EUROGRAPHICS 2001 / A. Chalmers and T.-M. Rhyne
(Guest Editors)

Volume 20 (2001), Number 3

Drawing for Illustration and Annotation in 3D

David Bourguignon*, Marie-Paule Cani* and George Drettakis**

iMAGIS–GRAVIR/IMAG–INRIA †
* INRIA Rhône-Alpes, 655 avenue de l'Europe, F-38330 Montbonnot Saint Martin, France
** INRIA Sophia-Antipolis, 2004 route des Lucioles, F-06902 Sophia-Antipolis, France

Abstract

We present a system for sketching in 3D, which strives to preserve the degree of expression, imagination, and simplicity of use achieved by 2D drawing. Our system directly uses user-drawn strokes to infer the sketches representing the same scene from different viewpoints, rather than attempting to reconstruct a 3D model. This is achieved by interpreting strokes as indications of a local surface silhouette or contour. Strokes thus deform and disappear progressively as we move away from the original viewpoint. They may be occluded by objects indicated by other strokes, or, in contrast, be drawn above such objects. The user draws on a plane which can be positioned explicitly or relative to other objects or strokes in the sketch. Our system is interactive, since we use fast algorithms and graphics hardware for rendering. We present applications to education, design, architecture and fashion, where 3D sketches can be used alone or as an annotation of an existing 3D model.

Keywords: Drawing, Stroke-based illustration, Interface

1. Introduction

Drawing has long been an intuitive way to communicate complexity in a comprehensible and effective manner, due to visual abstraction. Compared to a photograph of a real object, extraneous details can be omitted, and thus attention can be focused on relevant features. The impression a drawing produces on a viewer is very different from the one a solid model produces: strokes mark the presence of a surface or a contour, but they can be "heavier" or "lighter", giving an indication of uncertainty where needed. The viewer's imagination is immediately engaged.

This kind of communication is useful in educational applications such as teaching, but also in the early stages of design, because drawing a sketch is much faster than creating a 3D model, and definitely more convenient to express ideas. However, the obvious drawback of 2D sketches is their limitation to a single viewpoint. The user cannot move around the object drawn, nor view it from different angles.

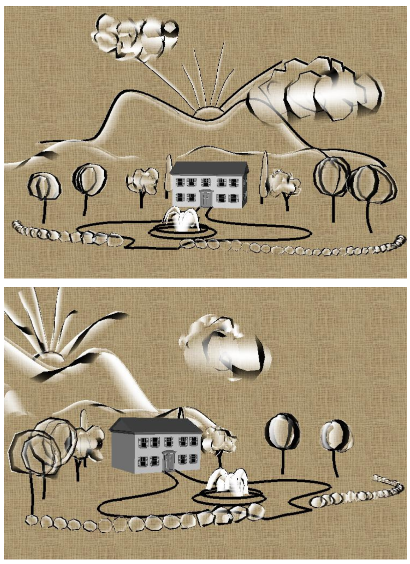


Figure 1: *A single landscaping sketch, which can also be seen as an annotation of an existing 3D model; the two different views are automatically generated by our system.*

Adding the ability to render a single sketch from multiple viewpoints has evident advantages, since the work of

† iMAGIS is a joint research project of CNRS/INRIA/UJF/INPG. E-mail:{David.Bourguignon|Marie-Paule.Cani}@imag.fr, George.Drettakis@sophia.inria.fr ; http://www-imagis.imag.fr/

the artist is significantly reduced. As an example, consider Fig. 1, which shows two views of a single rough landscaping sketch, generated by our system. It is imperative that such a system be interactive, since a slow, non-interactive system would interfere with the natural artistic creation process.

The aim of this paper is to provide a system that enhances classical sketching with 3D capabilities. Here, as in traditional 2D drawing systems, the user draws strokes that may represent either surface silhouettes or 1D features. These strokes may either belong to a single object or to many different ones, embedded in a complex scene. Both open and closed strokes may be used. As in 2D drawing, the user can draw several neighbouring strokes to accentuate a given contour. The main difference is that the viewpoint can be changed while drawing, thus creating a 3D sketch.

