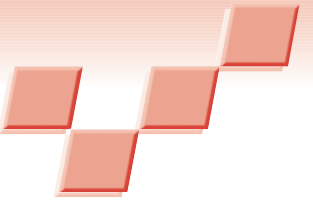
A sketch-based interface for clothing virtual characters

Emmanuel Turquin, Jamie Wither, Laurence Boissieux, Marie-Paule Cani, John F. Hughes

A Sketch-Based Interface for Clothing Virtual Characters



Emmanuel Turquin, Jamie Wither, Laurence Boissieux, and Marie-Paule Cani
Grenoble University

John F. Hughes
Brown University

This interactive system for garment creation determines a garment's shape and how the character wears it based on a user-drawn sketch. The system then uses distances between the 2D garment silhouette and the character model to infer remaining distance variations in 3D.

Despite sometimes questionable user friendliness, software solutions for 2D image authoring and editing are now a widespread commodity among average computer users. Clearly, the same cannot be said of 3D suites; the introduction of a third dimension on intrinsically 2D input and output devices—the mouse and screen—adds a discouraging complexity that only professional artists and a few skilled amateurs can overcome.

To make such tools appealing to a broader audience, researchers have devoted considerable effort to enhancing ease of use and creating intuitive interfaces based on simple, well-understood tasks. To convey 3D shapes, people typically sketch them—or, more precisely, sketch a 2D projection of the shape from which our brains can seamlessly reconstruct a full 3D representation. Sketch-based interfaces seem all the more convenient because most people who design 3D objects and regularly use 3D modeling programs already use sketches and artwork extensively in early phases of the creative process. As Figure 1 shows, this is especially true of fashion designers when they create new garments.

In fashion design, the required step between the initial concept art and the final product is currently cumbersome and requires know-how possessed only by trained designers (see the "Feedback from a Fashion Designer" sidebar for further discussion on what this entails). Fashion design's final product might be a real garment or virtual clothes for virtual actors or video game characters, which we focus on here. To clothe such characters, designers use a range of approaches. For incidental characters, the clothing might be no more than a texture map. For lead characters in feature films, they might use full-fledged physical simulation of detailed cloth models. In between, designers often use simple skinning techniques (that is, a garment is deformed by a skeleton), combined with texture mapping to create clothing that deforms somewhat as the character moves.

Our goal is to make it easy for users to generate simple garments adapted to existing models. We believe that for most people, drawing garments worn by a mannequin—which is how they imagine them—is easier than the traditional pipeline in which they create the 2D patterns needed to sew garments. We therefore developed a sketch-based interface that lets users quickly construct 3D virtual garments on a character model. Our system offers simple yet effective solutions to shape generation and placement, and basic clothing simulation in a resting position. The system is also easy to use: our contributing designer created the entire wardrobe featured in this article in less than an hour.

Designing for virtual characters

Clothing virtual characters entails three problems: designing the clothes (tailoring), placing them on the character (dressing), and making them look physically correct (simulating).

Existing methods

In the tailoring process for human beings, the designer must choose the cloth, fit it to the body, adjust the garment's pattern pieces to fit the model, and sew the pieces into a garment. A virtual character's garment typically has no patterns with which to sew, and is instead represented by a simple polygon mesh that designers construct to fit the body. Currently, constructing such meshes is tedious, even without patterns and stitching. Designers sometimes construct the garment by directly incorporating the cloth mesh into a character's geometric model so that the character has pants but no legs, for example. In this case, physical simulation is impossible, so when a character needs new clothes, designers must largely remodel it.

As an alternative, designers can draw pattern pieces for a garment and position them over the character's naked form, defining stitching constraints and so on. This can be wearisome, however, especially when the character is relatively unimportant. This approach also requires that designers understand how cloth fits over forms,

 Published by the IEEE Computer Society

although the actual pattern-and-stitching information might be irrelevant once tailoring is completed. (In rare cases, the cloth's physical properties—such as whether it was cut on the bias or it resists folding along one axis—might be relevant to a full-fledged physical simulation.) To ease the process, we draw inspiration from previous work on sketch-based interfaces (see the "Related Work" sidebar at the end of the article for more details).

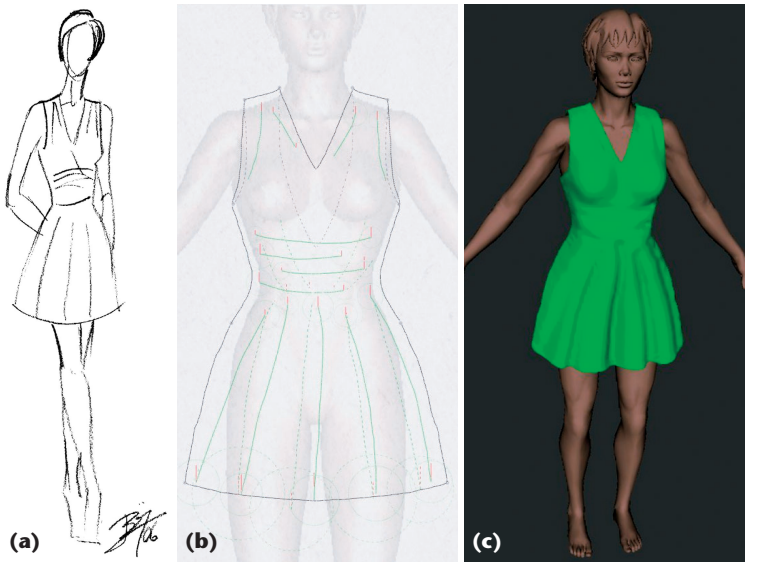
A sketch-based approach

Our approach combines tailoring, dressing, and physical plausibility into a single step to create a mesh that is both visually pleasing and suitable for later complex simulation or skinning approaches. We described a preliminary version of our system elsewhere;[1] here, we present the system in detail and describe several new aspects, including

- generation of complete (back and front) garments,
- fold-sketching,
- graphical user interface improvements, and
- feedback from a seasoned fashion designer.

Our system's two key features are its pleasant user-interaction experience and its method for reconstructing the garment's 3D geometry and placement from a 2D sketch. As in work by Bourguignon and colleagues,[2] our system lets users sketch garments directly on a 3D virtual actor body model. However, our method outputs a full 3D geometry for the garment, using the distance from the 2D garment silhouette to the body model to infer distance variations between the garment and the character in 3D.

