phi(t_2, x)) = \phi(t_1 + t_2, x).$$

For a (row-stochastic) transition matrix

$$v^T \cdot M^0 = v^T$$

and

$$(v^T \cdot M^{n_1}) \cdot M^{n_2} = v^T \cdot M^{n_1+n_2}$$

holds. Further, the product of any(!) transition matrices M_1 and M_2 of the same dimension is a transition matrix again.
The conversion of a continuous system to a a discrete system poses several challenges:
While the flow $\phi(x,t)$ is invertible in the range of its existence, i.e., we are able to go back to our original position by an integration using the same time with a negative sign, transition matrices do not need to be invertible. A possible solution is to construct two transition matrices, one describing the forward movement (M_+) and one for the backward movement (M_-). However, it is still not guaranteed that the equation

$$M_+ \cdot M_- = Id$$

is fulfilled here; so it is natural to ask why we convert the flow to that discrete system. The reason is not only the gained robustness and uniform treatment of invariant sets. Transition matrices allow us to analyze the sensitivity of the initial value problem at infinite times, which is not possible with purely particle-distance-based algorithms like FTLE. In publication [22], Reich et al. make use of that fact and extract separating features of planar flows.

4 THE ALGORITHM

Now we are going to combine the surface integrals with transition matrices, i.e., we move from local feature extraction to a global topology by describing the interaction between the cells and the flow through their common facettes.
Unlike the preceding work, our algorithm will be completely independent from numerical streamline integration and works in any dimension. However, our presented results will primarily contain 3-dimensional flows, while, for the sake of simplicity sub-steps of the method are illustrated in 2D.

4.1 Encoding Particle Movement

Recall the Gauss Theorem (1). If we look at the right side, we have a volume integral over the divergence of a region Ω, say, a cell of a piecewise linear vector field. While the integral can be zero, e.g., the cell contains a purely rotational stationary point, there are still particle movements between the cell and its neighbor cells. So the right side is of no use when we want to create transition matrices. Let us have a look at the surface integral instead:

$$\int_{\partial\Omega} < f, n > dA.$$

This can be further refined to

$$\sum_i \int_{(\partial\Omega)_i} < f, n > dA,$$

where $(\partial\Omega)_i$ is a boundary segment of our cell, i.e. an edge i of a triangle, or a face i of a tetrahedron. Further, we can refine the formula by distinguishing between in- and outflow:

$$\sum_i \int_{(\partial\Omega_+)_i} < f, n > dA \; + \; \sum_i \int_{(\partial\Omega_-)_i} < f, n > dA,$$

where $(\partial\Omega_+)_i$ is the region where $< f, n > \geq 0$ holds (outflow), and $(\partial\Omega_-)_i$ the region where $< f, n >$ is smaller than 0 (inflow).
It follows immediately from our formulas, that if we have a piecewise linear flow that is divergence-free, then

$$\sum_i \int_{(\partial\Omega_+)_i} < f, n > dA = -\sum_i \int_{(\partial\Omega_-)_i} < f, n > dA$$

must hold for every cell.
By assuming a linear field, the flow relative to each boundary edge/facette can change its behavior just once, at a tangential point in 2D, or a tangential line in 3D. The tangential point, respectively the endpoints of the tangential line can computed by seeking a solution λ in $[0..1]$ satisfying

$$\lambda = \frac{< n, f(p_2) >}{< n, f(p_2) - f(p_1) >}$$

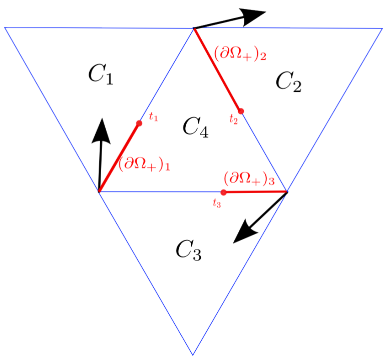


Figure 2: An example of encoding particle movement. All three edges of the cell C_4 have a non-trivial exit set. We compute the integrals of $< f,n >$ by our formulas from section 3 using the vertices and the tangential points as integration range (red lines). The transition probability from cell C_4 to C_1 is determined by the value of the surface integral on $(\partial\Omega_+)_1$ divided by the value on $\sum_i (\partial\Omega_+)_i$.