1.1. Related work

Our work is a natural continuation of drawing or sketching tools which have been developed in computer graphics over the last few years. In particular, researchers have realized the importance of providing usable tools for the initial phase of design, beyond traditional 3D modeling. These have taken the form of 3D sketch or drawing systems, typically with direct line drawing or stroke interfaces.

Some of these systems are directly based on 3D input[7, 4, 19]. They require special equipment (VR glasses, tracking sensors, arm-support, etc.) and often cannot be used on traditional workstations. We will not discuss them further; we will instead concentrate on solutions which do not require specific additional equipment and which are more practical than drawing directly in 3D.

Some systems use direct 2D drawing to reconstruct 3D scenes. Cohen et al.[6], present a system in which each 3D curve is entered by successively drawing its screen plane projection and its shadow on the floor plane. This approach can be useful for camera path description for example, but seems potentially unintuitive for drawing shapes. Tolba et al.[24], use projective 2D strokes in an architectural context. By projecting 2D points onto a sphere, 3D reprojection is achieved under different viewing conditions. Other systems restrict drawing to predefined planes, either as a generalization of traditional axis aligned views[15], or specifically onto cross-sections[10].

Another family of systems[12, 25, 8, 17], try to infer 3D models from 2D drawings and a set of constraints. Ivan Sutherland's "Sketchpad" system in 1963[23] can be considered as the inspiration for this work. During the 1990's a number of researchers advanced in this direction.

Among them, Pugh[17] presented the "Viking" system, where preferred directions and cutting planes guide the user in the placement of 3D vertices. Direct user intervention is possible both for explicit constraint specification and hidden line designation. Akeo et al.[1] used cross section lines to infer 3D shape, while Branco et al.[2] required the user to draw models without hidden lines, and used a classification scheme and direct user input to reconstruct in 3D. Eggli et al.[8, 9] interpret pen strokes by matching them to a small set of predefined primitives, and by applying a sophisticated constraint based system. The result is rapid construction of high-quality, CAD-like models. Lipson and Shpitalni[13] use optimization on a 2D edge-vertex graph which is tolerant to inaccurate positioning and missing entities; in addition, a number of different object types are supported.

In some of these systems, the advantage of viewing the designed model or scene in a non-photorealistic manner has been exploited to get a result that does not inhibit the designer imagination, as stressed by Strothotte et al.[22]. The SKETCH system[25] concentrates heavily on the use of efficient gestural input and non-photorealistic rendering to allow the creation of approximate, but complete, 3D models. More recently, the "Teddy" system[12] takes closed 2D freeform strokes as input, and reconstructs a polygonal object, which is converted back to strokes for rendering. Both systems emphasize a natural binding of gestures for the creation of specific shapes, affording a powerful and intuitive interface for rapid creation of specific types of 3D models.

The "Harold" system[5] is probably the most closely related previous work. The goal of this system is to provide a world populated by 3D "drawings". Three specific modes are provided: billboard (projection in a plane which turns towards the user), terrain and ground mode. The system allows the creation of "bridge strokes" between billboards. However, it does not handle strokes that correspond to the silhouette of a 3D object, and should thus deform when the point of view changes. As we shall see, this is an essential feature of our approach, since the capability to draw silhouettes is heavily required in both annotation and initial design applications.

1.2. Overview

The central idea of our approach is to represent strokes in 3D space, thus promoting the idea of a stroke to a full-fledged 3D entity. Even in 3D, we think that strokes are an excellent way to indicate the presence of a surface silhouette: several neighbouring strokes reinforce the presence of a surface in the viewer's mind, while attenuated strokes may indicate imprecise contours or even hidden parts.

Finding surface properties of objects from their silhouette is a classic hard problem in computer vision. The algorithms presented here do not address this issue, since our goal is to develop a drawing system rather than to perform geometric reconstruction. As a consequence, we develop approximate solutions that are appropriate in the context of drawing and sketching.