The system performs this reconstruction in four steps. First, it automatically classifies the 2D garment drawing's contour lines as either silhouettes (lines that don't cross the body) or borderlines (lines that cross the body). Next, it computes a distance-to-the-body value for each point of a silhouette segment and uses these distances to determine desired point-to-body distances for the borderlines. It then propagates this distance information in 2D to find desired point-to-body distances, which it uses to determine the garment's 3D position. When the drawing includes fold strokes indicating fold location and placement, the system adjusts the garment's distance values so that the established level moves closer to or farther from the body.



1 A sketched-based approach to 3D cloth modeling. (a) Our designer created a traditional fashion design sketch early in the design process to convey her vision on paper. (b) She then reproduced the design using our sketch-based interface. (c) The resulting system-generated 3D garment.

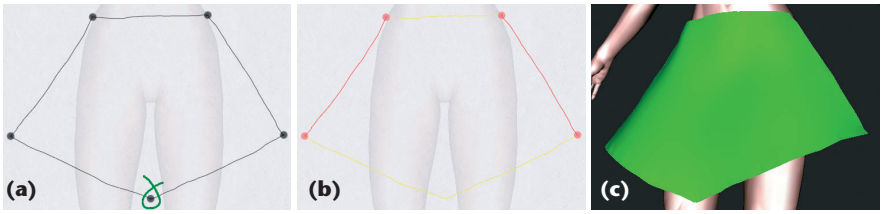
Feedback from a Fashion Designer

Fashion designer Laurence Boissieux has experience in creating both real and virtual garments. She produced the sketch in Figure 1 and most garments featured in this article. Boissieux offers the following commentary on the system.

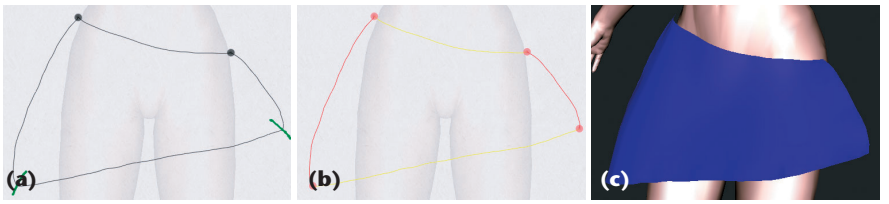
For a designer, the most natural way to create new fashion is with a simple sheet of paper and a good old pencil. But designers must keep up with the times and enter the digital age. So far, the existing tools for creating clothes in most 3D modeling systems are based on a quite complex sequence of steps. The first is to draw flat panels, which presupposes a strong knowledge of pattern making. Not everyone has such skills; in the fashion industry, pattern making is a distinct job and is not typically handled by the designers themselves. Once the panels are created, users must arrange them correctly in space around the body. The next step is to specify seams, and finally to launch a physical simulation that will pull the different panels toward each other. Assuming users choose all parameters well, given a lot of trials—and time!—they'll get a 3D garment.

The real strength of the sketch-based interface is—as the name claims—that it's true to the designer's natural gestures. It replaces paper and pencil with a graphic tablet, which is strictly equivalent. And, above all, designers are only expected to draw—something they're used to doing. The system bears a likeness to reality in its use of metaphors: to remove a line, you just scribble on it. Users can draw and redraw strokes until they're satisfying. Generating the 3D shape is as simple as clicking a button, and the computation is efficient and quick. The result is nearly instantaneous—no need to wait for endless iterations.

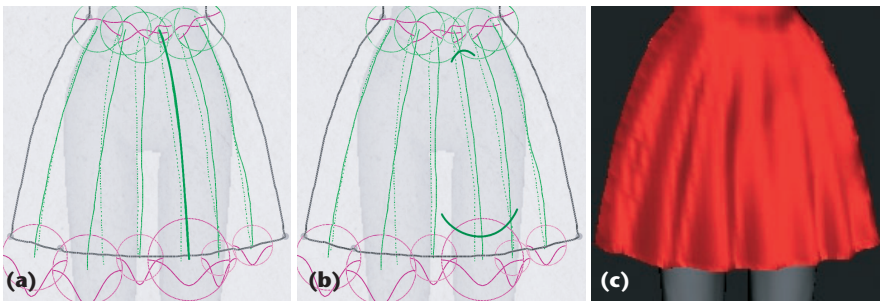
This speed lets designers go back and forth from the sketch to the garment, making rapid changes. The same drawing metaphor lets the designer add folds. Although the fold shapes are somewhat automatic, they're aligned on free-hand curves and thus look natural. This is another great feature that painlessly enriches the models and gives them more realism at no cost. Another interesting aspect is that even neophytes can play designer: the system doesn't require any particular know-how. It's easy to get into it and very intuitive. My wish as designer would be to see this system included in well-known 3D modelers so that developers could create nicely shaped garments quickly, easily, and directly—and then have more time to focus on animation!



2 Example user interaction. (a) After the user draws a few lines in contour mode to indicate the skirt's shape, the system's corner-detector detects a breakpoint that the user doesn't want. The user therefore deletes the breakpoint by drawing a circle around it. (b) The system classifies the remaining lines and displays silhouette lines in red and borders in yellow. (c) The user requests a reconstruction, and the system displays the inferred surface.



3 Adding new breakpoints. The user had outlined a skirt in contour mode without sharp corners at the bottom, so the system's corner-detector failed to insert breakpoints. (a) The user draws short strokes (in green) that cross the contour to indicate the need for new breakpoints. (b) The system inserts the new breakpoints. (c) The reconstructed skirt.



4 Folding mode. (a) The user draws a few fold lines (thick green lines) that correspond to ridges or valleys on the garment's surface. (b) The width of the u-shaped gesture crossing each end of a fold line determines the fold's width; its depth determines the fold's depth. The system indicates both width and depth using a pink-circled Gaussian profile at the end of each fold line. (c) The system adds the folds to the skirt.

The sketch-based interface

A central objective of any sketch-based interface is to give users the benefits of a traditional pencil-and-drawing-board, while providing as much new functionality as possible. To achieve this, we seek to minimize modifier keys and buttons, and UI modes (such as *Caps Lock*). Our system uses only one button (which lets users employ devices such as graphical tablets or tablet PCs), a few mode sets, and an optional symmetry mode.

