# Relief: A Modeling by Drawing Tool

David Bourguignon, Raphaëlle Chaine, Marie-Paule Cani, George Drettakis

EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2004)
John F. Hughes and Joaquim A. Jorge (Guest Editors)

# Relief: A Modeling by Drawing Tool

David Bourguignon[1] Raphaëlle Chaine[2] Marie-Paule Cani[3] George Drettakis[4]

[1]Princeton University/INRIA Rocquencourt [2]LIRIS/Université Claude Bernard Lyon
[3]GRAVIR/INP Grenoble [4]REVES/INRIA Sophia-Antipolis

**Abstract**

*This paper presents a modeling system which takes advantage of two-dimensional drawing knowledge to design three-dimensional free-form shapes. A set of mouse or tablet strokes is interpreted by the system as defining both a two-dimensional shape boundary and a displacement map. This information is used for pushing or pulling vertices of existing surfaces, or for creating vertices of new surface patches. To relieve the burden of 3D manipulation from the user, patches are automatically positioned in space. The iterative design process alternates a modeling by drawing sequence and a viewpoint change. To stay as close as possible to the traditional drawing experience, the system imposes the minimum number of constraints on the topology of either the strokes set or the resulting surface.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling Geometric algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques Interaction Techniques J.5 [Computer Applications]: Arts and Humanities Fine arts

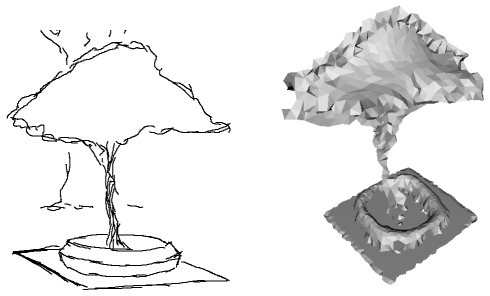
**Keywords:** drawing, modeling, interface, reconstruction, point-set surface

## 1. Introduction

Most people draw. We sketch, doodle, and scribble to keep track of our thoughts or communicate ideas to others. We consider drawing as an alternative to writing, because it is often faster and more concise to describe three-dimensional shapes and spatial relationships with two-dimensional lines than with words. Drawing tools are simple and their dexterous use constitutes a wealth of common knowledge most people have acquired since kindergarten. However, this know-how is seldom used in computer graphics for anything but two-dimensional vector or pixel-based drawing applications.

Few people sculpt. Even though many people used to play at an early age with modeling toys such as Play-Doh and LEGO, creating forms in three dimensions usually involves modeling clay, chiseling wood or marble, etc. These materials are difficult to manage, and shaping them generally requires highly specialized tools and skills. Using computers does not make the sculpting process simpler: three-dimensional data are obtained either by scanning an existing sculpture, or by modeling directly with the computer, using 2D or 3D input devices. In practice, designers turn to computers when they need to be definite and precise. We believe that advantages of the digital medium could be used in the initial idea stage.

Present day modeling systems rely heavily on 3D widget-based interfaces to ease the user task of specifying 3D ac-



(a) (b)

**Figure 1:** *Model creation using Relief. In (a), a simple sketch of a tree, on paper. In (b), a three-dimensional model of a tree, obtained with Relief.*

tions using 2D input devices. This way, the user visualizes the task by looking at the 3D scene from several viewpoints, such as the traditional blueprint projection planes. The underlying rationale is that a 3D task is better performed using 3D tools [CSH*92]. This is true; however, these tools are generally controlled using 2D input devices, and this results in nonintuitive, advanced-user-oriented systems.

The inverse approach has received far less attention. Since most of the input devices we have are two-dimensional, we believe that 2D tools are more appropriate to perform 3D operations. This way, the user is able to visualize the task completely from only one viewpoint, i.e., the current camera image plane. The most simple 2D tools, that take advantage of all the expressive capabilities of a human hand, are drawing tools.

To stay as close as possible to the traditional drawing experience, our system input must consist of *just plain strokes*. They are interpreted by the system as defining a two-dimensional shape boundary and a displacement map. The former is computed using a curve reconstruction algorithm. The latter is obtained by following a simple shading convention. Together, they are used for pushing or pulling vertices of existing surfaces, or for creating vertices of new surface patches. (These new patches are attached or not to existing surfaces.)

In order to relieve the burden of three-dimensional manipulation from the user, new surface patches are positioned in space according to a few simple rules. Our surface representation handles arbitrary topology changes without the drawbacks of either polygonal meshes (difficult to edit and maintain through deformations) or voxel grids (storage overhead and signal aliasing). It is based on points and a surface reconstruction algorithm which is the three-dimensional equivalent of the algorithm mentionned above.

The iterative design process alternates a modeling by drawing sequence and a viewpoint change. Its output is a three-dimensional triangular mesh that fits easily in the standard computer graphics production pipeline.

Our system reuses some of the ideas proposed by previous modeling systems, such as specification of displacement through shading, and depth determination through projection on existing surface. However, it overcomes three of their key limitations. First, it formulates the *two-dimensional shape from strokes* problem as a curve reconstruction problem and thus handles most stroke input with ease. Second, it imposes no constraint on the appearance and topology of the surface during the design session, thus freeing the user from the burden of planning its entire sculpture process in advance. Third, it does not limit modeling operations to modifications of the initial surface, thus allowing a wider range of resulting shapes.