To enable the user to view stroke-based sketches from multiple viewpoints, we interpret 2D silhouette strokes as curves, and use a curvature estimation scheme to infer a local surface around the original stroke. This mechanism permits efficient stroke-based rendering of the silhouette from multiple viewpoints. In addition to stroke deformations, this includes variation of intensity according to the viewing angle, since the precision of the inferred local surface decreases when we move away from the initial viewpoint. It also includes relative stroke occlusion, and additive blending of neighbouring strokes in the image.

Apart from silhouette strokes, our system also provides line strokes that represent 1D elements. These have the ability to remain at a fixed position in 3D while still being occluded by surfaces inferred using silhouette strokes. They can be used to add 1D detail to the sketches, such as the arrow symbols in the example of annotation (see Fig. 11).

Because strokes have to be positioned in space, we present an interface for 3D stroke input. The user always draws on a 2D plane which is embedded in space. This plane is most often the screen plane, selected by changing the viewpoint. The depth location of this plane can be controlled either explicitly via the user interface or implicitly by drawing onto an existing object. The user may also draw strokes that are not in the screen plane, but that join two separate objects.

The combination of rapid local surface reconstruction and graphics hardware rendering with OpenGL results in truly interactive updates when using our system.

Finally, we show that our method can be applied to artistic illustration as well as annotation of existing 3D scenes, such as for rough landscaping or educational purposes. An existing 3D object can also be used as a guide to allow the design of more involved objects, e.g., using a model mannequin to create 3D sketches for clothing design.

2. Drawing and rendering 3D strokes

Two kinds of strokes are used in our system: line strokes that represent 1D detail, and silhouette strokes that represent the contour of a surface. This is the case for both open and closed strokes.

For line strokes, we use a Bézier space curve for compact representation. These strokes are rendered using hardware, and behave consistently with respect to occlusion.

Silhouette strokes in 3D are more involved: a silhouette smoothly deforms when the view-point changes. Contrary to line strokes, a silhouette stroke is not located at a fixed space position. It may rather be seen as a 3D curve that “slides” across the surface that generates it. Our system infers the simplest surface, i.e. the same local curvature in 3D as that observed in 2D. For this we rely on the differential geometry properties of the user-drawn stroke, generating a local surface around it. But the degree of validity of this surface decreases when the camera moves. Therefore, we decrease the intensity of the silhouette as the point of view gets farther from the initial viewpoint. This allows the user to either correct or reinforce the attenuated stroke by drawing the silhouette again from the current viewpoint.

2.1. Local Surface Estimation from 2D input

Since the inferred local surface will be based on the initial stroke curvature, the first step of our method is to compute the variations of this curvature along each 2D silhouette stroke drawn by the user.

Each 2D silhouette stroke segment is first fit to a piecewise cubic Bézier curve using Schneider’s algorithm[20]. This representation is more compact than a raw polyline for moderately complex curve shapes. Then, using the closest-point method[21], each control point of the piecewise Bézier curve is associated with a given value of the parameter u along the curve.

For each parameter value u associated with a control point $V(x(u), y(u))$, we find the center of curvature $C(\xi(u), \eta(u))$ by first computing the derivatives and then solving the following equations[3]:

$$\xi = x - \frac{\dot{y}(\dot{x}^2 + \dot{y}^2)}{\dot{x}\ddot{y} - \dot{y}\ddot{x}} \qquad \eta = y + \frac{\dot{x}(\dot{x}^2 + \dot{y}^2)}{\dot{x}\ddot{y} - \dot{y}\ddot{x}},$$

where $\dot{x}$ and $\ddot{x}$ are first and second derivatives of x in u. Therefore, we obtain a curvature vector between a point on curve at parameter u and its associated center of curvature C (see Fig. 2(a)). We will be using these curvature vectors to reconstruct local 3D surface properties. However, if the stroke is completely flat, the norm of the curvature vector (i.e. the radius of curvature) goes to infinity; the method we present next solves this problem.