The first mode set lets users toggle between contour mode (sketching the garment's contours) and folding mode (sketching the garment's folds). Contour mode is the default, and is the starting point for any garment creation. Folding mode is subsidiary. In the second mode set, users choose between applying strokes to the garment's front, back, or both (the default behavior). Finally, an optional symmetry mode helps users draw vertically symmetrical garments.

Although we believe our interface provides an intuitive way of designing garments, the major contribution of our approach is in the method for generating a 3D garment from a finished 2D sketch. The sketching phase in itself clearly leaves room for improvement; we could push the drawing-board metaphor further by introducing techniques such as clustering and beautification of fuzzy strokes, or oversketching as a way of editing the current sketch.[3]

Typical user interaction

To illustrate the system's performance, we describe a typical interaction in which a user, Edna, sketches a skirt on a female model.

Contour mode. As Figure 2a shows, Edna first draws a line across the model's waist (indicating the top of the skirt), then draws a line down the side (indicating the skirt's silhouette). Next, she draws a line across the bottom in a V shape to indicate that the skirt's front should dip down. She finishes with the other side. The system then applies a simple corner-detection process—based on the 2D curvature's variation—to break the sketch into parts. The system accidentally detects one extra corner (at the bottom of the V shape), which Edna deletes using a deletion gesture. She could also add new breakpoints, but none are necessary here. (As we describe later, breakpoints determine the garment's global 3D position with respect to the body, and thus play an important role in the 3D positioning process.) As Figure 2b shows, the system classifies the two side lines as silhouettes, and the other lines as borderlines.

Edna now presses a button to see the garment that the system has inferred (see Figure 2c); almost instantly, the system displays a surface that matches the drawn constraints, but adapts to the underlying form's shape (see the waistline, for example). Sometimes, the system's breakpoint-inference fails to detect all the points the user wants; in this case, she can make a gesture (see Figure 3) to add new breakpoints.

Folding mode. Now that Edna is satisfied with the skirt's global shape, she decides to add a few folds to obtain a more physically plausible 3D surface. To do this, she simply switches to folding mode and draws strokes that mark the presence of either ridges or valleys. She can also specify the folds' width and amplitude in an intuitive way (see Figure 4).

Front/back modes. By default, the user's strokes affect both the garment's front and back parts. Typically,

the two views share most lines. This is always true for silhouettes, which by definition join the front and back parts; it's often true of borders as well. However, our system also lets users edit front and back borders independently by toggling to the appropriate mode—so long as the contour remains closed (see Figure 5). To avoid confusion, when borders differ, the system renders the borders belonging to the current view with a continuous stroke, while others appear dashed. There is no constraint on how users edit the folds, but it's usually best to generate different folding for the front and back parts.

Vertical symmetry. Garments often exhibit vertical (that is, left to right) symmetry. Consequently, the system has a mirror mode in which only half the canvas is active; the other half automatically reproduces mirrored versions of the strokes. When a stroke crosses the symmetry axis, the system cuts it at the intersection and joins the left and right parts. Users can deactivate the symmetry mode by pressing a key.

Gestural interface components

The system interprets the user's marks as gestures; in contour mode, it defaults to silhouette and borderline segment construction. As Figure 6 shows, other gestures add classification-process breakpoints, delete breakpoints, delete a segment or segment chain, and clear all segments.

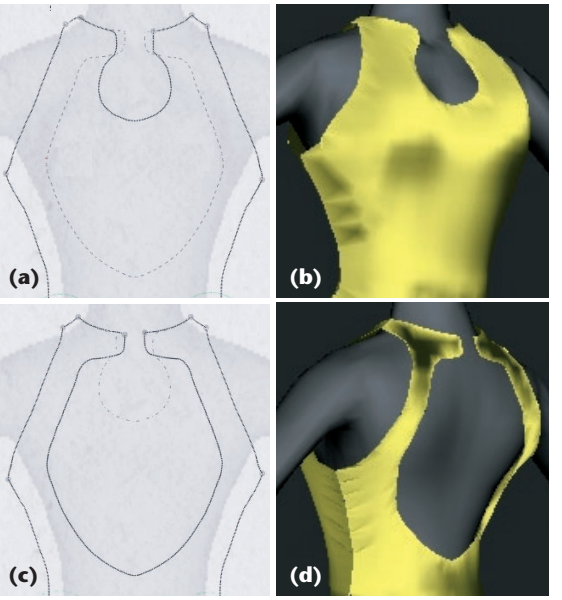
The folding mode default gesture creates a new folding line. Stroke deletion gestures are valid, but because breakpoints are irrelevant in this mode, the system replaces corresponding gestures with other gestures to control fold profiles (see Figure 7 on the next page).

The breakpoint-deletion gesture is similar to the standard proofreader's deletion mark; other deletion gestures require multiple intersections with existing strokes to prevent accidental deletions.

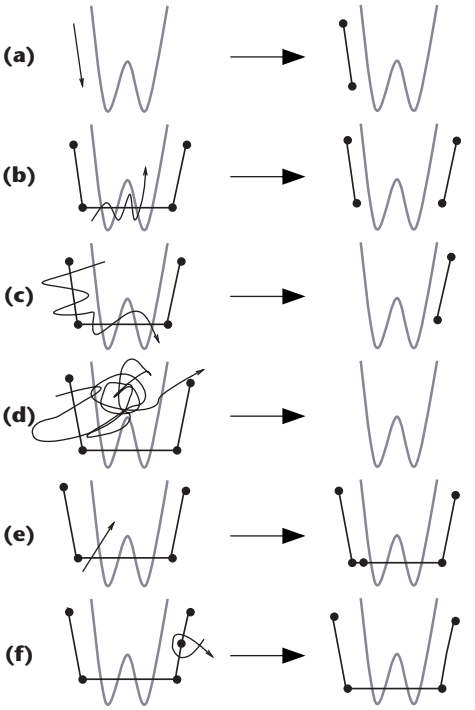
Interpreting garment sketches

Reconstructing a 3D surface from a 2D drawing is clearly an underconstrained problem; however, by considering clothing's special nature we can generate a plausible guess of the user's intentions.

