Clothing Manipulation

Takeo Igarashi
Computer Science Department, University of Tokyo
7-3-1 Hongo, Bunkyo, Tokyo, 113-0033 Japan
E-mail: takeo@acm.org

John F. Hughes
Computer Science Department, Brown University
Providence, RI 02912, USA
E-mail: jfh@cs.brown.edu

ABSTRACT
This paper presents interaction techniques (and the underlying implementations) for putting clothes on a 3D character and manipulating them. The user paints freeform marks on the clothes and corresponding marks on the 3D character; the system then puts the clothes around the body so that corresponding marks match. Internally, the system *grows* the clothes on the body surface around the marks while maintaining basic cloth constraints via simple relaxation steps. The entire computation takes a few seconds. After that, the user can adjust the placement of the clothes by an enhanced dragging operation. Unlike standard dragging where the user moves a set of vertices in a single direction in 3D space, our dragging operation moves the cloth *along the body surface* to make possible more flexible operations. The user can apply pushpins to fix certain cloth points during dragging. The techniques are ideal for specifying an initial cloth configuration before applying a more sophisticated cloth simulation.

KEYWORDS: User Interface, Clothing.

INTRODUCTION
Putting clothes on a 3D character is often a tedious, time-consuming task. A typical approach is to place parts of the clothes around the target body as rigid thin plates and use a simulation to enforce "stitch-together" constraints and show the effects of gravity [23]. The 3D character may be placed in a particular pose (e.g., arms outstretched) and then some "throwaway" animation may be used to get the character into a desired pose [3]. However, placing thin plates in free 3D space using a 2D input device is difficult, and it is not very flexible for exploring various nonstandard ways of wearing clothes. Recent fast cloth simulation systems enable real-time manipulation of clothes: the user can grab a piece of clothing and drag it around in 3D space [7,8]. But this is like manipulating clothes with chopsticks; it's not ideal for putting clothes on a 3D character.

In this paper we introduce a set of *interaction techniques* for putting clothes on a 3D character (here called the *body*) quickly and intuitively using 2D input devices. The techniques are designed for specifying an approximate initial cloth configuration before applying a high-quality cloth simulation to obtain a final, good-looking cloth shape or animation. The intention is that the interface should also be useful for exploring various cloth configurations quickly during the design process, both in 3D character design and real-world fashion design. The interaction techniques are supported by an underlying *approximate simulation*

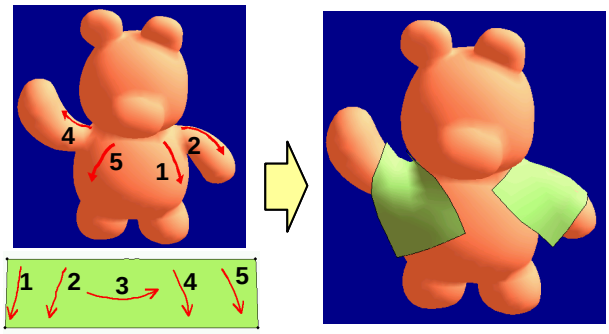
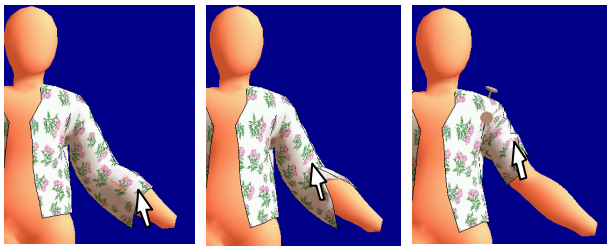


Figure 1: Wrapping. The user paints pairs of freeform marks on the target body and on the clothes (left); the system places the clothes on the body so that the corresponding marks match (right). The result appears almost instantly. (The mark numbering has been added by hand to clarify the correspondences.)



Before dragging Vertex dragging Surface dragging

Figure 2: Surface dragging. A typical vertex-dragging operation moves only one vertex explicitly, causing large local distortion. Surface dragging explicitly propagates motion across the clothes, enabling global manipulation. The pushpin on the shoulder blocks further propagation.

UIST'02, October 27-30, 2002, Paris, FRANCE.

technique whose details we describe briefly, particularly as they relate to the interactions, but which could be replaced by any other sufficiently rapid simulation. Our sole requirement is that both the clothing and the 3D character be represented as polygonal manifold meshes.

The first technique, *wrapping*, is for putting the clothes on the body from scratch. The user paints freeform marks on the clothes and corresponding marks on the body, and in a few seconds the system places the clothes on the body in such a way that the corresponding marks match (Figure 1). The second technique, *surface dragging*, is for adjusting the configuration of clothes already on the body. While a typical cloth-dragging operation moves a set of vertices in a single direction in 3D space, our dragging operation moves the cloth along the body surface (Figure 2). The user can also place pushpins to hold some clothing parts fixed during dragging. We describe the user interface of the system first and describe the implementations of those operations later.

RELATED WORK

The computer graphics community has been interested in cloth modeling for decades [14,23]. Early approaches were purely geometric [24], but recent systems use physically based simulations for generating realistic pictures and animations [2,5,6]. Some systems also allow real-time cloth manipulation [8]: the user can drag the cloth around in a 3D space with appropriate cloth-like behavior, and can even get haptic feedback [7].

Cloth simulation is common in commercial 3D computer graphics programs today [20,22]. The typical interface for putting clothes on a 3D character is to set the character in a canonical dress-up pose, place the clothes around it as rigid objects, and then start a simulation to let the clothes fall into a natural position. Some systems let the user specify various constraints or motion paths for specific cloth vertices to control the simulation.