on an edge spanned by p_1 and p_2. Afterwards, the tangential point t can be computed by $\lambda \cdot p_1 + (1-\lambda) \cdot p_2$. The vector at a tangential point or a point on a tangential line is never needed to be evaluated, because it has no component in direction of the normal, so it can be assumed as zero in our integration formulas. We do not need interpolation at any sub-step of the algorithm.
As an example, the surface integral for a edge with tangential point t in 2D splits up into

$$\int_e < f,n > dx = \int_{e_+} < f,n > dx + \int_{e_-} < f,n > dx$$
$$= ||p_1 - t||_2 \cdot < \frac{f(p_1)+0}{2}, n >$$
$$+ ||t - p_2||_2 \cdot < \frac{0 + f(p_2)}{2}, n >,$$

where it still has to be checked which term is the positive part.
In case of a face of a tetrahedron, a tangential line decomposes a boundary face into a triangle and a quad. The latter one can again be decomposed into two triangles, so it is necessary to evaluate the 3D-surface integral three times per face. If the flow is transverse at a boundary edge/face, which is the common case, in particular for flows with weak rotation, we can compute the surface integral directly in one step. For a 2D-illustration of a evaluation also see Figure 2.

The following is the key aspect of the whole paper. We are going to put the outflow through a boundary edge/face in relation to the outflow of the whole cell. Which yields the quotient "outflow through a face i that connects cell a with cell b" divided by "outflow through the whole cell a", or

$$m_{ab} = \frac{\int_{(\partial\Omega_+)_i} < f,n > dA}{\sum_i \int_{(\partial\Omega_+)_i} < f,n > dA}, \qquad (5)$$

where $\partial\Omega$ is the boundary of cell a and face i is connecting a with b. The values of m_{ab} fill our transition matrix M_+. We can state a analogous formula for M_- by just substituting $\partial\Omega_+$ with $\partial\Omega_-$.
If we sum up all m_{ab} from a cell a and all of its neighbors b, the result will always be 1.0, so M_+ and M_- will be transition matrices, that describe the weighted outflow/inflow of a linear flow through a cell.
To avoid division by zero, we must intercept the cases, where there is no outflow/inflow at all. These cases are cells containing nodal stationary points, so we just set m_{aa} to 1.0 and all m_{ab} are 0.0 for $a \neq b$.

4.2 Transition Matrix Processing

From now on, the rest of the algorithm will be pure matrix-vector-iterations with our constructed transition matrices M_+ and M_-.
Since the amount of cell neighbors is always limited to 3 (in 2D) or 4 (in 3D), our transition matrix will be sparse of the compressed size $(N \times 3)$ or $(N \times 4)$, where N is the number of cells in our dataset. There are programs with proper data structures [9] who are designed for the procession of sparse matrices. Alternatively, one could also use an own implementation, e.g., using the construct $vector < map < unsigned\ int, float >>$ in C++, which also has been used to process the transition matrices generated from the datasets in this paper. We did not experience any significant computational time changes when switching from our classes to [9]. The complexity of a matrix-vector-multiplication is reduced from $O(N^2)$ to $O(N)$, when the matrix is not densely populated.
The power iteration is a task always to be executed the same way:
Given an equivalence precision $\varepsilon > 0$ and an initial vector v_0 we compute

$$v_{k+1}^T = v_k^T \cdot M,$$

until $||v_{k+1} - v_k|| < \varepsilon$.
A small ε will lead to accurate results but can increase the computational time significantly. A summary on multiple power iteration methods for matrices can be found in [30].
We are going to use three types of useful initial distributions:

- The uniform distribution $v_0 = u$, where all entries of u are $\frac{1}{N}$ with N being the number of cells.
- The impulse distribution $v_0 = e_i$, where the i-th position of the vector contains a 1.0 and all others are 0.0. This distribution is localized in cell i only.
- The neighborhood distribution $v_0 = n_i$, where for all neighbors j of a cell i, who share common edges/faces, have the value $\frac{1}{3}$ (in 2D) or $\frac{1}{4}$ (in 3D).