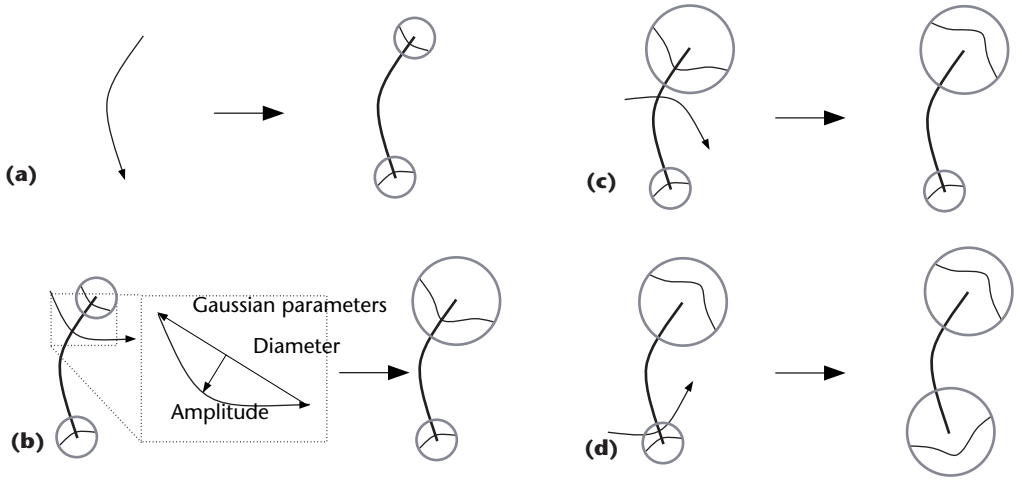
First, we want to find a model position and a 2D projection such that for every $P(x_p, y_p)$ of the image plane, the back-projected 3D ray possesses at most two intersections z_p^f, z_p^b with the model's body surface, because then this property also holds for most one-layer garments. In other words, we need a pose that minimizes surface overlappings (or maximizes the visible surface), and lets us represent the garment with two height fields—one for the front and another for the back. To satisfy this constraint, we chose a standing stance with spread-eagled arms that's viewable from the front or back. In this pose, the only body parts that don't comply with the desired property are the extremities (head, hands, feet), which aren't usually covered by cloth. Second, users should be able to construct the front and back parts with two different viewpoints so they can edit the currently visible part. Because we want the two views to share the same silhouette lines, we must use an orthographic projection.



5 Using front and back modes. (a) When the garment's front is displayed in front mode, the folds in the back are rendered in dashed rather than solid lines. (b) The resulting garment front. (c) The garment back in back mode. (d) The result.



6 Contour mode gestures. The system draws an arrowhead to indicate stroke direction and displays the model using thick lines, prior strokes using medium lines (with dots at breakpoints), and new strokes using thin lines. (a) The user adds a segment and (b) deletes a segment (the user's stroke must intersect the segment at least five times). (c) To delete several segments, the user's stroke must intersect more than one segment and intersect the set of segments at least five times. If the stroke intersects segments from two different chains, the system deletes both chains. (d) To clear all segments, the user's stroke must intersect some segment and count at least 40 self-intersections. (e) The user adds a breakpoint. (f) To delete a breakpoint, the stroke must intersect the segments on either side of the breakpoint and intersect itself once.



7 Folding mode gestures. The system draws new strokes using arrows. (a) Adding a fold (by default, a valley). (b) Modifying the fold's profile at one end (the closest to the intersection). The shape of the user's stroke defines the fold's amplitude and width. A stroke that's convex with respect to the fold's endpoint results in a valley while (c) a concave stroke results in a ridge. (d) By changing the fold's other extremity, the user creates a pure ridge.

For the sake of clarity, we assume that the character's model is aligned with the *xy*-plane, viewed along the *z*–direction. We also assume that clothing silhouettes indicate points where the tangent plane to the cloth contains the view direction and that they usually occur as the clothing passes around the body. This lets us estimate a silhouette edge's *z*-depth as being the *z*-depth of the nearest point on the body. In other words, we want the garment silhouettes to be as close as possible to the body's silhouettes. Moreover, the distance from a silhouette to the body can help us infer the distance elsewhere, since a garment that fits the body tightly in the *xy*-plane will also tend to fit tightly in the *z*-direction, while a loose garment tends to float everywhere.

Algorithm overview

At this point, we assume that we already have a valid 2D drawing of a garment, sketched with our interface. Although we treat the garment's front and back parts separately, they possess the same silhouettes so they're easily joined together to construct a C_0 surface of the whole garment. Therefore, we only describe the process for one side, as follows.

In step 1, the system processes the 2D sketch of the garment as follows:

- Find high-curvature points (breakpoints) that split the contours into segments.
- Let user add and/or delete breakpoints.
- Classify curve segments between breakpoints into borderlines (which cross the character's body) or silhouette lines.

In step 2, the system infers the 3D position of silhouette and borderlines as follows:

- For each breakpoint that does not lie over the body, find the distance from the body, *d*, and set the point's depth, *z*, to the depth of the closest point on the body.
- For each silhouette line, interpolate *z* linearly over the line's interior and use the interpolated *z* values to compute *d*-values (distances to the body in the 3D space) over the line's interior. This allows for straight junctions between the two parts.
- For each borderline, linearly interpolate *d* over the interior to establish a desired distance from the model, and set a *z*-value for each point on the line.

In step 3, the system generates the garment's interior shape as follows:

- Create a mesh consisting of points within the simple, 2D closed curve that the user has drawn, sampled on a rectangular grid in the *xy*-plane.
- Extend the *d*-values, which are known on this grid's boundary, over the interior.
- For each interior grid point (x,y), determine *z*'s value for which the distance from (x, y, z) to the body is *d*'s associated value.
- When appropriate, adjust this tentative assignment of *z*-values to account for garment tension between the character's limbs (we describe this in more detail later).

In step 4, the system applies the garment folds as follows:

- For each fold stroke, determine which grid points (x, y) intersect the fold.
- For each intersected point, linearly interpolate the two parameters (radius, amplitude) supplied by the user at each stroke's end.
- Evaluate a Gaussian within the pixel's neighborhood.
- Apply the resulting displacement map to the *z*-values.

Finally, in step 5, the system tessellates the grid with triangles (clipped to the boundary curves) and renders the triangles.

Precomputing a distance field

To accelerate algorithm steps 2 and 3, we precompute a distance field to the character's model when the model is loaded. That is, for each point of a 3D grid around the model, we use an octree-based algorithm to determine and store the distance to the model's nearest point. We discretize the distance field on a regular grid. We could use more advanced techniques—such as adaptive structures—to both represent and compute the distance field. However, this computation is a preprocessing step and performance is not crucial; moreover, accessing a regular grid during runtime is fast enough.