The garment-design industry has been using 2D pattern design programs (apparel CAD) for years [10,17], and recently started incorporating 3D features [1,9]. They use predefined mappings between 2D cloth patterns and a 3D mannequin surface, where the manipulations in the 2D editor appear simultaneously in 3D space.

Our interface is motivated by the recent sketching interfaces for 3D modeling [12,25]. These tools are designed for exploratory design and for communication during discussion; they are designed to support ease of use in rapid model creation rather than the refined kinds of modeling needed in the final design stages. Our goal is to develop similar easy-to-use design tools for clothes.

THE USER INTERFACE

The system has two windows: the *pattern-design window* for editing 2D cloth patterns and the *3D window* for manipulating cloth on a 3D character (the *body*)(Figure 3). The user first edits a 2D cloth pattern in the pattern-design window and then puts the clothes on the body using the wrapping operation. The user can then manipulate the clothing using surface dragging and pushpins.

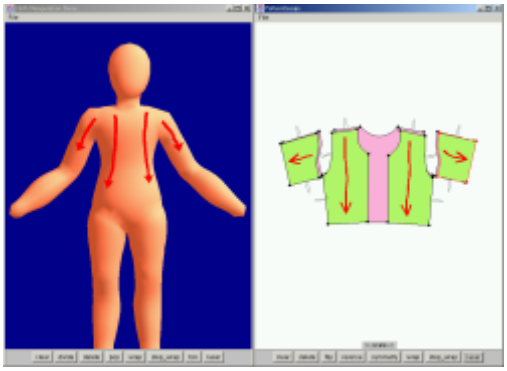


Figure 3: A screen snapshot of the system. Pink is the inside surface of the cloth, green is the outside, and the small gray lines indicate sewing constraints applied by the user.

The 2D pattern editor is a specialized 2D drawing program. The user draws pieces of cloth as closed polygons that can be freely moved and scaled. Each piece has distinct front and back sides, and the user can flip a piece to see its other side. The user can also indicate that two edges from different pieces are to be connected by specifying *sewing constraints*. The system maintains equality of the lengths of connected edges during pattern editing. We also provide simple editing operations such as duplication of pieces and making a piece horizontally symmetric. The implementation of the pattern editor is straightforward and hence not described here.

The 3D viewing window works as a typical 3D object viewer. The user can rotate and move the body three-dimensionally using the right mouse button [15].

Wrapping

To put 2D clothes on the body, the user paints freeform marks on the cloth pattern and the body using the left mouse button (Figure 1). The marks are numbered internally based on the order of painting independently on the clothes and the body, and marks with corresponding numbers are associated with each other.

After painting the marks, the user presses the "wrap" button. The system calculates the desired 3D cloth configuration on the body and shows the result in the 3D window. For the examples shown in the figures here, wrapping takes a few seconds in our current implementation. After presenting the initial result, the system continuously refines the cloth configuration via a relaxation operation. Similar mark-based interaction technique is used in a feature-based image morphing [4].

Wrapping is a best-effort operation. The system tries for a reasonable result satisfying the constraints specified by the user, but undesirable results can be generated depending

on the configuration of marks. If the problem is small, the user can adjust the cloth placement via the surface dragging operation. However, if the result has serious topological problems such as the body penetrating the cloth, the user must cancel the wrapping operation and adjust the configuration of marks.

For the user's convenience, the system includes a "laser-paint" mode [11] in which the user's mark is automatically painted on both the front and back sides of the body. In the pattern-design window a laser-painted mark is painted on the front-facing cloth piece and the underlying back-facing piece (if any).

Wrapping can also be used to adjust the configuration of clothes on the body (called *rewrapping*) (Figure 4). Pushpins may be used to restrict the rewrapped area. Rewrapping is especially useful for edits involving topological changes (Figure 4 right).

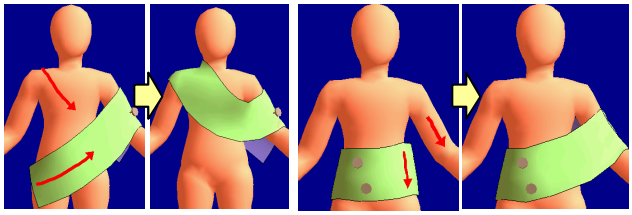


Figure 4: Rewrapping clothes already on the body. The pushpins limit the rewrapping region.

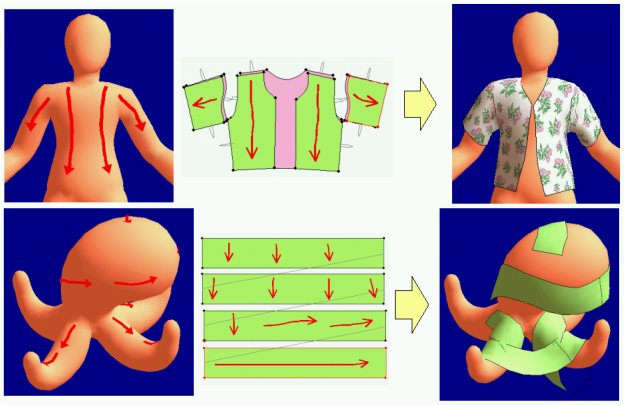


Figure 5: Examples of wrapping. In the first, laser-paint is used to duplicate marks front and back; in the second, we put a scarf on an octopus.

Figure 5 shows two examples of wrapping. The system generally returns the desired results, but the user must provide enough marks to avoid undesirable effects. For example, both the front and back sides must be painted to put a sleeve around an arm. Otherwise the cloth stays on one side of the arm. It is difficult to see the correspondence between pairs of marks in a still picture, but is easy for the user, who can paint corresponding marks on the pattern and the body alternately.