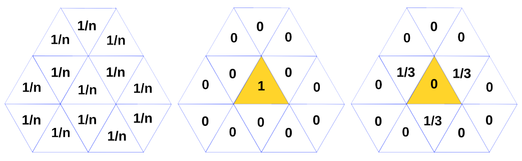


Figure 3: All three distributions that were used by us for power iterations: (a) a uniform distribution, n is the number of cells, (b) an impulse distribution, which is only located in the yellow cell, (c) a neighborhood distribution, located around the yellow cell, for a tetrahedron, the values have to be $\frac{1}{4}$.

> This distribution is localized in the neighborhood of cell i in triangular or tetrahedral mesh.

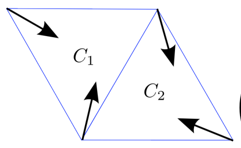
An illustration of all distributions can be found in Figure 3. We do not consider cells as 'proper' neighbors if they share just a common vertex, because the resulting flow integrals will be trivial on a set with measure 0.

4.2.1 *Invariant Sets*

Since we are no longer moving in the conventional dynamical system $\phi(x,t)$, which is induced by a vector field, we must redefine the invariant sets:
An invariant set is an eigenvector of either M_+ or M_- to the eigenvalue of 1. An invariant set is attracting, if the matrix is M_+, else it is repelling.
As a remark, these are indeed invariants, because if we see the eigenvector as a collection of cells, then the set of these cells, that are represented by non-zero entries, does not change by any new multiplication with the transition matrix. The set of all cells in an invariant set is always connected, because the movement between those cells always takes place between their common edges/faces.
To extract all attracting stationary states, we just need to iterate with M_+ and the initial vector u, which represents a distribution over the whole domain. For the repelling ones we take M_-. We experienced that purely rotational stationary points can be extracted, together with their neighborhood, by both methods.
Using u as an initial vector also has the side effect, that sinks and sources are also weighted with the size of their α/ω-basin. Some invariants might attract or repel "more" particles than others.

4.2.2 *Separation*

Separation manifolds cannot be automatically derived from the spectrum of our transition matrices. However, we still are able to do a power iteration with the initial vectors e_i and n_i and compare the resulting stationary states by the l_1-metric and eventually measure the dependence of the iteration process from the initial vector. We obtain forward separation by using M_+ and backward separation by using M_-. That method has by far



$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

(a) (b)

Figure 4: A possible case that could, without using equation (6), lead to a divergent transition matrix : (a) there is clearly a singularity in one of those cells, but if the flow leaves cell C_1, it will be towards C_2 with probability 1.0 and vice versa. The resulting matrix (b) will be divergent, except one chooses the initial distribution $(0.5, 0.5)$.

the highest complexity, which can be estimated by the computational time of extracting all invariants times the number of cells in the dataset. Separation features extracted by transition matrices have to be seen in a global context by describing particles that tend to have different limit sets. Every finite-time expansion of flow, as well as particle distances, will not influence our result.

4.2.3 *Boundary Topology*

We process the boundary similar to the method by Mahrous et al. [15]. First, we extract all connected boundary segments, where the flow leaves the domain. These segments will be treated like additional cells and have mapping of probability 1.0 to themselves. Finally all cells adjacent to them, will be mapped to these artificial invariants by the probability that is, as in the ordinary case, determined by the outflow integral. The size of the matrix will grow by the number of so-called exit sets of the domain, which are in general of much smaller cardinality than the number of cells in our dataset.

4.3 How to prevent divergent transition matrices

Transition matrices that are generated from surface integrals may be divergent, i.e., not every stationary state may be reached by power-iteration only. Most of these cases are clusters of cells that are ordered in a cycle, where the transition probability from one cell to its successor is 1.0. If one puts an impulse distribution in one of these cells, the power-iteration will just move that distribution around the cycle without reaching a stationary state. A special case is given in Figure 4, where a critical point near the common edge of two cells leads to a divergent 2-cycle.
However, we are able to perform one simple operation, so that a new matrix M_{new} will have the same stationary states, but will be convergent by potentiation. We set