The system uses the distance field each time it needs to find the z-coordinate to assign to a point $p(x_0, y_0)$ to make it lie at a given distance from the model. It easily accomplishes this by stepping along the ray $R(z) = (x_0, y_0, z)$ and stopping when it reaches the adequate distance value (we interpolate trilinearly to estimate distances for nongrid points). When the system performs this computation during a sweeping procedure, it starts the stepping at a neighboring pixel's existing z-value, which ensures efficiency and the result's spatial coherence. Otherwise, it starts the process near the near plane of the rectangular frustum on which the distance field has been computed.

Results quality and computation time depends directly on the resolution of the 3D grid storing the distance field. The size of the 3D grid is user configurable, but we generally use a $128 \times 128 \times 128$ grid to cover the whole body.

Processing contour lines

To generate the garment's 3D surface, the system must analyze the user's 2D strokes and assign them a 3D position.

2D processing

First, the system must classify the parts of the user's sketch. When the user starts or ends a new line within a few pixels of an existing line's endpoint, the system assumes that the lines connect. While the user is drawing, the system breaks finished lines into segments by detecting points of high 2D curvature (breakpoints).

Once the sketch is complete—that is, it forms a simple, closed curve in the plane—the system further classifies all segments. It classifies a segment as a borderline if the segment's projection meets the body's projection in the xy-plane; otherwise, it classifies it as a silhouette. To make such classification efficient, we precompute the body's projection mask and store it in a buffer (the body mask). Users can see the resulting segmentation, and can add or delete breakpoints indicating segment boundaries as necessary. Following this, the system reclassifies the segments.

Distance and z-value at breakpoints

We use the body mask to find breakpoints that are located over the body model; these points indicate garment regions that fit tightly to the body. We assign such points a zero distance from the model, setting their z-value to the body's z at this specific (x, y) location.

To assign a distance value d to a breakpoint that doesn't lie over the body, the system:

- steps along the ray from the eye in the direction of the breakpoint's projection into the xy-plane,
- checks distance values in the distance field data structure as it goes, and
- finds the z-value that minimizes this distance.

By assigning the breakpoint the discovered z- and d-values, we position the breakpoint in 3D.

Line positioning in 3D

We use the breakpoints' computed 3D positions to roughly position the garment in 3D, inferring the garment's shape primarily from distances to the body along the sketch silhouettes. To position the silhouette lines in 3D, we interpolate z linearly along the edge between the two breakpoints at the silhouette's extremities. We then set the d-values for interior silhouette points to those stored in the precomputed distance field. Instead, we could interpolate d directly, and compute associated z-values. However, if the body curves away from the silhouette curve, the interpolated d-value might have no z-value. Alternatively, we could compute d directly for each interior point, then assign the closest body point's z-value (as with breakpoints). In practice, however, this leads to wiggly lines because of the coarse grid on which we precompute the approximate distance-to-body field.

So, having established the z- and d-values along silhouette edges, we must extend this assignment to the borderlines. We do this in the simplest possible way: we assign d linearly along each borderline. Thus, for example, in Figures 2, 3, and 4, the d values at each end of the waistline are small, so the entire waistline's d-value will be small. Likewise, the d-values for the hemline's ends are quite large, so the values along the rest of the hemline will be large, too.

3D reconstruction of the garment's surface

To infer the garment's surface position, we use information gathered on the strokes.

Using distance to guess surface position

As with the contour lines, our main clue for inferring the garment interior's 3D position is the interpolation of distances to the body. Propagating distance values inside the garment consists of several steps.

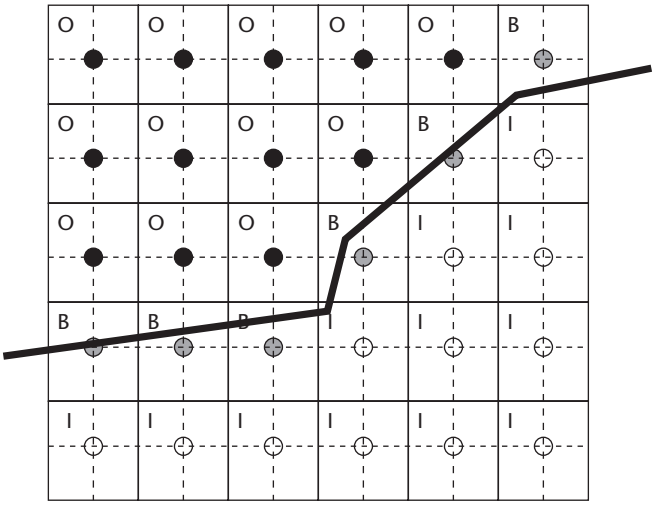
First, we use the closed 2D contour lines to generate an (x,y) buffer (sized to the sketch's bounding box). As Figure 8 (on the next page) shows, we assign each pixel in the buffer a value—in, out, or border—based on its position with respect to the contour. In a border pixel, a contour line intersects a small vertical or horizontal line in the pixel's center. Other pixels are either inside or outside the contour, depending on the contour's winding number at the pixel center. We assign the border pixels the distance values already computed along the silhouette and borderlines.

We also want to minimize the distance variation inside the garment, so that the resulting garment is as

tight as possible, given the border constraints. Let Ω be the set of inside and boundary pixels and $\delta\Omega$ the boundary. We already know $f_d^*|_{\delta\Omega}$ the predetermined distance values on the boundary, and want to find an interpolant f_d without extrema over Ω. This interpolant satisfies

$$\Delta f_d = 0 \text{ over } \Omega, \text{ with } f_d\Big|_{\delta\Omega} = f_d^*\Big|_{\delta\Omega} \quad (1)$$

Equation 1 is a Laplace equation with Dirichlet boundary conditions, which we can solve by simply iterating convolutions with a 3×3 neighbors averaging mask over Ω. We then sweep in the 2D grid for computing z-values at the inner pixels, corresponding to the distances obtained with Equation 1.