Surface Dragging

After putting clothes on the body, the user can adjust the placement of the clothes using *surface dragging*. The user clicks and drags the clothes using the right mouse button. This operation is superficially the same as the typical dragging operations in interactive real-time cloth-simulation systems.

In typical cloth-simulation systems [8], a user's dragging operation applies a force to a single vertex, and the system simulates the consequent forces on the rest of the cloth to create a larger-scale effect. This approach (which we call *vertex dragging*) is useful for adjusting very local cloth shape, but is inconvenient for more global cloth manipulations such as revolving a skirt around a body or pulling the sleeve upwards (see Figure 6). Single vertex dragging induces large deformations near the vertex, since other vertices resist the motion because of the friction against the body. In addition, vertex dragging can only *pull* the cloth and cannot *push* it – if the user tries to push the cloth, flips and folds result near the vertex. Finally, vertex dragging is often implemented as an unconstrained 3D movement and is therefore difficult to control with 2D input devices. Some commercial systems allow the simultaneous modification of multiple points, possibly with an attenuation factor to ease out the deformation, but these vertices are all moved in the same direction in 3D space and so it is still cumbersome to move clothing along the body surface.

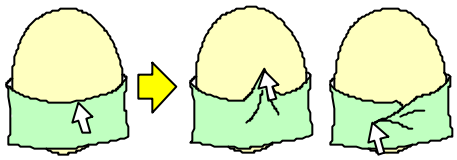


Figure 6: Limitations of conventional vertex dragging: it causes large stretch and folds instead of the desired upward slide or horizontal rotation of the entire cloth.

Our surface-dragging operation explicitly propagates the user's input motion across the clothes along the body surface to create a global effect. For example, if the user drags a vertex upwards, the system explicitly moves the surrounding cloth vertices upwards at the same time, and if the user drags the front side of a skirt to the right, surface dragging actually rotates the skirt horizontally around the body (Figure 7). Just as in wrapping, we apply a relaxation step after each dragging step to maintain the basic cloth constraints.

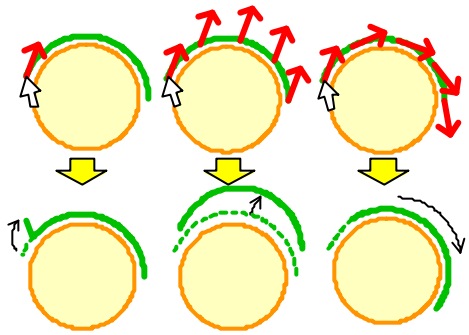


Figure 7: Various dragging approaches: vertex dragging (left), rigid dragging (center), surface dragging (right).

Surface dragging is constrained to directions parallel to the associated body surface and the user cannot pull the clothes away from the body[1]. The mouse cursor is projected onto the tangent plane to the body surface at the clickpoint. This makes dragging with a 2D input device much simpler and easier than completely free 3D motion. This constraint caused us no practical problems during typical operations (Figure 8).

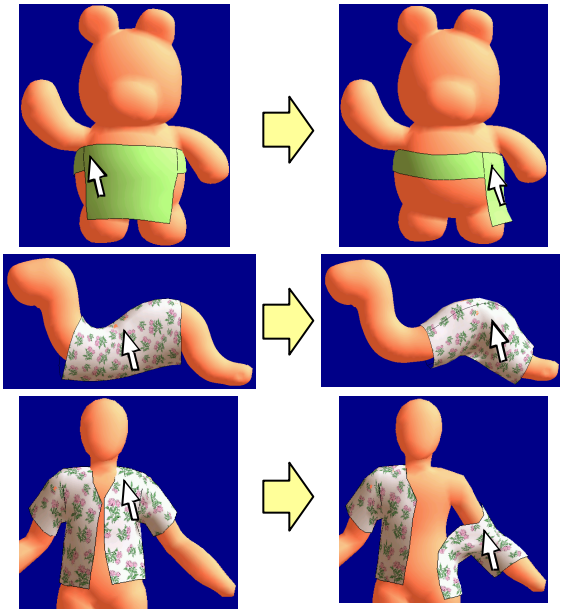


Figure 8: Surface dragging. The third example uses two pushpins on the back to block propagation. All examples run at a few frames per second.

Pushpins

The user can control the behavior of the clothes during surface dragging and the subsequent relaxation steps by putting *pushpins* on the clothes (Figure 9). The user places or removes a pushpin by clicking on the clothes with the left mouse button. A pushpin fixes a cloth vertex at some position on the body, thus helping in local cloth adjustments by blocking the propagation of motion during surface dragging. Pushpins are especially useful because dragging is a single-mouse operation—pushpins are often necessary to perform operations that require two hands in the real world.

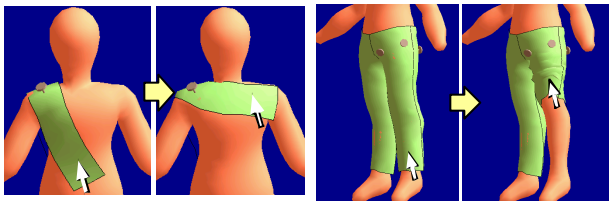


Figure 9: Surface dragging with pushpin.

ALGORITHMS

This section describes the algorithms for calculating the cloth configuration during manipulations. First the immediate goal position for each cloth vertex is computed in response to user input, and then relaxation steps adjust the positions to preserve basic cloth constraints such as prevention of penetration and limiting stretch. These two phases are actually closely integrated, but we describe them separately for clarity. We first discuss how to calculate immediate goal positions for cloth vertices during wrapping and surface dragging, and then describe how to preserve the cloth constraints.

The body and the clothes are represented as standard triangular meshes, and each cloth edge has an associated *rest length*. The parameters defining the behavior of the algorithms must be set accordingly to the characteristics of the target polygonal models. Our current implementation uses body models of 1.0~2.0 units in height and width that consist of a few thousand polygons.