In order to infer a plausible surface in all cases, we use a heuristic based on the curve’s length to limit the radius of curvature. One way of looking at this process is that we are attempting to fit circles along the stroke curve. Thus, if we encounter many inflection points, the circles fitted should be smaller, and the local surface should be narrower; in contrast, if the curve has few inflection points, the local surface generated should be broader.

To achieve this, we construct axis-aligned bounding boxes of the control polygon of the curve between each pair of inflection points (see Fig. 2(b)). Inflection points can be found easily since we are dealing with a well-defined piecewise cubic Bézier curve. We discard bounding boxes which are either too small or too close to the curve extremities. If the norm of the curvature vector is larger than a certain fraction of the largest dimension of the bounding box computed previously it is clamped to this value (see Fig. 2(b)). We use a fraction value at most equal to $\frac{1}{2}$, which gives a length equal to the radius of a perfect circle stroke.

We also impose a consistent in/out orientation of the curve

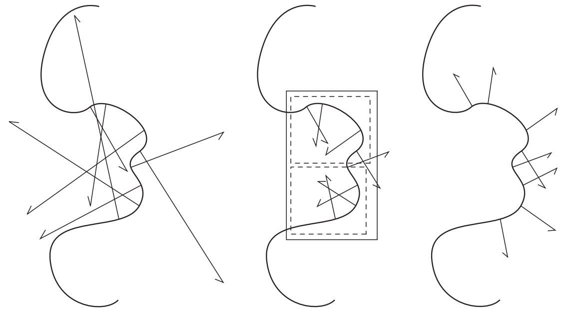


Figure 2: *Processing vectors of curvature. In (a), curvature vectors before clamping. In (b), curvature vectors after being clamped relative to solid bounding box length (dotted bounding boxes were too small to be selected). In (c), curvature vectors after correcting orientation.*

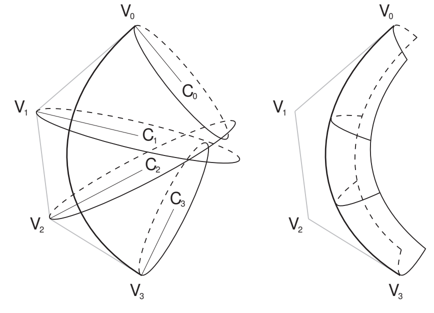


Figure 3: *Construction of a 3D stroke from a 2D stroke composed of one cubic Bézier curve with control points V_i. In (a), the 2D centers of curvature C_i computed with our method and the corresponding 3D circles. In (b), the Bézier surface obtained.*

based on the orientation of the curvature vectors in the first bounding box computed, thus implicitly considering initial user input as giving correct orientation (see Fig. 2(c)). This intuitive choice corresponds to the intent of the user most of the time. If not, a button in the UI can be used to invert all the curvature vectors along the stroke.

From these 2D strokes we infer local surface properties, which are then used to create a 3D stroke representation. Each center of curvature embedded in the drawing plane is considered as the center of a circle in a plane perpendicular to the drawing plane and passing by the corresponding control point (see Fig. 3(a)). We consider an arc of $\frac{2\pi}{3}$ radians for each circle, thus defining a piecewise Bézier surface by moving each control point on its circle arc (see Fig. 3(b)). This piecewise tensor product surface is quadratic in one dimension, corresponding to a good approximation to circle arcs, and cubic in the other, which corresponds to the stroke curve.

In practice, the inferred radius of curvature may of course be inaccurate, but as stated earlier, the inferred surface will only be used for generating a probable silhouette when the viewing angle changes slightly. If more information is needed about the 3D surface geometry, the contour will have to be redrawn by the user at another viewpoint.

For a complete overview of the behavior of our method in a simple "textbook example", see Fig. 4.