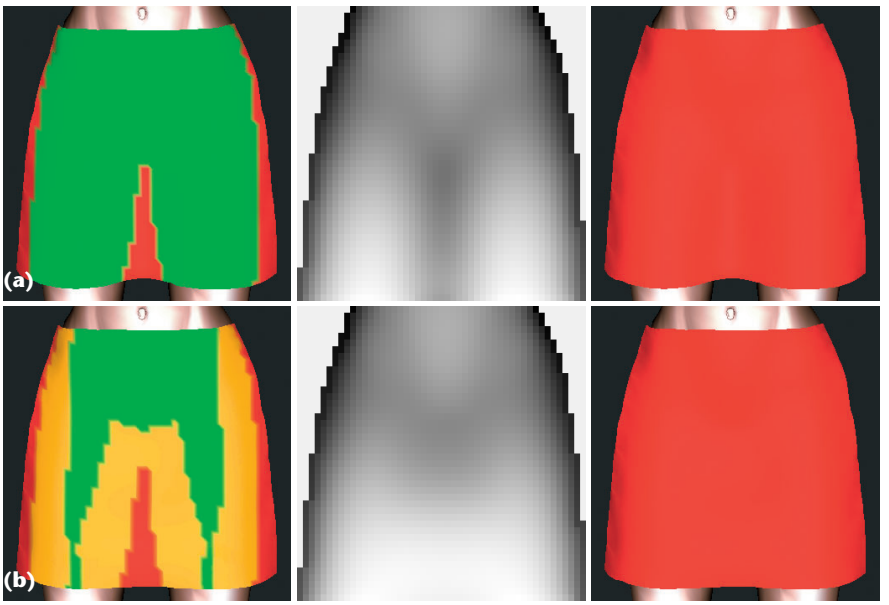
8 Using distance to guess surface position. Each grid point represents a small square (solid lines). If a contour (heavy line) meets a small vertical or horizontal line through the pixel center (dashed lines), the pixel is classified as a boundary (B); otherwise, pixels are inside (I) or outside (O) the contour, depending on the winding number.

Mimicking cloth tension

The previous computation gives us a first guess of the garment's 3D position, but still results in artifacts between two limbs of the character. Due to tension in the cloth itself, a garment should not fit tightly in the region between two limbs (as in Figure 9a), but rather smoothly interpolate the limbs' largest z values. To achieve this, we first erode the 2D body mask by a proportion that increases with the underlying d-value (see Figure 9b, left). We then use a series of Bezier curves in horizontal planes to interpolate the z-values for the in-between pixels. We chose horizontal gaps because of the human body's structure: for an upright human (or most other mammals), gaps between portions of the body are more likely to be bounded by body on the left and right than to be bounded above and below.

To maintain garment surface smoothness near the recomputed region, we extract distance values from the new z-values and the distance field. We perform some distance propagation iterations again in 2D before recomputing the z-values in the regions not lying over the body; these regions were not previously filled with the Bezier curves (as in the right side of Figures 9a and 9b).

Finally, we apply a smoothing step to the z-values to get a smoother shape for cloth that floats far from the character's model. To do this, we compute a smoothed version of the z-buffer by applying a standard smoothing filter. We then take a weighted average, at each pixel, of the old and smoothed z-values, with weighting coefficients depending on each pixel's distance from the model.



9 Mimicking cloth tension. (a) Surface reconstruction without accounting for tension. (b) Surface reconstruction that takes tension into account. In the left images, the part of the surface over the body mask is shown in green. At bottom left, the body mask is eroded and the system uses Bézier curves to infer the z-values between the legs. The middle images show the resulting z-buffers; the images on the right show the reconstructed surfaces.

Adding surface folds

We express folds as a garment's surface deformation, where the deformation's magnitude in z is at a maximum on the fold stroke, and decreases away from the stroke. The deformation's magnitude corresponds to a 2D Gaussian centered on the stroke point closest to the surface sample point. The algorithm proceeds as follows.

For each segment in the fold stroke, determine which pixels in the garment map intersect the segment. For each pixel intersected:

- Clip the segment to the pixel.
- Determine the segment's center point (the origin for sampling the pixel's 2D Gaussian).
- Sample the Gaussian within the pixel's neighborhood (see Figure 10).

We want the effect of the Gaussian to fall to zero at the radius users specify. Because 99.7 percent of the Gaussian's support is within three standard deviations (σ), we

set the Gaussian's σ to be one-third of the radius. We linearly interpolate the Gaussian's radius and amplitude in each sampled pixel between the fold stroke's extremities. This interpolated radius specifies the size of the pixel's neighborhood to sample.

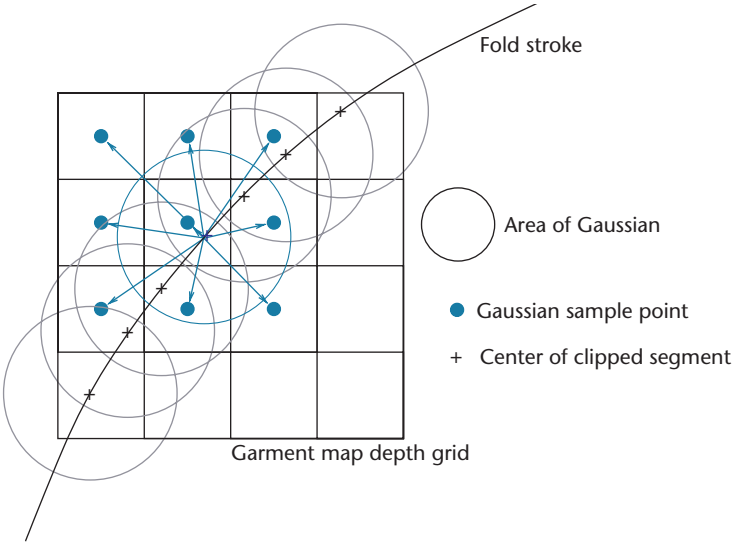
Once we've computed all the folds' contributions, we apply the resulting offset to the previously computed z-values.

Mesh generation

In the last step, we use the standard, diagonally subdivided mesh's triangles as the basis for the mesh we render. We retain all inside vertices, remove all outside vertices and triangles containing them, and move boundary vertices to new locations using a simple rule:

- If any segments end within the unit box around the vertex, we move the vertex to the average of those segments' endpoints. (Because segments tend to be long, it's rare to have more than one endpoint in a box.)
- Otherwise, some segments must intersect the box's vertical and/or horizontal midlines; we move the vertex to the average of all such intersections.