Wrapping

Wrapping tries to put the clothes on the body so that the freeform marks on the clothes match the corresponding marks on the body. We begin by triangulating the cloth[2], since each cloth piece is initially a single polygon. Then we construct a single continuous mesh structure by combining the pieces of cloth according to the sewing constraints. Finally, we compute the geometry of the clothes by building a piecewise-linear map f from this mesh to 3-space by mapping the vertices one at a time (Figure 10). We'll say that an edge is *mapped* if both its vertices have been mapped, and that a triangle is mapped if all three of its vertices are mapped. The steps are

[1] Note that gravity can pull the clothes away from the body during the relaxation steps.

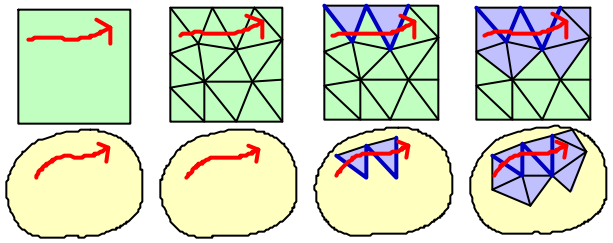
[2] We start with a constrained Delaunay triangulation and refine it iteratively, as in the "skin" algorithm [19]. An alternative triangulation algorithm [21] could work as well. The triangles must be small enough faithfully to represent the geometry of the body. We use a triangle edge length of 0.07~0.08 units.

1 Paste the clothes around the marks by defining f on the *root edges*, i.e., edges that cross marks on the clothes.

2 Grow the clothes by repeating the following process until all triangles are mapped:

(1) Find triangles with one unmapped vertex;
(2) Order the unmapped vertices;
(3) Map the vertices, performing relaxation after each.

To define f on a vertex v of a root edge e that crosses a mark m on the cloth corresponding to a mark M on the body, we first find the point p where e intersects m. The point p is some fraction of the way along the mark m; we find the point P that's a corresponding fraction of the way along M (we'll call this a *proportional correspondence* between m and M). The edge e makes some angle a with the tangent vector to the mark m at p, and the vertex v is some distance d from the mark m. We construct a ray tangent to the body at P with angle a to the tangent to M at P (see Figure 11) and walk[3] a distance d in this direction; the resulting point is defined to be $f(v)$. If v is an endpoint of multiple root edges, this calculation is carried out for each edge and $f(v)$ is defined to be the average of the result.[4]



a) user input b) triangulation c) root edges d) growing the mesh

Figure 10: Overview of the wrapping process.

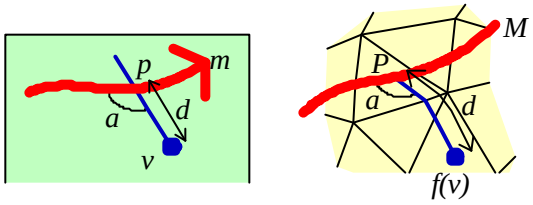


Figure 11: Mapping root-edge endpoints.

The remaining task is to grow the cloth mesh starting from the already mapped root edges (Figure 10d). For each triangle that is already *mapped*, we check whether the vertices around the triangle (i.e., the vertices of triangles that share an edge with this one) are already mapped or not.

[3] The "walk" in a given direction is found by traversing the mesh, as shown in Figure 11.

[4] The average is computed in space and the skin algorithm's surface tracking is used to find a closest surface point to this result.

If a vertex is not mapped yet, the system marks it as *ready* (Figure 12) and places it in a priority queue with priority given by mesh distance to the nearest mark[5]. The system dequeues the lowest-priority vertex, maps it to the 3D body space (see below), and updates the queue based on the result. This process is repeated until all cloth vertices are mapped. We apply this procedure to the *merged* cloth mesh; sewn edges are treated as a single edge and the clothes grow across the sewn edge as usual.

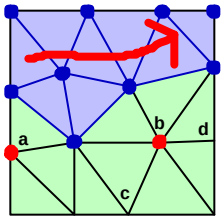


Figure 12: Growing process. Blue points and triangles are mapped vertices and triangles, red points are ready vertices. Vertex *a* has one parent triangle and *b* has two parents. Vertices *c* and *d* will become ready once *b* is mapped.

The position of a newly mapped vertex in the 3D space is calculated based on the triangles (which we call *parent triangles*) around the vertex that are already mapped. We'll describe the computation done for each parent triangle; the final value is the average of the results.

Let P be the parent triangle, sharing an edge e with another triangle T whose other vertex v needs to be mapped. Because the vertices of P are already mapped, there's a plane H in 3-space that contains $f(P)$. Consider the segment $f(e)$ in the plane H oriented so that $f(T)$ lies to its right (Figure 13). The basepoint of an altitude from v to e lies somewhere along the line containing e; the distance from v to this point is some number d. Find (using a proportional mapping) the corresponding point on the line containing $f(e)$, and go a distance d to the *right* of the directed segment to find the point $f(v)$ where the vertex v is mapped relative to this parent.

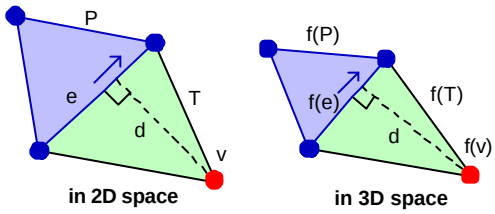


Figure 13: Calculating the position of a newly mapped vertex based on a parent triangle. The vertex is placed on the plane that contains the parent triangle in the 3D space.

[5] The *mesh distance*, computed by finding the shortest sequence of edges between two points, is used instead of geodesic distance because it can be computed quickly.