$$M_{new} = \frac{M_{old}^2 + M_{old}}{2} \tag{6}$$

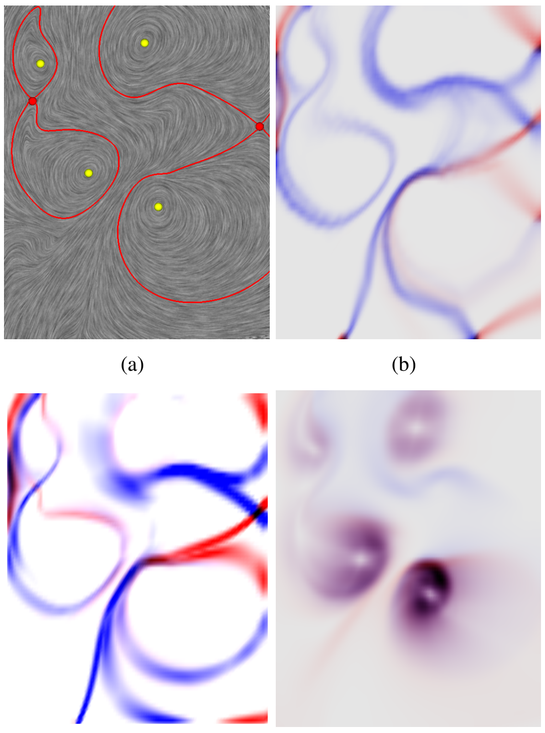


Figure 5: A CFD-dataset simulating a fluid entering at the left bottom: (a) 4 center points (yellow) and 2 saddles (red) with their separatrices, the left ones form two homoclinic orbits, the background is a LIC, (b) separation computed with a streamline-based method from the preceding work ([22]), (c) separation computed with surface integrals by iterating all distributions e_i and n_i, (d) iteration with uniform distribution u reveals the center points.

and obtain a process with the same stationary states. It can be easily shown that the only remaining eigenvalue in M_{new}, that has magnitude 1, is 1 itself. There are no other eigenvalues on the unit circle of the complex plane present, which could cause divergent behaviour. Our new transition matrix will always converge by power iteration. In the special case of Figure 4, the resulting new matrix block has only the entries 0.5 and both cells belong to the same stationary state which is associated with the critical point of the vector field.

It is not necessary to compute the explicit second power in practice, the propagated iteration scheme just needs to be extended by another step with an additional averaging of two distribution vectors. As a side effect, vector fields containing highly rotational flows and many closed streamlines are also faster processed, because distributions get blurred immediately along the probabilistic streamline.

The discrepancy to the former streamline-based method in [22] is, that transition matrices that have been generated by flow maps are practically never divergent, because a flow map never maps one cell with a probability of exactly 1 to another cell.

Google [14] uses a similar method when computing the page rank of websites. Their so-called matrix damping formula is

$$\widetilde{M}_{google} = (1-\alpha)\cdot M + \alpha \cdot G, \tag{7}$$

where $G = \{\frac{1}{N}\}$, N being the size of G.

The convergence is ensured by the Perron-Frobenius-Theorem [27] there. If we would have wanted to apply such a convex linear combination to our problems, the consequences would be devastating. Not only that the resulting dense matrices make efficient computations with large datasets extremely costly. The resulting matrix from (7) has an unique stationary state and all power iterations will converge to that eigenvector. Measuring separation will be impossible. Finally, our formula is superior in the feature extraction of flows, because it preserves the low population of entries in the sparse matrices and the multidimensionality of the Eigenspace of eigenvalue 1.

Another interesting aspect is a geometric interpretation of M^{∞}_{new}, which is now well-defined. We already know that the norm of transition matrices is bounded by 1, implicating the same for all (possibly complex) eigenvalues . If we consider

$$M^n_{new}\cdot v = \lambda^n \cdot v$$

for $n \to \infty$, all λ obeying $|\lambda| < 1$ will be set to zero. From the existence of M^{∞}_{new} it can be excluded that, with the exception of 1, there are any other eigenvalues of magnitude 1. Eventually 1 and 0 are the only "surviving" eigenvalues in M^{∞}_{new}, which can now be considered as a projection operator. We project an initial distribution into the Eigenspace of eigenvalue 1, which is spanned by all stationary distributions. In fact, all of our used power iterations are projections.

5 RESULTS AND DISCUSSION

All iteration methods result in scalar data which is either visualized by a color map (2D) or volume rendering and isosurfaces (3D).