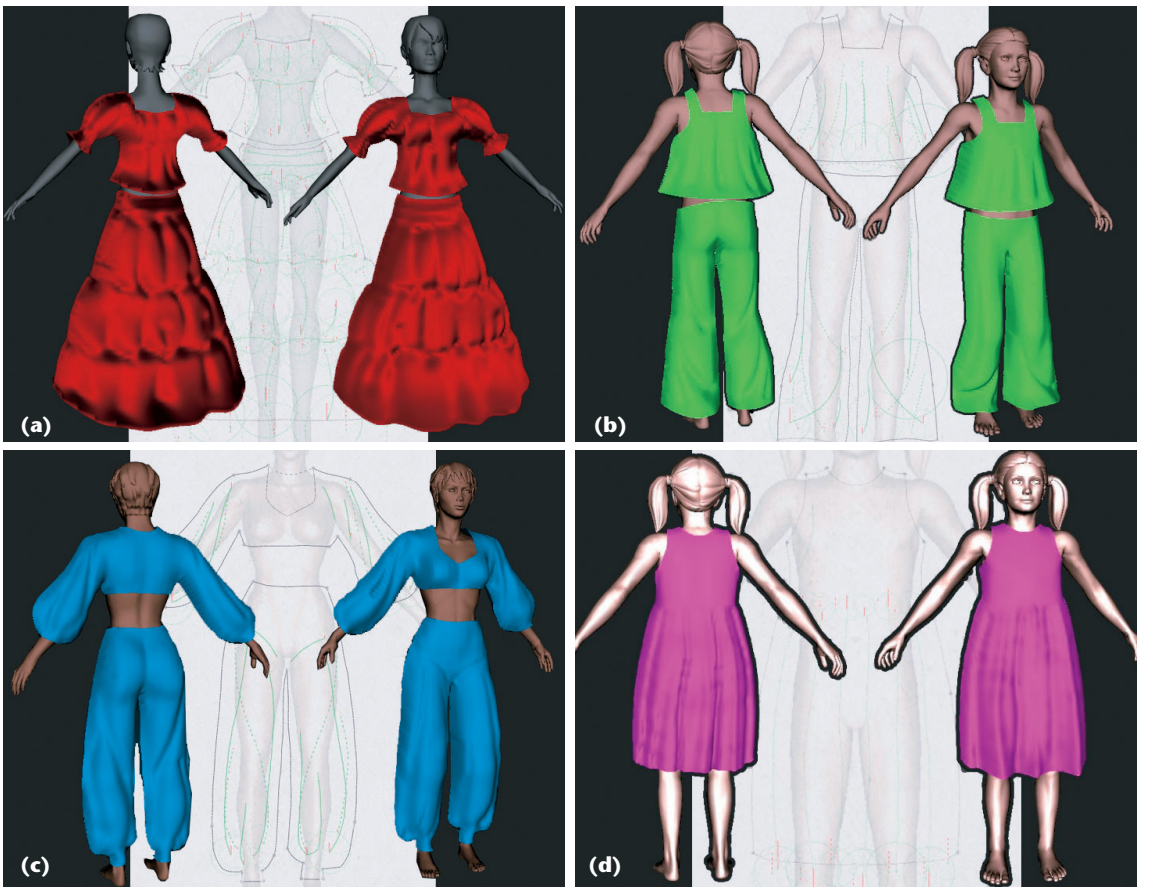
Essentially, we provide this simple triangulation to let users instantly visualize the garments. To produce meshes suitable for simulation system use, for example, we could replace this approach with a more elaborate meshing scheme to generate more uniform meshes.



10 Computing the z-offset generated by a fold. For each pixel intersecting the fold line, a Gaussian is applied to the neighboring pixels.

Results and discussion

Our designer drew the examples in Figures 1 and 11 in less than 5 minutes each. The strokes' jagged appearance in the drawings resulted from our using a mouse as the input device, rather than a more adequate graphics tablet. Our gallery includes simple clothes such as pants, skirts, robes, and shirts, as well as less-standard



11 Example garments created using the sketch-based approach.

Related Work

Current interactive systems[1-3] for designing garments and dressing virtual actors can be quite tedious. Typically, users must:

- draw each pattern piece on a planar virtual cloth,
- specify the edges to be stitched together,
- position the pieces around the virtual actor, and
- run a simulation to obtain a convincing garment rest shape on the character model.

Not only is this a long process, it is fairly unintuitive for users with no prior experience in fashion design.

Sketch-based modeling systems have become popular for interactively generating 3D geometry from sketches. This popularity exists not only within the research community and among graphics enthusiasts, but also among the general public and within large businesses—as exemplified by Google's recent acquisition of SketchUp (see http:// sketchup.google.com), a user-friendly CAD system. One trait these systems share is that, to infer the third (missing) dimension, they make assumptions about the objects the user is about to create. Such hypotheses are often expressed by low-level geometrical considerations: the widely cited Teddy program[4] helps users create smooth volumes, whereas Cherlin and colleagues created a system that can generate two kinds of parametric surfaces.[5] Such systems can also be based on higher-level a priori knowledge—as in Malik's system,[6] which narrows the range of expressible entities to hairstyles. Our system is part of the latter category; we limit ourselves to surfaces-with-boundaries to represent garments.

Two projects have combined the sketching idea with the goal of designing clothes: Bourguignon provided a 3D sketching method to design garments over virtual actors.[7] Users could view the sketch from arbitrary viewing angles, but the system didn't reconstruct a 3D garment surface. Igarashi and Hughes[8] described a sketch-based method for positioning garment patterns over a 3D body, but users couldn't directly sketch the garment in the system and they had to know which pattern shapes would result in the garments they desired. That is to say, the program used sketching to address the dressing and (to some extent) simulation problems, but not the tailoring problem.