After each vertex is mapped, we apply a relaxation process (described below) that tries to keep the edge lengths of the already mapped mesh close to the corresponding "rest" lengths and prevents flipping of triangles on the body (i.e., tries to ensure that the map is orientation-preserving).

This growing algorithm extends the clothes using the already mapped vertices as guide and ignoring the body as long as the cloth does not collide with the body (the relaxation step detects and fixes such intersections). An alternative is to grow the cloth using the body as guide (see Figure 14). We experimented with this, but rejected it because of undesirable artifacts such as that shown in Figure 15. In general, body-guiding tends to create visually distracting folding and overlapping that are difficult for relaxation process to fix.

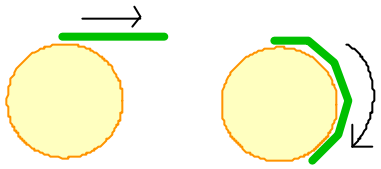
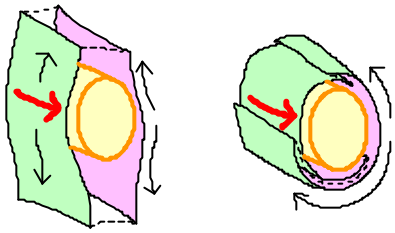


Figure 14: Two possible approaches for growing: current approach (left) and growing-on-the-body approach (right).



a) current approach b) growing-on-the-body approach

Figure 15: Putting a loose sleeve around an arm. The growing-on-the-body approach causes undesirable folds.

Surface Dragging

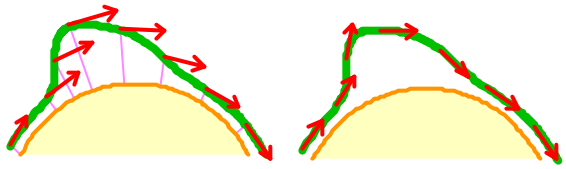
As discussed before, surface dragging explicitly propagates the dragging effect across the cloth vertices (Figure 7). At the beginning of a dragging operation, the system constructs a dependency graph whose root is near the click-point. Then as the user drags the grabbed vertex the system propagates the motion vector across the cloth according to the dependency graph. The system inserts a relaxation step after each dragging step.

Dependency graphs for surface dragging look similar to those for wrapping, but have two major differences. First, the dependency graph for surface dragging starts from a single root *vertex*, while that for wrapping starts from multiple root *edges*. Second, each vertex is dependent on multiple *parent vertices* in surface dragging, while each vertex is dependent on *parent triangles* in wrapping.

We build the dependency graph incrementally: we start with the grabbed vertex as the root, and insert its neighbors in a priority queue, with priority given by the distance to the root. Distances are calculated based on the edges' target rest lengths (i.e., the lengths of the corresponding edges in the cloth mesh). Now vertices are extracted from the priority queue and processed until the queue is empty. To process a vertex v, we first insert it into the graph and then examine its neighbors: if the neighbor vertex n is already in the graph, we add a directed edge from n to v. If not, we add the length of the edge nv to the priority of v to get a priority for n, which we insert in the queue. This process generates a directed acyclic graph of vertices with the grabbed vertex as the root.

We now describe how to propagate a motion vector from the root node to all other nodes. Just as in the wrapping algorithm, we compute a motion vector for each vertex from the motion vector for each of its parent vertices and then average the results to get the true motion vector (which may be zero).

There are two ways to propagate the motion over the clothing, one based on the body geometry and the other based on the cloth geometry. Figure 16 illustrates the difference between the two. The first approach works better when the clothes are close to the body surface, but causes undesirable motion when the clothes are far from the body. The second approach works better when the clothes are away from the body, but can be unstable because of its recursive nature, especially if significant wrinkles are present. Our current implementation uses the first approach because of its stability. In addition, the system slightly pulls the clothes near the body towards the body at each surface dragging step to make it stable (currently, a cloth vertex moves towards the nearest body surface so as to halve the distance when the distance is less than 0.036).



a) along body geometry b) along cloth geometry

Figure 16: Two possible approaches to surface dragging. Our current implementation uses the first one.

We now describe how to compute a child vertex's motion vector from that of its parent vertex. We first define a local coordinate system for each vertex. We use the normal vector of the corresponding body surface as the z-axis; we let $\vec{u}$ be a unit vector along the directed edge from the parent to the child, and use $\vec{u} - (\vec{u} \cdot \vec{z})\vec{z}$ as the direction of the x-axis and the cross product of the two as the y-axis. The motion vector for the child vertex is defined as the parent's motion vector mapped from the parent's coordinate system to the child's.

Pushpins

Pushpins provide additional control for surface dragging. A naïve approach to implementing them is simply to fix the pinned vertex and move the other vertices normally, but this generates large distortions around the pinned vertex (Figure 17).

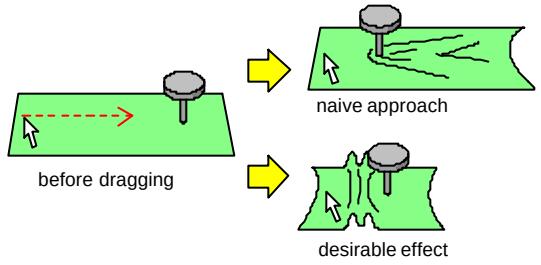
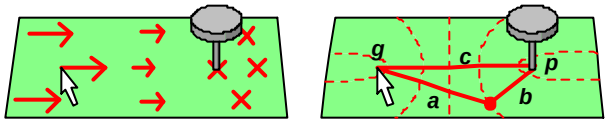


Figure 17: Pushpin effect. Naïve approach causes distortion.