2.2. Rendering in 3D

Given a local surface estimation, our goal is to display the initial stroke from a new viewpoint. When the viewpoint changes, we expect the stroke to change its shape, as a true silhouette curve would do. We also expect its color to change, blending progressively into the background color to indicate the degree of confidence we have in this silhouette estimation. Recall that we want our system to be interactive, which imposes an additional computational constraint. In what follows, the term "local surface" corresponds to the polygonal approximation of the local surface estimation of the stroke.

The solution we adopt is to generate a fast but approximate silhouette based on the local surface generated as described above. We simply render a "slice" of the local surface that lies between two additional clipping planes, parallel to the camera plane and situated in front of and behind the barycenter of the centers of curvature (see Fig. 5(a)). The distance between clipping planes depends on the stroke width value we have chosen. This ensures silhouette-like shape modification, with minimal computational overhead.

It is important to note that our approach to silhouette rendering is very approximate. It is obvious that its behaviour will be somewhat unpredictable for wide camera angles and very long strokes. A good accurate solution for computing a new silhouette from the estimated surface would be to use one of the existing algorithms[14, 18, 11]. However, we have seen that our surface is only a coarse inference of the local surface to which the silhouette belongs, so computing an accurate silhouette would probably be unnecessary in our case.

Initially, we render all geometry other than the silhouette strokes (for example the house in Fig. 1). Therefore, the depth and color buffers are correctly filled with respect to

Figure 4: *"Textbook example": a simple circular stroke. In (a), front view; in (b), side view rotated by 30 degrees; in (c), side view rotated by 90 degrees; in (d), top view.*

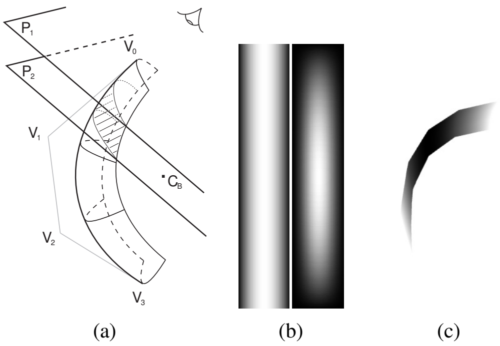


Figure 5: *Stroke rendering. In (a), the final stroke is a slice of Bézier surface obtained by using two clipping planes P_1 and P_2 facing the camera; C_B is the barycenter of the C_i. In (b), two texture samples, one of "stroke texture" (left) and one of "occluder texture" (right). White corresponds to an alpha value of 1, black to an alpha value of 0. In (c), image obtained from rendering a black stroke against a white background, with the slice position corresponding roughly to (a).*

this geometry. In the next step, we use different elements to display the silhouette strokes and to perform stroke occlusion. Because of this, we need a multipass algorithm.

First Pass: Rendering Silhouette Strokes

In the first pass, we render the strokes as clipped local surfaces, with the depth test and color blending enabled, but with depth buffer writing disabled. Thus correct occlusion is performed with respect to other (non-stroke) geometry.

To represent the confidence in the surface around the initial stroke we apply a "stroke texture" (see Fig. 5(b), left) as an *alpha* texture to the local surface. This confidence is maximum at the initial stroke position and minimum at left and right extremities of local surface. We use a Gaussian distribution that progressively blends the stroke color into the background color for modeling this confidence function. As a result, the stroke becomes less intense as we move away from the initial viewpoint. This blending also allows two different strokes to reinforce each other by superposition, which corresponds to the behaviour of traditional ink brush drawings.

Second and Third Pass: Occlusion by Local Surfaces

In addition to occlusion by other geometry, we also need to handle occlusion by strokes. This required a slightly more sophisticated process, since we do not want local surfaces to produce hard occlusion (such as that created by a depth buffer) but rather to softly occlude using the background color, in a visually pleasing way. To meet these requirements, stroke occlusion is achieved in an additional two passes. Recall that we start with a depth buffer which already contains depth information for other objects in the scene.