5.1 Artificial Data

5.1.1 The Lorenz Attractor

The Lorenz-Attractor was discovered when E.N. Lorenz attempted to set up a system of differential equations that would explain some of the unpredictable behavior of the weather. It is one of the most popular chaotic systems featuring a dense collection of unstable streamlines. The attractor in our example obeys the ODE-system

$$\begin{aligned} x' &= 10(y-x)\\ y' &= x(28-z)-y\\ z' &= xy-\frac{8}{3}z, \end{aligned}$$

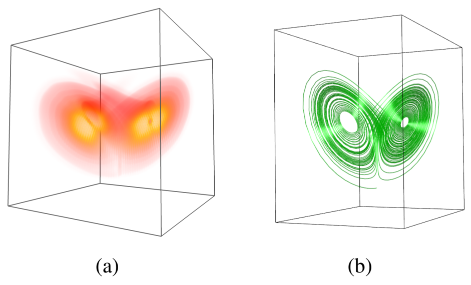


Figure 6: The Lorenz-Attractor: (a) The uniform distribution u is iterated by a transition matrix that was generated by surface integrals, (b) illuminated streamlines were planted in the detected region.

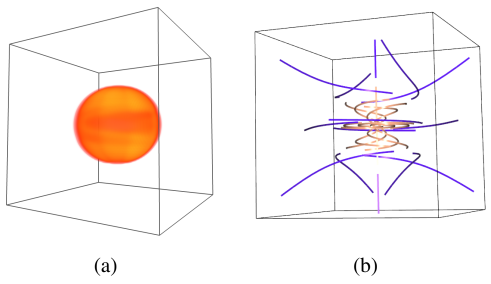


Figure 7: The Invariant Sphere: (a) e_i and n_i were cellwise iterated by a transition matrix generated by surface integrals, (b) streamlines were planted in various positions, the violett colored ones were outside of the domain $|r| = 1$ and diverge while the red colored ones were inside and remain in the sphere for all integration times.

and can be seen in Figure 6. We had chosen the uniform distribution and used power iteration with M_+.

5.1.2 *The Invariant Sphere*

The invariant sphere can be produced by using the ODE-system

$$\begin{aligned} x' &= -y + x \cdot (1-r) \\ y' &= x + y \cdot (1-r) \\ z' &= z \cdot (1-r), \end{aligned}$$

where $r = \sqrt{x^2+y^2+z^2}$. It consists of an infinite number of dense closed streamlines for $r = 1$, every streamline seeded in the neighborhood of the sphere will converge to one of these closed orbits. Low-precision integrators tend to produce unstable solutions here. We computed the backward separation with M_- and successfully extracted the whole sphere, which is visible in Figure 7.

5.2 CFD-Simulations

5.2.1 *Swirling Jet*

Figure 5 will be our only example of a 2D-dataset. In contrast to the preceding work, our focus will be on 3 dimensions. We computed forward and backward separation with M_+ and M_- and extracted the topological skeleton of the fluid simulation quite accurate. Moreover, the surface integral method seems to extract boundary related features better than the streamline-based approach in [22].

5.2.2 *Gas Furnace Chamber (Velocity)*

In Figure 8 we analyzed the backward separation field of the gas furnace chamber. The gas furnace chamber is a divergence-free vector field on a grid that contains approximately $2 \cdot 10^5$ cells. The forward separation field is of minor interest, because all particles will end in the same exit set. High separation values around the injectors were detected and their influence to the topology of the field is also revealed.
The global separation is hard to interpret by seeding stream surfaces in the stable and unstable manifold of saddle points, because there are simply too many of them inside the vector field.

6 CONCLUSION AND FUTURE WORK

We presented a novel approach to the topology of steady 3D vector fields by exploiting that surface integrals can be expressed as simple formulas on piecewise linear vector fields. We constructed transition matrices by the information of these integrals, which allow an infinite-time evaluation of separation and are able to extract many topological features of 3D flows without having to rely on numerical integration schemes, e.g., a forth-order Runge-Kutta. The latter advantage develops into great robustness towards classical problems, like critical points located near the boundary of cell, boundary slip conditions, and stiffness problems of ordinary differential equations. Transition matrices are much easier constructed with surface integrals in any dimension than with the streamline based approach, which is still a not completely solved task for a tetrahedral mesh ([3], [22]). While the surface integral based method produces smoother separating structures in locations near the boundary of the domain, both methods do not differ much in their high computational times, which can be several hours or even days for large data.
Further, we neither need a differential operator, nor evaluation (interpolation) of values outside of our vertices in our grid. The algorithm also includes the boundary of our domain into its calculations.
Regarding computational costs, the topology of the vector field is much more influential than the number of cells. Distributions in gradient fields converge very quickly to a stationary state. Highly rotational fields take much longer.
The prospects on the following work can be subdivided into three branches:

1. **Reduction of the Computational Costs** To make significant progress in reducing the computational time, a GPU-implementation will be necessary. To the best of our knowledge, prevalent GPU-based linear algebra software parallelizes row- and column-operations of matrix-vector-products. However, what we need is a parallelization in a much more extensive context, i. e., allowing multiple vectors being operated on by the same matrix.
   Further, one might think about a better initial distribution seeding, e. g., similar to a divide and conquer approach, so that we do not have to iterate each single cell by an impulse distribution.

2. **Uncertainty** Because transition matrices are a special type of stochastic processes, it would make them a very useful tool to explore uncertainty in dynamics, which has been stated as one of the most important branches in the future of visualization [12]. We do not need to change our method at any stage for that. We can manipulate initial distributions, or even our matrix, in any way we want and study the changes that they create.

3. **Time-Dependent Flows** The fact that we are able to process steady 3D vector fields automatically opens the gates to a method, which can create transition matrices of a time-dependent 2D vector field by joining all time slices in their order. Unfortunately, the concept of stationary states of distributions works only, if the time-dependent field is periodic.

7 REFERENCES

[1] H. Bhatia, S. Jadhav, P.-T. Bremer, G. Chen, J. A. Levine, L. G. Nonato, and V. Pascucci. Edge maps: Representing flow with bounded error. In *PacificVis*, pages 75–82, 2011.

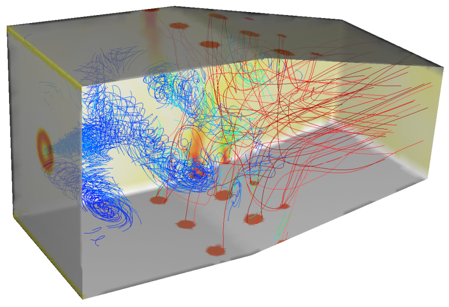
[2] S. L. Brunton and C. W. Rowley. Fast computation of finite-time lyapunov exponent fields for unsteady flows. In *Chaos*, volume 20, 2010.

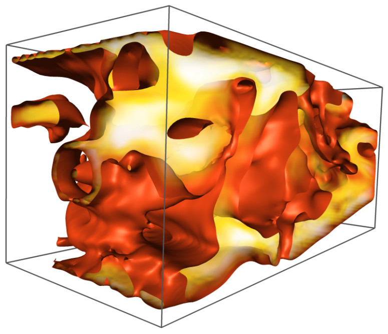
[3] G. Chen, K. Mischakow, R.S.Laramee, and E. Zhang. Efficient morse decompositions of vector fields. In *IEEE Transactions on Visualization and Computer Graphics*, volume 14, pages 848–862, 2008.

[4] M. Dellnitz and O. Junge. On the approximation of complicated dynamical behavior. In *SIAM Journal on Numerical Analysis*, volume 36, pages 491–515, 1999.

[5] K. M. E. Boczko, W. Kalies. Polygonal approximation of flows. 2004.

[6] R. Forman. Morse theory for cell complexes. In *Advances in Mathematics*, volume 134, pages 90–145, 1998.


(a)


(b)

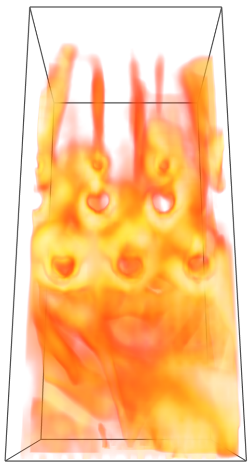

(c)

Figure 8: The gas furnace chamber: (a) One main injector located on the left, where the blue streamlines were seeded. Many smaller injectors on the bottom and top of the chamber, where the red streamlines are entering. (b) Backward separation field of the gas furnace chamber: isosurfaces on the highest separation values, (c) volume rendering on the same dataset with a viewpoint from above.

[7] C. Garth, A. Wiebel, X. Tricoche, K. Joy, and G. Scheuermann. Lagrangian visualization of flow-embedded surface structures. In *Computer Graphics Forum*, volume 27, pages 767–774, 2008.