To obtain the desirable effect in Figure 17, we attenuate the dragging vectors at the cloth vertices near the pushpin and diminish them on the other side of the pushpin (Figure 18a). This is done by calculating an *attenuation ratio* for each cloth vertex at the beginning of surface dragging; for each cloth vertex *v*, the system computes the mesh distance *a* to the grabbed vertex *g*, and the mesh distance *b* to the pinned vertex *p* (see Figure 18b). The system also computes the distance *c* between *g* and *p*.



a) attenuation of vectors b) calculation of distances

Figure 18: Calculating the attenuation ratio.

Given these distances, the attenuation ratio for the vertex is defined as

$$\begin{array}{lll} 1 & \text{if} & a - b = -c, \\ (c - a + b)/2c & \text{if} & -c < a - b < c, \\ 0 & \text{if} & c = a - b \end{array}$$

where 1 means full motion and 0 means no motion.

If multiple pushpins are used, the system calculates the attenuation ratio for each pushpin and uses their minimum[6]. The user can conveniently block the surface dragging effect by putting in a few pushpins in a row.

Pushpins are also important in controlling rewrapping (see Figure 4). Rewrapping first removes the cloth triangles from the 3D scene and then pastes them back around newly placed marks. But the removal of triangles is blocked by the pushpins – the system does not move vertices whose distance from the mark is greater than the distance between the mark and the pushpins.

Keeping Clothes on the Body

We now describe the algorithms for maintaining basic cloth constraints during wrapping and surface dragging. This section describes the algorithm for handling cloth-body collision, and the next section describes the algorithm for preventing excessive stretching and folding.

Collision detection is the most time-consuming part in cloth simulation in general [14,23]. In addition, exact collision detection can impede placing cloth in the intended position. To achieve real-time operation, we ignore cloth-cloth collision and handle cloth-body collision in a limited way, by simply preventing cloth vertices from sinking into the body at each step and ignoring collisions between cloth edge and body edge. The system also ignores possible collisions during transitions. This simplified strategy obviously exhibits flaws in some situations, but it is fast and works well for our purpose.

To detect collisions between a cloth vertex and the body surface efficiently, the system keeps track of the nearest point on the body surface (*track point*) for each cloth vertex (this is the strategy used in the skin algorithm [19] for tracking the nearest skeleton surface for each skin vertex). Whenever a cloth vertex is moved, the system updates its track point by locally searching the body surface (Figure 19 left). Given the track point, detecting collision is a straightforward. If the cloth vertex is inside the body surface, the system pushes the cloth vertex back to above the body surface (Figure 19 right). The system actually keeps the cloth vertices a bit away from the body so that cloth edges do not penetrates the body surface (the current offset is 0.012). A vertex also shares information with its immediate neighbors so as to jump from a local solution to a distant solution (Figure 20). This migration feature is important when a garment spans separate body regions such as an arm and a torso.

This simple approach cannot detect collisions with body parts approaching from above or with separate body parts that were not covered by the cloth before. This causes no practical problems in our experience, but the current simple approach must be extended to handle more complex cases (see "Additional Algorithm Details" section).

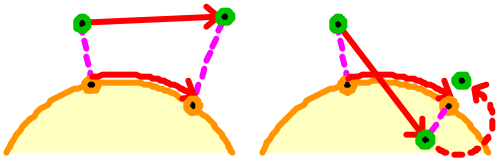


Figure 19: Each cloth vertex is associated with the nearest body surface. A vertex inside of the body is pushed back to the body surface.

[6] It is possible to use a blend function or the product of pin attenuations, but our simple approach shows satisfying results and we opt for the simplicity.

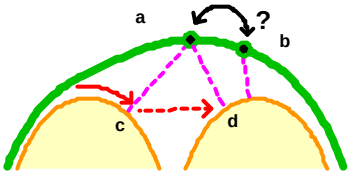


Figure 20: Cloth vertices share information with their neighbors. This enables vertex *a* to find the true solution *d* instead of being stuck with local solution *c*.

Relaxation Steps

Relaxation steps are inserted during wrapping and surface dragging to keep the clothes visually plausible by preventing excessive stretch and folds. Note that the purpose of this relaxation step is to move the cloth towards a class of desirable static configurations. Our goal is to add useful behavior to cloth so as to help the user put clothes on characters, *not* to mimic physically realistic behavior. For example, our relaxation steps automatically unfold flipped clothes, which does not happen in the real world.

A relaxation step has four parts. First, we try to make each edge's length closer to its rest length to prevent stretch and shrinkage. Second, we try to recover flipped triangles to prevent folds. Third, we try to flatten the cloth at each edge of triangle; this corresponds to a dihedral-angle spring and helps generate attractive wrinkles. Finally, we mimic the effects of gravity and friction.

Preventing stretch and flip The system addresses the first two goals simultaneously by adjusting vertex positions so that each triangle T recovers its rest shape (called the *reference triangle*) on the body surface. The reference triangle is uniquely defined by the rest length of the edges. The system places a copy U of the reference triangle as close to T as possible, and moves each vertex of T towards its corresponding vertex in this copy of U (Figure 21). U is placed in a plane (described below) with the centers of gravity, O and O', of T and U aligned, and is rotated as follows. The system computes B'' by rotating B by $\angle B'OA'$ around O, and computes C'' by rotating C by $\angle C'OA'$ around O. The system rotates U so that $\overrightarrow{OA'}$ parallels $\overrightarrow{OA}+\overrightarrow{OB''}+\overrightarrow{OC''}$. A similar technique is used in automatic texture coordinate optimization [18].