## 2. Related Work

Since the invention of Sketchpad by Ivan Sutherland in 1963 [Sut63], many computer graphics systems have used drawing metaphors for 3D shape modeling, most often requiring the user to draw polyhedral surfaces in wireframe [LS96, SC04]. However, this CAD atavism keeps these solutions out of reach of common drawing practice. Notably, Eggli et al. alleviate this burden of unintuitive input by using simple drawing techniques that have an unambiguous interpretation in three dimensions [EBE95].

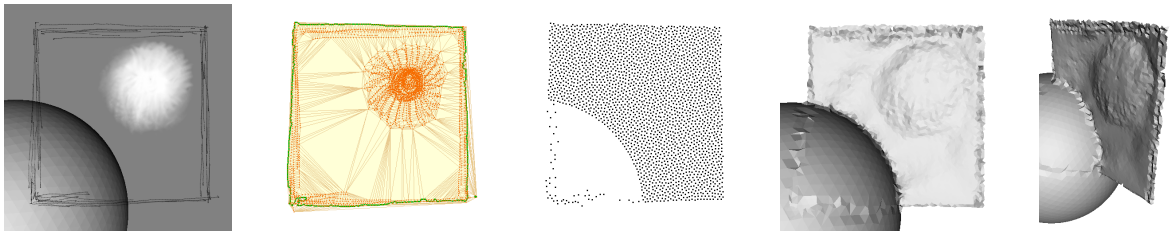
Departing from these approaches, SKETCH and Teddy demonstrated that gesture-based interfaces are powerful and intuitive tools for 3D model design [ZHH96, IMT99]. However, they trade their simplicity against limitations on the appearance (boxy, rotund) or topology of models generated. Concerning Teddy, the latter problem has been addressed using either variational implicit surfaces [KHR02] or a volumetric shape representation [ONNI03].

In systems mentioned previously, drawing is used for describing boundaries and shape features using strokes, but not as a way of modeling the relief of surfaces. In his pioneering work, Williams proposes to create a height field by directly painting the luminance value corresponding to a given elevation [Wil90]. On the contrary, the GRADED system uses shading information to infer a gradient map and a consistent depth map [van96]. Inspired by these attempts, a modeling tool based on a shape-from-shading algorithm has been proposed for surface retouching [RGB*03]. However, it is too slow for interactive use and requires photorealistic shading, which is difficult to obtain by drawing.

Three interesting solutions to the modeling by drawing problem are found in commercial systems. Artisan is a Maya software toolset with a common painting metaphor [Ali04], inspired from interactive texturing interfaces [HH90]. The Sculpt Polygons Tool pushes, pulls, or smoothes the surface when the user paints it with the corresponding brush. ZBrush greatly expands the mesh editing operations of Maya's Artisan [Pix04]. The user first sculpts a rough shape envelope using ZSpheres, and then refines it thanks to its subdivision surface multiresolution capabilities. Finally, SketchUp is a thorough architecture design system, inspired by SKETCH [@La04].

## 3. Tool Workflow

The Relief system is a modeling by drawing tool, i.e., modeling operations are achieved by drawing. The user draws only on the image plane, never directly on the objects of the scene. However, drawing operations take into account three-dimensional information, such as visibility and depth, by performing image queries in buffers, such as ID and depth buffers, using the graphics hardware.



(a) (b) (c) (d) (e)

**Figure 2:** *Tool workflow. In (a), the user has drawn both on an existing surface (an imported sphere) and on empty space, with a pencil (hard black strokes) and a brush (soft white strokes). In (b), reconstruction of the boundaries (green lines) of the drawn shape. In (c), adaptive sampling of the displacement regions. In (d) and (e), new surface obtained after reconstruction.*

A modeling by drawing sequence can be divided in two distinct steps: in the first step, the user draws, in image space; in the second step, the system uses the drawing information for moving or creating surface vertices, in world space. Then, the user can change the viewpoint and repeat a modeling by drawing sequence all over again.

### 3.1. First Step: Drawing

When the user draws a stroke on the image plane (see Fig. 2a), either with a mouse or a tablet, two types of information are recorded. First, the *stroke path* obtained from the input device is regularly sampled into two-dimensional points that are stored in a common point set, with other stroke samples (points are represented as orange dots on Fig. 2b). Second, the *stroke image* obtained from the stroke rendering process is accumulated in a pixel array, e.g., the frame buffer, with other stroke images (see Fig. 5a).

Once the drawing is finished, the system computes two image maps. First, from the point set, one or several closed curves defining the boundaries of the drawn shape are obtained using a two-dimensional reconstruction algorithm described in Sec. 4 (curves are represented as green lines on Fig. 2b). These curves are then transformed into a *binary map*, i.e., a mask defining image regions inside the shape. Second, from the pixel array, a *displacement map* is obtained using a simple shading convention, described below. Together, these two maps define *displacement regions* on the image plane.

The shading convention relates tones of the drawing to displacement values along an oriented axis. The user draws with black and white tones on a midgray canvas (see Fig. 2a). Shades (black tones) translate into negative displacement values, highlights (white tones) translate into positive displacement values, with respect to the oriented axis. The midtone (midgray canvas) has no influence. The magnitude of the displacement is proportional to the distance (in grayscale space) between the tone and the midtone.

### 3.2. Second Step: Modeling

The displacement information obtained in the first step is used to perform two types of modeling operation: either pushing or pulling vertices of existing surfaces, or creating vertices of new surface patches. (These new patches are connected or not to existing surfaces.) To achieve this, existing vertices are moved away from their previous position according to the displacement value, along the surface normal vector. New vertices are introduced when the displacement is not well sampled by the existing vertices. This is due either to the insufficient resolution of the existing surface, or the complete absence of it, when the user draws on a blank canvas.