[8] L. Grady. Random Walks for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783, 2006.

[9] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[10] G. Haller. Distinguished material surfaces and coherent structures in three-dimensional flows. In *Physica D*, volume 149, pages 248–277, 2001.

[11] J. Helman and L. Hesselink. Visualizing vector field topology in fluid flows. In *IEEE Computer Graphics and Applications*, volume 11, pages 36–46, 1991.

[12] C. Johnson. Top scientific visualization research problems. In *IEEE Computer Graphics and Applications*, volume 24, pages 13–17, 2004.

[13] J. Kasten, C. Petz, I. Hotz, B. Noack, and H.-C. Hege. Localized finite-time lyapunov exponent for unsteady flow analysis. In *Proceedings Vision, Modeling and Visualization 2008*, pages 265–274, 2009.

[14] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.

[15] K. M. Mahrous, B. H. J. C. Bennett, and K. I. Joy. Improving topological segmentation of three-dimensional vector fields. In *IEEE TCVG Symposium on Visualization*, 2003.

[16] J. Marsden and A. Tromba. *Vector Calculus*. W. H. Freeman, 2003.

[17] M. Otto, T. Germer, H.-C. Hege, and H. Theisel. Uncertain 2d vector field topology. In *Computer Graphics Forum*, volume 29, pages 347–356, 2010.

[18] A.-T. Pang, C.-M. Wittenbrink, and S.-K. Lodh. Approaches to uncertainty visualization. *The Visual Computer*, 13:370–390, 1996.

[19] C. Petz, K. Pöthkow, and H.-C. Hege. Probabilistic local features in uncertain vector fields with spatial correlation. *Computer Graphics Forum*, 31(3):1045 – 1054, 2012.

[20] K. Polthier and E. Preuss. Identifying vector field singularities using a discrete hodge decomposition. pages 112–134. Springer Verlag, 2002.

[21] F. Post, B. Vrolijk, H. Hauser, R. Laramee, and H. Doleisch. The state of art in flow visualization: Feature extraction and tracking. In *Computer Graphics Forum*, volume 22, pages 775–792, 2003.

[22] W. Reich and G. Scheuermann. Analysis of streamline separation at infinity using time-discrete markov chains. *IEEE Transactions on Visualization and Computer Graphics*, 18, 2012.

[23] J. Reininghaus and I. Hotz. Combinatorial 2d vector field topology extraction and simplification. In *Topology in Visualization*, 2010.

[24] F. Sadlo and D. Weiskopf. Time-Dependent 2-D Vector Field Topology: An Approach Inspired by Lagrangian Coherent Structures. *Computer Graphics Forum*, 29(1):88–100, 2010.

[25] G. Scheuermann, B. Hamann, K. Joy, and W. Kollmann. Visualizing Local Vector Field Topology. *Journal of Electronic Imaging*, 9(4):356–367, 2000.

[26] D. Schneider, J. Fuhrmann, W. Reich, and G. Scheuermann. A variance based ftle-like method for unsteady uncertain vector fields. In *Topological Methods in Data Analysis and Visualization II*, pages 255–268, 2012.

[27] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[28] A. Szymczak and E. Zhang. Robust Morse Decompositions of Piecewise Constant Vector Fields. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):938–951, 2012.

[29] H. Theisel, T. Weinkauf, H.-C. Hege, and H.-P. Seidel. Saddle connectors - an approach to visualizing the topological skeleton of complex 3d vector fields. In G. Turk, J. J. van Wijk, and R. Moorhead, editors, *Proc. IEEE Visualization 2003*, pages 225–232, Seattle, U.S.A., October 2003.

[30] L.-N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.

[31] X. Tricoche. *Vector and Tensor Field Topology Simplification, Tracking, and Visualization*. PhD thesis, 2002.

[32] T. Weinkauf, H. Theisel, H. c. Hege, and H. p. Seidel. Boundary switch connectors for topological visualization of complex 3d vector fields. In *In Proc. VisSym 04*, pages 183–192, 2004.

[33] D. Weiskopf and B. Erlebacher. Overview of flow visualization. In *The Visualization Handbook*, pages 261–278, 2005.

[34] T. Wischgoll and G. Scheuermann. Detection and visualization of planar closed streamlines. In *IEEE Trans. Visualization and CG*, volume 7, pages 165–172, 2001.