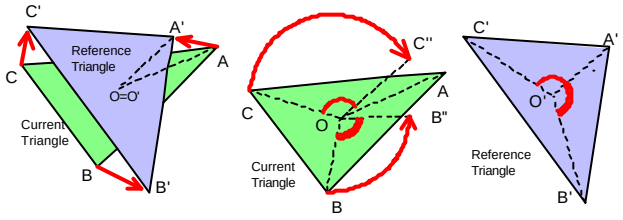


Figure 21: Matching a triangle and its reference triangle.

We've found that this triangle-based strategy works faster than an edge-based strategy (e.g. [8]) and generates better results for our purpose. In addition, it automatically recovers flipped triangles if we place the reference triangle front-face up. "Face up" is determined by a "temporary normal vector." The temporary normal is the body surface normal when the cloth is near the body surface (distance < 0.012), but is the cloth surface normal when the cloth is far from the body (distance > 0.08); the normals are blended in the intermediate region. The triangle-based relaxation is done on the plane perpendicular to this temporary normal: the system projects T to that plane, applies the above method above, and then moves the vertices according to the resulting vectors (which are parallel to the plane).

Flattening the cloth We flatten the cloth by moving vertices so as to make the dihedral angle at each edge closer to 180 degrees. We compute the vectors shown in Figure 22 for the four vertices associated with each edge; the sum of these is then applied to the vertices. This corresponds to the dihedral-angle spring found in typical cloth simulations [2].

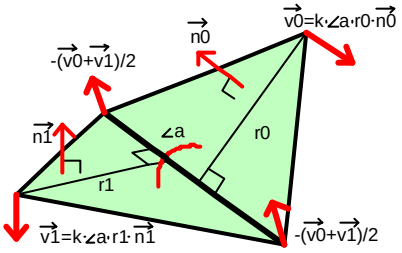


Figure 22: Each edge on a ridge moves four adjacent vertices to become flat.

Gravity and friction To mimic the effects of gravity, we move each cloth vertex downward by a predefined amount unless it collides with the body surface (i.e., we make clothes fall at a constant speed). Friction is mimicked by not allowing any vertex to be moved if 1) the vertex is in contact with the body surface, 2) the requested motion vector heads downwards with respect to the underlying body surface, and 3) the requested motion vector is smaller than a predefined threshold.

ADDITIONAL ALGORITHM DETAILS

This section describes some further implementation details. The features described are optional: one can manipulate clothes reasonably with the basic algorithms alone, but these features help make the system robust and improve the user experience.

Adaptive Subdivision

As discussed in the previous section, we prevent the vertices from sinking into the body but do not prevent edges from sinking into the body. This works well when the underlying body surface is reasonably flat, but causes serious problems for high-curvature regions such as arms and legs. The body surface appears on top of the clothes and is very distracting.

This is essentially an aliasing problem, and our solution is to adaptively change mesh resolution according to the

curvature of the body surface. A cloth edge is automatically split when it intersects the body, and restored when it no longer intersects the body (the original edge no longer exists in the mesh if it is split, but the system remembers the original edge information). We use the $\sqrt{3}$ subdivision scheme [16] because it allows edgewise split/merge and generates reasonable mesh patterns (Figure 23). Our current implementation allows only one subdivision step for simplicity, and this hides most problems sufficiently.

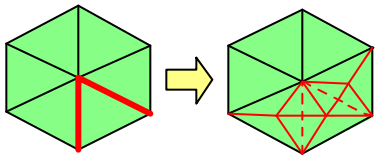


Figure 23: Splitting two edges with $\sqrt{3}$ subdivision.

We use Figure 24 to describe how the system decides whether to split an edge. Here, the system needs to know whether edge *AB* intersects the body. It is too expensive to do precise collision detection by traversing the body surface, so the system performs an approximate computation using local information, that is, the locations of *A*, *B*, and *P*, where *P* is the nearest point on the body to *A*, which is always available from the "skin" algorithm. It is obviously not possible to detect actual collision, so we approximate it by testing collision with a sphere of radius L = length(*AB*) that is tangent to the surface at *P*. The test is approximated by $(d + L)\sin q < L$ and $(d + L)\cos q < L$; if both inequalities hold, we split the edge *AB*. The justification for using L as the body's local radius is that it provides a minimum radius that we must worry about. If the body radius is actually larger than L, the "vertex is always on the body surface" constraint approximately guarantees that the edge does not sink below the surface. The system performs the equivalent test at *B* as well.

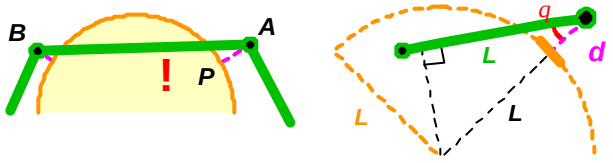


Figure 24: Testing an edge for collision with an approximating sphere.

Collision Detection with Bones

Adaptive subdivision effectively prevents most edge-to-edge penetration, but excessive movement can cause the clothes to penetrate the body. We can ignore small amounts of "sinking" because the relaxation process gradually recovers from the error, but the system cannot recover from significant topological errors such as the cloth penetrating the body all the way from one end to the other. This happens typically where thin parts such as a neck or an arm stick out from the body (Figure 25 center). To prevent this, we implemented collision detection against simple "bone structures" (Figure 25 right). A *bone* is a simple edge defined by two end points, and collision with all bones is checked whenever a cloth edge is moved. If a collision occurs, the system pushes the cloth edge back to prevent penetration. For the human body in Figure 2, we used six bones.

Figure 25: Collision detection against simple bones.

IMPLEMENTATION AND RESULTS

The current prototype system is implemented in Java™ (JDK1.4), and uses directX7 for 3D rendering. Figure 26 shows some clothing designed using the system. The clothes have a few hundred triangles and the system maintains reasonable frame rates during surface dragging on a high-end PC (AMD Athron™ 1.5GHz).