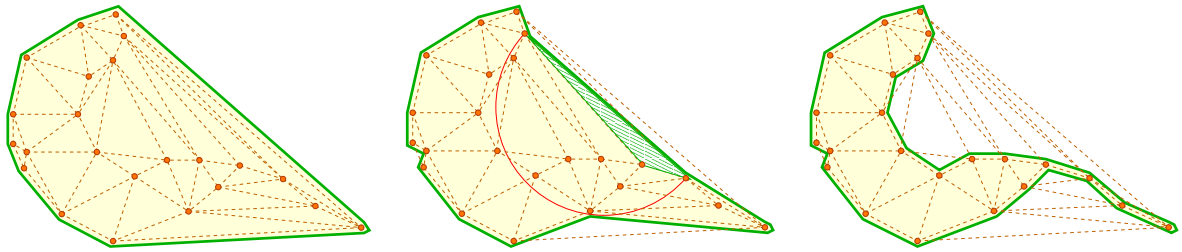
To determine if and where new vertices are introduced, displacement regions are adaptively sampled on the image plane (see Fig. 2c), using an algorithm described in Sec. 5. From their positions in image space, the new vertices positions in world space are computed using an algorithm described in Sec. 6. Finally, the new surfaces resulting from these operations are obtained using a three-dimensional reconstruction algorithm on the entire point set (see Fig. 2d-e). It is the three-dimensional equivalent of the algorithm mentioned in the first step.

### 3.3. Changing Viewpoint

Once the second step is over, the user is free to change the viewpoint to evaluate the result and repeat the entire modeling by drawing sequence (described in Sec. 3.1 and Sec. 3.2). Thus, the user progressively builds a three-dimensional model out of a series of modeling sequences performed each time from a different viewpoint.

## 4. Curve and Surface Reconstruction

In Sec. 3.1 and 3.2, we use the same reconstruction algorithm to obtain the shape of a point set: in the first step, we reconstruct curves from a two-dimensional point set resulting from the sampling of the stroke set; in the second step, we reconstruct surfaces from a three-dimensional point set resulting from the sampling of the displacement regions.



(a) (b) (c)

**Figure 3:** *The two-dimensional convection process. In (a), before the convection starts. The pseudo-curve lies on oriented edges of the Delaunay triangulation of the point set. In (b), the pseudo-curve evolves as long as the Gabriel criterion is not met for every oriented edge (half-disk, in red, associated with an oriented edge, in green). In (c), after the convection stopped. The result of the process is a topologically consistent set of oriented edges; it can contain coupled oriented edges called thin parts (the "tail" at the bottom right of the shape).*

In both cases, our reconstruction algorithm must satisfy several requirements:

- It must be able to handle an arbitrary number of connected components to allow modeling of an entire scene, containing many objects.
- It must be able to handle *outliers*, i.e., points located off shape boundaries. For example, in the two-dimensional case, strokes can describe either geometry or texture information, and thus do not always lie on the silhouette of the shape.
- It must be sufficiently fast to allow interactivity for average-sized point sets (around $5 \times 10^3$ points).

Our first attempt to solve this reconstruction problem was successful in 2D, but did not generalize well to 3D [Bou03]. However, a recent method meets all the requirements previously listed, for two- and three-dimensional point sets [Cha03]. We will describe this reconstruction algorithm and illustrate its key features in the two-dimensional case.

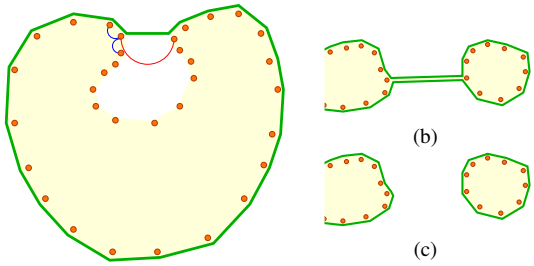
The reconstruction algorithm proposed by Chaine is based on an analogy with a physical phenomenon, convection. Let us consider a curve around a two-dimensional point set. Under the influence of convection forces, each point of the curve will move towards its nearest neighbor in the point set, with possible topological changes occurring during the evolution of the curve.

This process can be translated in computational geometry terms by evolving a data structure called *pseudo-curve* in the two-dimensional Delaunay triangulation of the point set. The pseudo-curve lies on oriented edges of the triangulation (see Fig. 3a), and it evolves as long as the half-disk associated with an oriented edge contains a point, i.e., an oriented edge does not meet the Gabriel criterion (see Fig. 3b). The result of the convection process is a topologically consistent set of oriented edges. For some point sets, this result contains *coupled* oriented edges, i.e., oriented edges with opposite orientations, that are geometrically dependent, but can be topologically independent. These coupled oriented edges are called *thin parts* (see Fig. 3c).

The resulting set of oriented edges can be partially or entirely made of thin parts, when the corresponding point set describes a non-manifold shape. (In the first step of our algorithm, described in Sec. 3.1, a set of oriented edges entirely made of thin parts is rejected, since it does not define a shape with one or several boundaries.)

Note that the convection process cannot correctly reconstruct a particular type of concavity called a *pocket*. To address this issue, pockets are detected during the evolution of the pseudo-curve, by comparing the size of each oriented edge with the local density, assuming that, unless the edge leads to a pocket, the edge size of a well-sampled object is consistent with the local density (see Fig. 4a).