References

1. B.K. Hinds and J. McCartney, "Interactive Garment Design," *The Visual Computer*, vol. 6, no. 22, 1990, pp. 53-61.
2. H.M. Werner et al., "User Interface for Fashion Design," *Proc. Int'l Conf. Computer Graphics*, North-Holland, 1993, pp. 197-204.
3. T. Bonte, A. Galimberti, and C. Rizzi, *A 3D Graphic Environment for Garments Design*, Kluwer Academic Publishers, 2002, pp. 137-150.
4. T. Igarashi, S. Matsuoka, and H. Tanaka, "Teddy: A Sketching Interface for 3D Freeform Design," *Proc. 26th Conf. Computer Graphics and Interactive Techniques*, ACM Press, 1999, pp. 409-416.
5. J.J. Cherlin et al., "Sketch-Based Modeling with Few Strokes," *Proc. 21st Spring Conf. Computer Graphics* (SCCG), ACM Press, 2005, pp. 137-145.
6. S. Malik, "A Sketching Interface for Modeling and Editing Hairstyles," *Proc. Eurographics Workshop Sketch-Based Interfaces and Modeling*, T. Igarashi and J.A. Jorge, eds., Eurographics, 2005, pp. 185-194.
7. D. Bourguignon, M.P. Cani, and G. Drettakis, "Drawing for Illustration and Annotation in 3D," *Computer Graphics Forum*, vol. 20, no. 3, 2001, pp. 114-122.
8. T. Igarashi and J.F. Hughes, "Clothing Manipulation," *Proc. 15th ACM Symp. User Interface Software and Technology*, ACM Press, 2002, pp. 91-100.

garments such as a bohemian dress and eccentric outfits suitable for haute couture collections. This wide range of clothing types shows our system's expressiveness.

We are aware that other approaches could be used for cloth design. For instance, someone could create a parametric template for shirts and a program that lets users place the template over a particular character and then adjust the shirt's neckline, overall size, and so on. However, this approach would limit design choices to a predefined template library and limit users to standard models as well. Nonetheless, such a model-based approach would be quite reasonable for many applications, such as a virtual Barbie doll.

In addition to the approaches we described here, we could use other methods or augmentations as well. First, our automated shape inference is simple and easy to understand, but might not be ideal in all cases. We've also yet to provide a way for users to edit the solution to make it better match their sketched ideas. Second, our system currently generates only single-layer garments. This is a reasonable limitation for dressing video-game characters, for example, but is certainly too restrictive for prototyping complex digital garments for movie characters—and even more so for designing real clothing.

Finally, our system can be a first step in a pipeline whose final product is a more physically realistic garment. Decaudin and colleagues, for example, have generated 2D flat panels from an initial garment that our system produced.[4] This, coupled with a fast physical simulation, could let users generate realistic garments that closely match the input sketch.

Conclusion

We plan to offer users more control over the generated surface's geometric properties. Currently, we can only ensure a C_0 surface continuity, notably at the silhouettes. A higher-order continuity might be desirable in many situations. To provide greater continuity, we plan to use an approximate (smoothed) distance field instead of the Euclidean distance. This would also permit faster convergence of the iso-sets toward a sphere, which would result in smoother surfaces as users move away from the body. We might also replace the harmonic distance diffusion inside the garment with a more customizable one.

We plan several other improvements as well. First, the tessellation we use to generate the final mesh is simple; we'd like to improve it to create uniformly triangulated meshes and to account for fold directions. Second, we could substantially improve system expressiveness by letting users edit and modify garments from multiple views. The system could then render the current surface nonphotorealistically, displaying the silhouettes and borders, which users could then oversketch.

Finally, we've sketched clothing as though it were simply a stiff polygonal material unaffected by gravity. We'd like to let users draw clothing, then indicate something about the material's stiffness to see how it would drape over the body. For example, silk (almost no stiffness), canvas (stiff), and tulle (very stiff) generate different draping behaviors. We also plan to consider the inverse approach, where the system would infer a fabric's mechanical properties from the fold patterns that users draw. In the much longer term, we'd like to incorporate a simulator that can simulate the difference between bias-cut cloth and straight-grain, the former being far more clingy than the latter. ■

References

1. E. Turquin, M.P. Cani, and J. Hughes, "Sketching Garments for Virtual Characters," *Proc. Eurographics Workshop Sketch-Based Interfaces and Modeling*, T. Igarashi and J.A. Jorge, eds., Eurographics, 2004, pp. 175-182.
2. D. Bourguignon, M.P. Cani, and G. Drettakis, "Drawing for Illustration and Annotation in 3D," *Computer Graphics Forum*, vol. 20, no. 3, 2001, pp. 114-122.
3. J.J. Cherlin et al., "Sketch-Based Modeling with Few Strokes," *Proc. 21st Spring Conf. Computer Graphics* (SCCG), ACM Press, 2005, pp. 137-145.
4. P. Decaudin et al., "Virtual Garments: A Fully Geometric Approach for Clothing Design," *Computer Graphics Forum, Proc. European Assoc. Computer Graphics* (Eurographics), 2006, European Assoc. Computer Graphics, pp. 625-634.

Emmanuel Turquin *is a PhD candidate in computer graphics in the Jean Kuntzmann Laboratory's ARTIS research group (acquisition, representation, and transformations for image synthesis) in Grenoble, France. His research interests include global illumination and relighting (offline and interactive), expressive rendering, and human– machine interfaces—especially sketch-based systems. Turquin has an MSc in vision, imaging, and robotics from the National Polytechnic Institute of Grenoble. Contact him at emmanuel.turquin@imag.fr.*

Jamie Wither *is a PhD candidate in computer graphics in the Jean Kuntzmann Laboratory's EVASION group (for virtual environments for animation and image synthesis of natural objects) in Grenoble, France. His interests include sketch-based interfaces and rendering natural phenomena. Wither has an MSc in vision, imaging, and virtual environments from University College London. Contact him at jamie.wither@imag.fr.*

Laurence Boissieux *is an engineer at INRIA Rhône-Alpes in Grenoble, France, where she is in charge of the VR platform. She used to be a research assistant in computer graphics at MIRALab, Switzerland. Her research interests include 3D modeling and design. Boissieux has a master's degree in computer graphics from Claude Bernard-Lyon I University. Contact her at laurence.boissieux@imag.fr.*

Marie-Paule Cani *is a professor of computer science at the National Polytechnic Institute of Grenoble, France. She is the head and creator of INRIA's EVASION research group. Her research interests include physically based simulation, implicit surfaces applied to interactive modeling and animation, and the layered model design that incorporates alternative representations and levels of detail. Cani has a PhD in computer science from the University Paris XI and a Habilitation degree from INP Grenoble. Contact her at marie-paule.cani@imag.fr.*

John F. Hughes *is an associate professor of computer science at Brown University. His research interests include geometric modeling, expressive rendering, sketch-based interfaces, mathematical foundations of computer graphics, the interface between computer vision and computer graphics, and machine learning. Hughes has a PhD in mathematics from the University of California, Berkeley. Contact him at jfh@cs.brown.edu.*