We have begun an informal user study. It took approximately 20 minutes before a user started using the system fluently under our supervision. The last image in Figure 26 was created by the test user. It took a while for the user to learn the peculiar behavior of the clothes in our system. The user tended to drag the clothes long way in a single interaction, making the system unstable; clothes must be moved gradually towards the goal position instead. It's also necessary to release the mouse occasionally during the dragging so that relaxation steps can dissolve the accumulated distortion. The user also had difficulty in designing the clothes of an appropriate size. It would be helpful if one could adjust the size of the clothes after putting them on the characters.

Figure 26: 3D characters in various clothes.

LIMITATIONS AND FUTURE WORK

Our current system has several limitations: our techniques

are designed specifically for clothing a character and not for manipulating clothes away from a body. We support only a single layer of clothes on a body, although we plan to extend the system to support multilayer clothes. We also plan to support explicit folding of clothes (e.g., collars). But to support these, we need to track the nearest object on top of each cloth vertex as well as the nearest object under the vertex.

Cloth-cloth collision is ignored in the current implementation. Although we believe that this is a reasonable decision given current processor performance, we need to incorporate cloth-cloth collision detection in the future. That will let us explore more interesting cloth-manipulation techniques such as tying a tie.

Wrinkles are an important part of clothes design [13]. We plan to develop interaction techniques for explicitly placing wrinkles on clothes. For example, it might be useful for the system to automatically adjust global cloth configuration so that wrinkles appear where the user paints freeform marks.

Some basic interface improvements would be very useful: An obvious extension is to let a user edit the clothes in 2D and 3D space simultaneously [8]. We are considering several operations such as cutting, stitching, and resizing. And our dragging operation should probably interleave "relaxation steps" during long drags.

We believe it is reasonably easy to incorporate our cloth-manipulation techniques into existing 3D graphics systems because we use standard triangular mesh structures for the cloth and the body. A potential difficulty is finding appropriate values for the many ad hoc parameters in our algorithms (the current values are chosen as the result of many experiments). They must be carefully adjusted according to object geometry and the user input.

ACKNOWLEDGEMENTS

We would like to thank the Brown University computer graphics group for thoughtful discussions, and the CMU stage3 research group, especially Dennis Cosgrove, for allowing us to use their Jalice scenegraph. We also thank reviewers for their constructive critique.

REFERENCES

1. Asahi AGMS, www.agms.co.jp
2. D. Baraff and A. Witkin. Large steps in cloth simulation. *SIGGRAPH 98 Conference Proceedings*, pages 43-4, 1998.
3. D. Baraff, PIXAR. Personal communication, 2001.
4. T. Beier and S. Neely. Feature-based image metamorphosis. *SIGGRAPH 92 Conference Proceedings*, pages 35-42, 1992.
5. D.E. Breen, D.H. House, and M.J. Wozny. Predicting the drape of woven cloth using interacting particles. *SIGGRAPH 94 Conference Proceedings*, pages 365-72, 1994.
6. M. Carignan, Y. Yang, N. Magnenat-Thalmann, and D. Thalmann. Dressing animated synthetic actors with complex deformable clothes. *SIGGRAPH 92 Conference Proceedings*, pages 99-104, 1992.
7. F. Dachille IX, J. El-Sana, H. Qin and Arie E. Kaufman. Haptic sculpting of dynamic surfaces. *Proc. of Interactive 3D Graphics 1999*, pages 103-110, 1999.
8. M. Desbrun, P. Schroder, and A. Barr. Interactive animation of structured deformable objects. *Proc. of Graphics Interface '99*, pages 1-8, 1999.
9. DressingSim, www.dressingsim.com
10. Gerber Technologies, www.gerbertechnology.com
11. T. Igarashi and D. Cosgrove. Adaptive unwrapping for interactive texture painting. *Proc. of Interactive 3D Graphics 2001*, pages 209-216, 2001.
12. T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3D freeform design. *SIGGRAPH 99 Conference Proceedings*, pages 409-416, 1999.
13. S. Hadap, E. Bangerter, P. Volino, N. Magnenat-Thalmann. Animating wrinkles on clothes. *Proc. of the Conference of Visualization '99*, pages 175-182, 1999.
14. D. House and D. Breen. *Cloth Modeling and Animation*. AK Peters, 2000.
15. J. Hultquist. A virtual trackball. *Graphics Gems* (ed. A. Glassner). Academic Press, pages 462-463, 1990.
16. L. Kobbelt, $\sqrt{3}$ subdivision, *SIGGRAPH 2000 Conference Proceedings*, pages 103-112, 2000.
17. Lectra, www.lectra.com
18. H. Malan, Righthemisphere Inc. Personal communication, 2001.
19. L. Markosian, J.M. Cohen, T. Crulli and J.F. Hughes. Skin: a constructive approach to modeling free-form shapes. *SIGGRAPH 99 Conference Proceedings*, pages 393-400, 1999.
20. Maya Cloth, www.aliaswavefront.com
21. J.R. Shewchuk. Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. *First Workshop on Applied Comp. Geometry Proc.*, pages 124-133, 1996.
22. 3ds MAX, www.ktx.com
23. P. Volino, N. Magnenat-Thalmann, *Virtual Clothing Theory and Practice*, Springer-Verlag, 2000.
24. J. Weil. The synthesis of cloth objects. *Computer Graphics (Proc. of SIGGRAPH)*, Vol. 20, No. 4, pages 49-53, 1986.
25. R.C. Zeleznik, K.P. Herndon, and J.F. Hughes. SKETCH: an interface for sketching 3d scenes. *SIGGRAPH 96 Conference Proceedings*, pages 163-170, 1996.