In three dimensions, this convection approach is extended to a *pseudo-surface* evolving in the three-dimensional Delaunay triangulation of a point set. The result of the con-



(a) (b) (c)

**Figure 4:** *Convection pocket and collapse. In (a), an example of pocket. The size of the pocket edge (red half-disk) is larger than the size of two of its incident edges (blue half-disks). In (b), before a collapse of the pseudo-curve. In (c), after a collapse of the pseudo-curve. In 2D, a collapse is always caused by a pocket. In 3D, six possible collapse cases correspond to different topological changes of the pseudo-surface occuring during the convection process.*

vection process is a topologically consistent set of oriented facets.

We implemented the two algorithms using the 2D and 3D triangulation packages of the Computational Geometry Algorithms Library [CGA04].

## 5. Adaptive Sampling

After the first step (Sec. 3.1) of the modeling by drawing sequence has been completed, we face in the second step (Sec. 3.2) the following problem: given displacement regions in image space, and given an existing surface in world space, will the displacement values be correctly sampled by the existing surface vertices projected in image space? In the case of a highly refined surface whose projection entirely covers the displacement regions, this will probably be true. However, for all the other cases (poorly refined surface, projection not covering all the displacement regions), we have to address this issue.

Finding an optimal sampling of the displacement regions amounts to finding an error map that will provide information about the areas that need to be sampled more than others. We evaluate this error map by simply projecting the existing surface vertices onto the image plane, giving them a displacement value by looking for the value at the corresponding position in the displacement map (see Fig. 5a), and rendering the corresponding 2D Delaunay triangulation using the graphics hardware, i.e., by linear interpolation of the displacement values, considered as grayscale color attributes of the vertices (see Fig. 5b). The absolute value of the difference between this approximate displacement map and the true displacement map gives us an error map (see Fig. 5c).

This error map is then adaptively sampled (see Fig. 5d) using a fast and accurate error-diffusion algorithm [ZF03]. This sampling method is inspired from interactive geometry remeshing techniques [AMD02]. It allows a simple tuning of the sampling rate by a linear scaling of the values of the error map.

If the projection of the existing surface does not cover all the displacement regions, or if there is no existing surface at all, corresponding parts of the approximate displacement map are filled with an arbitrary continuous value (see Fig. 5c), in order to prevent these areas from not being sampled.

## 6. Depth Inference

Once the displacement regions have been adaptively sampled as described in Sec. 5, each new sample can be considered as a three-dimensional surface vertex projected on the image plane, with a known displacement value, but an unknown depth value. Our problem amounts to defining the depth of these new vertices in order to compute their three-dimensional position by "unprojection" from image space to world space.

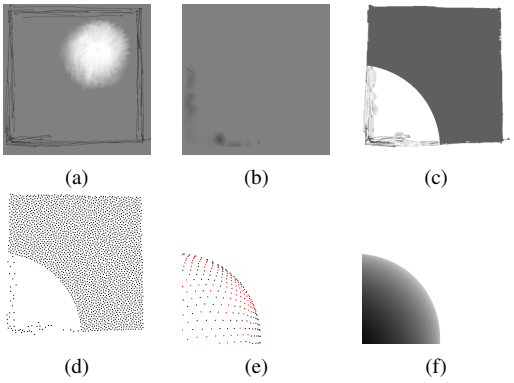



**Figure 5:** *Adaptive sampling and depth inference. (Same example as the one presented on Fig. 2.) In (a), the corresponding displacement map. In (b), the approximate displacement map, obtained by sampling the displacement map with the projections of the existing surface vertices. In (c), the error map corresponding to the difference between (a) and (b): the darker the tone, the larger the error. In (d), the adaptive sampling of the error map. In (e), the ID buffer used to identify the projections of existing surface vertices. In (f), the depth buffer corresponding to the existing surface.*

We infer the depth of the new vertices using the depth information of the vertices of the existing surface. To this end, we build a 2D Delaunay triangulation of all the vertices projected on the image plane: both the vertices of the existing surface, with known depth value (see Fig. 5e), and the new vertices, with unknown depth value. In practice, we reuse the triangulation data structure already built in Sec. 5, simply adding the new vertices to it.

Then, starting with the vertices of unknown depth value, whose incident vertices are of known depth value, possibly thanks to a depth buffer query (see Fig. 5f), we propagate depth information through the triangulation using a standard *best first search* algorithm, i.e., a breadth first search that uses a priority queue ordered according to, for a given vertex element, the number of incident vertices of known depth value. Successively examining the vertices in the queue, we evaluate their unknown depth using an interpolation scheme based on the distance between the vertex and each of its incident vertices of known depth value [Tau95]. As a result, the position of new vertices interpolates, and sometimes extrapolates, the position of existing surface vertices (see Fig. 8a-b).

In the case the user has not drawn at all on an existing surface, an arbitrary depth is assigned to the new vertices. This depth can be either chosen as the depth of the world space origin, or as the depth of a plane parallel to the image plane and defined by the user.

(a) (b) (c) (d)

**Figure 6:** *Hole marks. In (a) and (b), hole marks in comic book production: artwork from Stone #3, Avalon Studios. In (a), pencils by Whilce Portacio. In (b), inks by Gerry Alanguilan. Note the small crosses drawn by the penciler to give information about the positions of holes to the inker. In (c), and (d), hole marks in Relief. In (c), a hole mark defined by the user. In (d), the resulting effect on the reconstructed shape boundary. (In (a) and (c), marks are colored in red for clarity.)*

## 7. Tool Interface

We describe in this section a few algorithms that are used to ease the process of modeling by drawing, either to draw shapes with holes, to compute automatic shading effects, to obtain an adaptive shading-to-displacement mapping, or to perform various surface enhancement operations.