In the second pass, we render the local surfaces into the depth buffer with the depth test and depth buffer writing enabled. Local surfaces are textured with a different alpha texture called the "occluder texture" (see Fig. 5(b), right) and rendered with the alpha test enabled. As a result, occluder shape will be a rounded version of local surface shape.

In the third pass, we render the local surfaces into the color buffer with the depth test and color blending enabled, but with depth buffer writing disabled. Local surfaces are textured with the same "occluder texture" and colored with the background color. The occluder color thus blends with what is already present in color buffer: we obtain progressive occlusion from the edge of local surface to the center of initial stroke. During this pass, we use the stencil buffer to mask the stroke rendered during first pass, and thus the occluder does not overwrite it in the color buffer.

The multipass rendering algorithm is summarized in Fig. 6.

Drawing style

We can have a different color for the background and the stroke occluder, such as that shown in the artistic illustrations of Fig. 9 and 10. This gives a subtle indication of local surface around a stroke: it can be seen as equivalent to hatching or pencil pressure variation in traditional drawing.

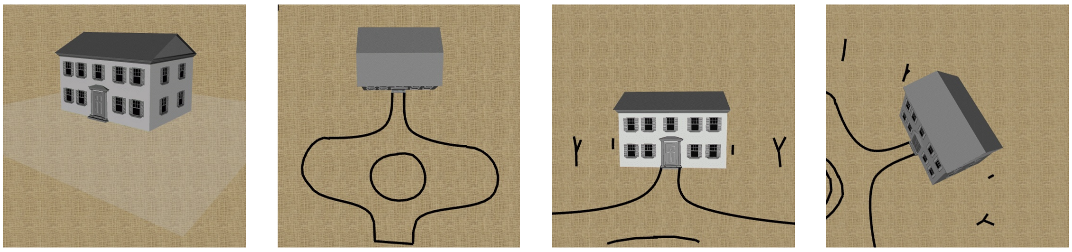
Finally, since "stroke texture" and "occluder texture" are procedurally generated, their parameters can vary freely. This allows the creation of different tools according to specific needs.

3. Interface for Drawing: Stroke Positioning

A drawing session using our system is in many ways similar to 2D drawing. A designer starts with an empty space, or, in the case of annotation, she can add in a pre-existing 3D model, such as the house in Fig. 1.

For strokes drawn in empty space, we project onto a reference plane, parallel to camera plane and containing the world origin (it is possible to choose a fixed offset relative to this position). Typically, the user will place the drawing plane in space using the trackball. An example is shown in Fig. 7, where we start drawing the grounds of the house. We want to draw in the plane of the ground corresponding to the house, so we position ourselves in a "top view". We then verify that the position of the plane is as intended (a) and draw the strokes for the grounds in the plane (b). Similarly, trunks of trees are drawn in planes parallel to the walls of the house (c).

Once such parts of the drawing have been created, we can use the existing entities to position the curves in space. More precisely, if at the beginning or at the end of a 2D stroke the pointer is on an existing object, we use this object to determine a new projection plane. We obtain the depth of



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

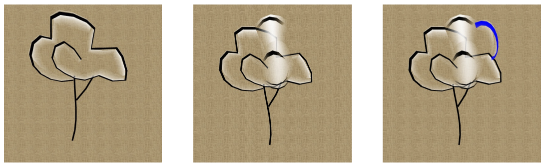
Figure 7: *Plane positioning. First, position ourselves in a "top view". Then, we verify the plane position colored in semi-transparent grey (a), and we draw the grounds in this plane (b). We next draw trees trunks in planes parallel to the walls of the house (c), and examine the result from another viewpoint (d).*

```
MultipassRendering()
{
    // Pass 1
    // Draw clipped local surfaces
    // with stroke texture, stroke color
    Enable Depth Test
    Enable Blend
    Disable Depth buffer write
    Draw strokes in color buffer
    // Pass 2
    // Draw local surfaces
    // with occluder texture
    Enable Alpha test
    Enable Depth test
    Enable Depth buffer write
    Draw occluders in depth buffer
    // Pass 3
    // Draw local surfaces
    // with occluder texture, occluder color
    Enable Stencil test
    Enable Depth test
    Enable Blend
    Disable Depth buffer write
    For each stroke:
        Draw clipped local surface in stencil buffer
        Draw occluder in color buffer
        Erase clipped local surface in stencil buffer
}
```