### 7.1. Hole Marks

When drawing a shape with holes, the definition of inside and outside parts is generally ambiguous. This problem is commonly solved in the comic book industry: the penciler (who has actually created the drawing, see Fig. 6a), gives hints to the inker (who will overdraw in ink some of the penciller lines, see Fig. 6b), about the parts of the drawing which are considered foreground and the ones which are considered background, that is, "holes" in the foreground.

Inspired by this traditional solution to the drawing topological ambiguity problem, we provide the user with the possibility of marking areas considered as holes with *hole marks*, i.e., special strokes obtained by maintaining a modifier key pressed while drawing (see Fig. 6c-d).

These marks are taken into account after the two-dimensional shape reconstruction is over (see Sec. 3.1). First, using the 2D Delaunay triangulation, we determine the faces intersected by each segment of a mark stroke and tag these *marked faces* as external to the shape. Second, inspecting each of these faces in turn, we gather the *boundary edges*, i.e., edges that belong both to a marked and a non-marked face. Third, these edges are used as starting edges for a rerun of the two-dimensional reconstruction algorithm, in the same way we used the convex hull edges in the initial reconstruction, as explained in Sec. 4.

### 7.2. Blobbing

In order to speed up modeling of regular rotund shapes, in the spirit of Teddy [IMT99], we provide a blobbing operation that performs an automatic inflation of the drawn shape, i.e., that creates the shading corresponding to an inflation of its silhouette.

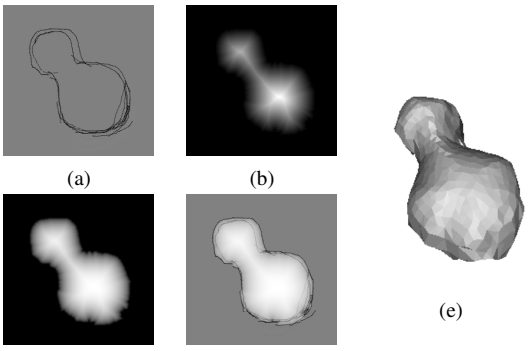
Solutions to the inflation problem can be classified as either object-based [vW97] or image-based [OCDD01]. Since the resulting blobbing is equivalent to a shading created by the user, we chose an image-based solution. Inflation is obtained in two steps. In the first step, we compute the Euclidean distance transform of the binary map corresponding to the drawn shape, using a fast algorithm [CM99]. This gives us the distance field $d(x,y)$ (see Fig. 7a-b). In the second step, in order to give a rotund appearance to the shape, we map the distance field to a unit sphere height field $z(x,y)$, i.e.,

$$z(x,y) = \sqrt{1 - \left(1 - \frac{d(x,y)}{d_{max}}\right)^2}$$

where $d_{max}$ is the maximum value of the scalar field $d$ (see Fig. 7c). Note that the value of $d_{max}$ gives us a distance (in pixel units) that can be used later to estimate the mapping of the shading corresponding to the normalized height field in image space, to a displacement field in world space (see Sec. 7.3). Finally, the resulting shading is accumulated on top of stroke images in the frame buffer (see Fig. 7d).

### 7.3. Shading-to-Displacement Mapping

According to our shading convention, displacement is proportional to the distance between a tone and the midtone (see



(a) (b) (c) (d) (e)

**Figure 7:** *Blobbing. In (a), the drawn shape. In (b), the corresponding distance field. In (c), the corresponding unit sphere height field. In (d) and (e), the resulting white shading and the corresponding surface.*

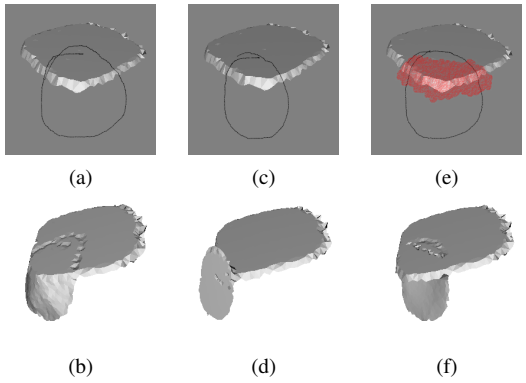



**Figure 8:** *Various depth modes. In (a) and (b), standard depth inference: the position of new vertices interpolates, and sometimes extrapolates, the position of existing surface vertices (see Sec. 6). In (c) and (d), modeling at depth: new vertices are positioned at the depth of the closest existing surface vertice involved (see Sec. 7.4). In (e) and (f), frisket mode: where frisket has been applied (red regions), the depth of existing surface vertices is not taken into account in standard depth inference computations (see Sec. 7.5).*

Sec. 3.1). However, in order to ease modeling, this mapping also depends on two other variables.

First, the mapping is proportional to the distance between the point of view and a point on the existing surface, i.e., the closer the viewpoint is to the surface, the smaller the displacements. Thus allowing the user to work with higher precision when zooming on surface details. Second, the mapping is inversely proportional to the current size of the surface: a small surface will undergo only small displacements, thus avoiding the user to adjust his work to the absolute size of the object.

### 7.4. Modeling *At Depth*

Inspired by Maya's Paint Effects depth modes [Ali04], we provide another modeling method (alternative to the one presented above in Sec. 3.2). This method can be useful when the user wants to avoid simply pushing or pulling vertices while drawing over an existing surface.

With this method, existing surface vertices projected in image space that fall into the displacement regions, are not influenced by displacement values. Moreover, the system computes their smallest depth value, and this value is assigned to the new vertices (see Fig. 8c-d). This method is sometimes convenient to accumulate several independent surface patches.

### 7.5. Frisket Masking

Inspired from the liquid masking gum of the same name, a frisket mode is provided for masking the projection of the existing surface on the image plane. This is very useful to perform fine surface edits or to prevent areas of the projected surface from being taken into account in subsequent depth inference computations (see Fig. 8e-f).

### 7.6. Other Surface Editing Modes

Surface editing operations, such as surface smoothing or vertex removal, are obtained using two simple algorithms. For surface smoothing, we use a low-pass filtering of the surface signal [Tau95]. This smoothing operation is currently available only if the user draws with a smoothing brush. However, it could be possible to apply it after each modeling sequence in order to automatically improve the smoothness of the resulting surface. For vertex removal, we simply identify vertices using an ID buffer and remove them from the three-dimensional point set. After each of those surface editing operations, the surface is reconstructed to take the modifications into account.

## 8. Results and Discussion

Results obtained with the Relief system are encouraging (see Fig. 9). As we mentioned in Sec. 1, our system improves over previous systems thanks to three key features. However, our preliminary implementation has still several limitations. We will discuss them here and review the work remaining to be done in Sec. 9.

The computing time bottleneck of the system is the construction of the three-dimensional Delaunay triangulation mentioned in Sec. 4. In fact, the complexity of the shape reconstruction algorithms is comparable to the complexity of the Delaunay triangulation of the point set. For example, reconstructing a surface from 6752 points takes 1.1 s [Cha03].

Three users have used our system to create models in different styles (compare for example the tree in Fig. 1b and Fig. 9e), which is a good indication that the system preserves the unique character of each user drawings. We demonstrate the actual use of the system in an accompanying video (showing the creation of the tree model displayed in Fig. 1b).

The informal testing of our software assessed the true benefits of several interface choices. For example, as opposed to what we could have expected, most users were not disturbed by the fact that the shading convention does not explicitly provide the exact displacement values applied. However, we found out two intrinsic problems with our drawing metaphor.

First, there is no continuous visual feedback: modeling operations occur after the user has finished drawing, not at the same time the user draws. An hybrid solution could combine a "synchronous" mode, for editing existing surfaces, as in commercial modeling systems [Ali04], and our "asynchronous" mode, for creating new surfaces.

Second, it is difficult to obtain a continuous shading, and

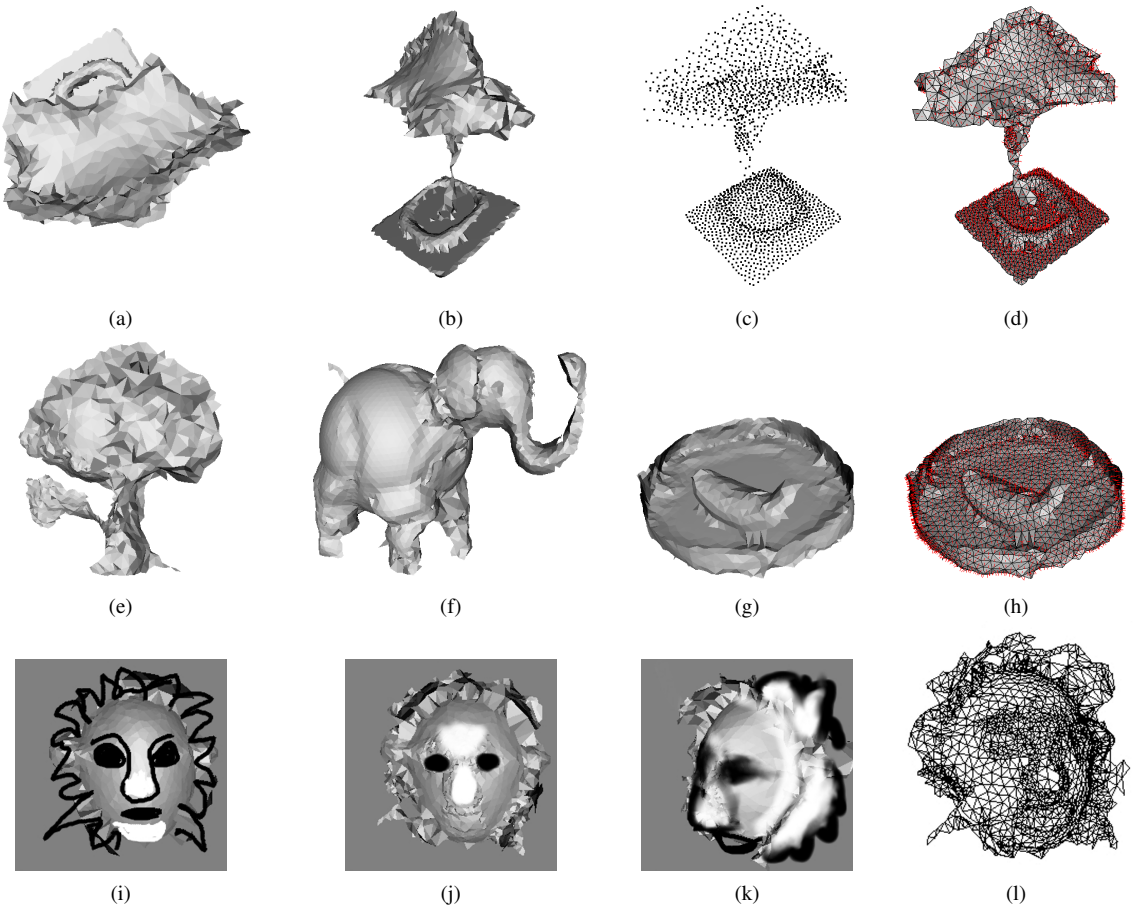



**Figure 9:** *Various models obtained with Relief. In (a) and (b), top and other side views of a tree (2139 points), previously displayed in Fig. 1b. In (c), the corresponding point set. In (d), the same tree rendered as a polyhedral surface with facet normals (in red). In (e), another tree (1026 points), modeled in a different style than the one displayed in (a-d). In (f), an imported sphere model transformed into an elephant (3718 points). In (g) and (h), a fruit basket with a lonely banana (1310 points), rendered either as a solid surface or as a polyhedral surface with facet normals (in red). In (i), (j), and (k), three sequences of the modeling of a lion head. In (l), the final model, rendered in wireframe.*

thus a smooth surface, with discontinuous strokes. Providing other higher level drawing operations, in the spirit of blobbing (see Sec. 7.2), could help the user easily achieve continuous shadings.

Finally, we would like to briefly compare our point-set surface representation based on evolving pseudo-manifold surfaces [Cha03] with recently proposed point-set surface representations based on Moving Least Square (MLS) surfaces [ABCO*01, AK04]. In fact, both representations handle arbitrary topology changes without the drawbacks of either polygonal meshes or voxel grids, but differences between them come from the way surface points are found.

Our representation amounts to finding topologically consistent surface neighborhoods, that together define an explicit surface. On the contrary, representations based on MLS surfaces amount to fitting a predefined surface around each point, and those local surfaces define together a global implicit surface. In the first case, there is no local surface model, and the resulting surface interpolates the point set. In the second case, there is a local surface model, and the resulting surface approximates the point set.

## 9. Conclusion and Future Work

We have presented Relief, a modeling by drawing tool which strives to stay as close as possible to the traditional drawing experience, while enabling the user to create three-dimensional models. Using a shading convention relating tone to displacement, we are able to edit existing surfaces or to create new surface patches, while imposing the minimum number of constraints either on the user drawing or on the resulting model.

Given our current implementation, this tool is more ap-

propriate for "quick and dirty" modeling than it is for precise editing. In that respect, it could be the modeling equivalent of a sketchbook, which is more appropriate for raw doodles than for polished drawings. Beyond the extensive work remaining in improving the interface and the internal algorithms of the system, three areas are of high interest for future work.

First, we are aware that the point-set surface representation could be improved. In fact, we perform a full surface reconstruction each time the point set is modified, while we could imagine a local surface reconstruction that considers only surface areas that have been invalidated by the last modifications. Since the reconstruction algorithm is based on the 3D Delaunay triangulation of the point set, the well-known locality property of this data structure is a strong incentive to investigate in this direction.

Second, we have not explored an alternative way to handle thin parts (see Fig. 10a). In fact, according to the definition given in Sec. 4, three-dimensional thin parts are coupled oriented facets that both meet the Gabriel criterion. Currently, when vertices are pushed or pulled, the corresponding thin parts behave as single pieces of surface (see Fig. 10b). However, an interesting modeling alternative would allow the separation of thin parts into two pieces of surface, by doubling the corresponding set of vertices (see Fig. 10c). In some sense, this could be considered as "blowing air" in between the coupled oriented facets.

This alternative approach would influence the depth inference algorithm (see Sec. 6). In fact, since the resulting surface would always be made of single-faced polygons, simple backface culling using the graphics hardware would remove backfacing polygons from the depth buffer, thus allowing more intuitive depth inference. As a result, this would maybe reduce the need for the frisket mode (see Sec. 7.5).

Third, the adaptive sampling algorithm described in Sec. 5 creates surface discontinuities at the boundaries of the drawn shape. In order to prevent them, we could define displacement regions differently by replacing the current binary map, used for masking the displacement map, with a grayscale map with progressive boundary attenuation. We could also take into account in the error metric higher-order properties of the displacement map, such as the curvature of the equivalent height field.

In fact, addressing the adaptive sampling issue could involve two distinct solutions, depending whether sampling is used when editing existing surfaces, or when creating new surface patches, connected or not to existing surfaces. In the latter case, our image-space solution would still be used, while, in the former case, an object-space solution [ZPKG02] could be found more appropriate, since it would allow adaptive resampling of the existing surface according to displacement values.

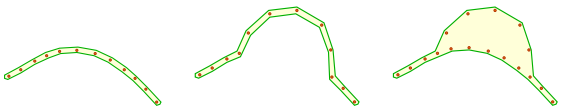



**Figure 10:** *Handling thin parts. In (a), the result of the convection process is a set of oriented edges entirely made of thin parts. Then, vertices in the middle are pulled upward, with two possible outcomes. In (b), the corresponding thin parts behave as single pieces of curve (current modeling approach). In (c), the corresponding thin parts are split into two pieces of curve, by doubling the corresponding set of vertices (alternative modeling approach).*

## Acknowledgments

This work has been performed while the first author was a visiting research fellow at Princeton University, supported by an INRIA post-doctoral fellowship. Many people have indirectly contributed to it. We would like to thank: Adam Finkelstein for inviting the first author to Princeton, Szymon Rusinkiewicz and Jason Lawrence for fruitful discussions; Pierre Alliez for his tips concerning halftoning, Mariette Yvinec for her help with CGAL; Laurence Boissieux, Laure Heïgéas and Laks Raghupathi, for installing and testing the system with the drawing tablet; Olivier Cuisenaire and Bingfeng Zhou, for putting their research code online.

## References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *Proceedings of the 2001 IEEE International Conference Visualization* (2001), IEEE Computer Society Press, pp. 21–28. 8

[AK04] AMENTA N., KIL Y.: Defining point-set surfaces. *ACM Transactions on Graphics (to appear)* (2004). 8

[Ali04] ALIAS: Maya 6.0, 3D animation and effects software, 2004. 2, 7

[AMD02] ALLIEZ P., MEYER M., DESBRUN M.: Interactive geometry remeshing. *ACM Transactions on Graphics 21*, 3 (2002), 347–354. 5

[Bou03] BOURGUIGNON D.: *Interactive Animation and Modeling by Drawing – Pedagogical Applications in Medicine*. PhD thesis, Institut National Polytechnique de Grenoble, 2003. 4

[CGA04] CGAL CONSORTIUM: CGAL 3.0, 2004. Computational geometry algorithms library. 5

[Cha03] CHAINE R.: A geometric convection approach of 3D reconstruction. In *Proceedings of the First Eurographics Symposium on Geometry Processing* (2003), pp. 218–229. 4, 7, 8

[CM99] CUISENAIRE O., MACQ B.: Fast and exact signed euclidean distance transformation with linear complexity. In *Proceedings of the 1999 IEEE International Conference on Acoustics, Speech and Signal*

*Processing* (1999), vol. 6, IEEE Computer Society Press, pp. 3293–3296. 6

[CSH*92] CONNER D. B., SNIBBE S. S., HERNDON K. P., ROBBINS D. C., ZELEZNIK R. C., VAN DAM A.: Three-dimensional widgets. In *Proceedings of the Second ACM Symposium on Interactive 3D Graphics* (1992), pp. 183–188. 2

[EBE95] EGGLI L., BRÜDERLIN B. D., ELBER G.: Sketching as a solid modeling tool. In *Proceedings of the Third ACM Symposium on Solid Modeling and Applications* (1995), ACM Press, New York, pp. 313–322. 2

[HH90] HANRAHAN P., HAEBERLI P.: Direct WYSIWYG painting and texturing on 3D shapes. In *Proceedings of ACM SIGGRAPH 90* (1990), Addison-Wesley, Boston, Massachusetts, pp. 215–223. 2

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: A sketching interface for 3D freeform design. In *Proceedings of ACM SIGGRAPH 99* (1999), Addison-Wesley, Boston, Massachusetts, pp. 409–416. 2, 6

[KHR02] KARPENKO O., HUGHES J. F., RASKAR R.: Free-form sketching with variational implicit surfaces. *Computer Graphics Forum 21*, 3 (2002), 585–594. 2

[@La04] @LAST SOFTWARE: SketchUp 3.0, Intuitive and accessible 3D design tool, 2004. 2

[LS96] LIPSON H., SHPITALNI M.: Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Computer-aided Design 28*, 8 (1996), 651–663. 2

[OCDD01] OH B. M., CHEN M., DORSEY J., DURAND F.: Image-based modeling and photo editing. In *Proceedings of ACM SIGGRAPH 2001* (2001), ACM Press, New York, pp. 433–442. 6

[ONNI03] OWADA S., NIELSEN F., NAKAZAWA K., IGARASHI T.: A sketching interface for modeling the internal structures of 3D shapes. In *Proceedings of Third International Symposium on Smart Graphics* (2003), Springer-Verlag, Berlin, pp. 49–57. 2

[Pix04] PIXOLOGIC: ZBrush 2.0, 2D and 3D painting, texturing and sculpting tool, 2004. 2

[RGB*03] RUSHMEIER H., GOMES J., BALMELLI L., BERNARDINI F., TAUBIN G.: Image-based object editing. In *Proceedings of the Fourth International Conference on 3D Digital Imaging and Modeling* (2003), IEEE Computer Society Press, pp. 20–28. 2

[SC04] SHESH A., CHEN B.: SMARTPAPER: An interactive and user friendly sketching system. *Computer Graphics Forum (to appear)* (2004). 2

[Sut63] SUTHERLAND I. E.: Sketchpad: A man-machine graphical communication system. In *Proceedings AFIPS Spring Joint Computer Conference* (1963), vol. 23, American Federation of Information Processing Societies Press, pp. 329–346. 2

[Tau95] TAUBIN G.: A signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH 95* (1995), Addison-Wesley, Boston, Massachusetts, pp. 351–358. 5, 7

[van96] VAN OVERVELD C. W. A. M.: Painting gradients: Free-form surface design using shading patterns. In *Proceedings of Graphics Interface 96* (1996), Canadian Human-Computer Communications Society, Toronto, pp. 151–158. 2

[vW97] VAN OVERVELD C. W. A. M., WYVILL B.: Polygon inflation for animated models: A method for the extrusion of arbitrary polygon meshes. *The Journal of Visualization and Computer Animation 8*, 1 (1997), 3–16. 6

[Wil90] WILLIAMS L.: 3D paint. In *Proceedings of the First ACM Symposium on Interactive 3D Graphics* (1990), ACM Press, New York, pp. 225–234. 2

[ZF03] ZHOU B., FANG X.: Improving mid-tone quality of variable-coefficient error diffusion using threshold modulation. *ACM Transactions on Graphics 22*, 3 (2003), 437–444. 5

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: SKETCH: An interface for sketching 3D scenes. In *Proceedings of ACM SIGGRAPH 96* (1996), Addison-Wesley, Boston, Massachusetts, pp. 163–170. 2

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3D: An interactive system for point-based surface editing. *ACM Transactions on Graphics 21*, 3 (2002), 322–329. 9