Figure 6: *Multipass Stroke Rendering Algorithm*

the point selected by a simple picking. The picked object can correspond to a geometric object or to (the local surface of) a stroke. There are three possibilities:

- If only the beginning of the stroke is on an object, we project the stroke on a plane parallel to camera plane, which contains the selected point. An example can be seen in Fig. 8, where we draw the leaves of a tree in one plane (a) and another (b).
- If the beginning and the end of the stroke are on an object, we interpolate depth values found at the two extremities by using the parameter u of the piecewise cubic Bézier curve. Each control point is projected on a plane parallel to the camera plane and located at the corresponding depth. See Fig. 8(c), where this "bridge" mechanism is used to join two parts of a tree.
- If it happens that the stroke extremities are in empty space, it is projected on the same plane as the previous stroke, except if the trackball has been moved. In this case, the reference plane is used.



(a) (b) (c)

Figure 8: *Different projections using objects of the scene. In (a) and (b), each time we draw on a plane automatically positioned in space with the help of the tree trunk, i.e., a plane passing through the trunk. This produces convincing tree foliage. In (c), we use a "bridge" to draw a new branch. It fits correctly in place because of the automatic positioning of the start and the end of the stroke.*

Classic 2D computer drawing capabilities extended to 3D are also very useful. Among them, we have implemented erasing strokes and moving strokes (in a plane parallel to camera plane).

4. Applications

We present the results of using our system for three different application scenarios. The first is for artistic illustration, the second involves annotation of a pre-existing 3D scene, and the third is for "guided design". In our current implementation, drawings can be saved in a custom file format, in world coordinates, but with no reference to an annotated object.

The initial learning curve for our system is relatively significant, requiring a few hours to get used to the idea of drawing in and positioning the planes. Once this is understood, typical drawings take between ten minutes to one hour to complete.

4.1. Illustration in 3D

Figure 9 and Figure 10 show two illustrations designed with our system. Most strokes are silhouette strokes. They have been rendered on a textured background so that the local surface occluder appears as a "fill" effect. Each illustration is shown from several different points of view, showing the effects of occlusion, varying stroke lightness, and silhouettes deformations.

4.2. Annotation of a 3D scene

Another application of 3D drawing is to use our system for annotating an existing 3D model. While 2D annotation is widespread, few systems provide a straightforward manner to do this in 3D. In a typical user session, we initially load the 3D model. It is subsequently integrated with the drawing in the same way as for local surfaces: if a stroke is drawn on the model, it is set to lie on it. Line strokes can be used for adding annotation symbols (e.g., arrows, text, etc).

Figure 1 has shown an example of annotation: adding a coarse landscaping sketch around an architectural model. Figure 11 shows annotation used for educational purposes: a heart model is annotated in a classroom-like context, for instance during an anatomy course. The use of a well-chosen clipping plane gives an inside view of the model and allows to draw anatomical details inside it.

Another example could be using our system in collaborative design sessions. Annotation could then be employed for coarsely indicating which parts of the model should be changed, and to exchange ideas in a brainstorming context.

4.3. "Guided design"

The idea of this third application is to load a 3D model and use it as a guide. When the drawing is completed, the model is removed. A good example of this kind of application is drawing clothes for fashion. A 3D model is used to obtain the body proportions (see Figure 12).

5. Conclusion and Future Work

We have presented a system which enhances the traditional 2D drawing process with 3D capabilities, notably by permitting multiple viewpoints for a single drawing. Instead of attempting a complete 3D reconstruction from 2D strokes, we infer a local surface around the stroke. This is achieved by assuming that strokes are drawn in a plane, and by using differential geometry properties of the curve.

The resulting local surfaces are then drawn efficiently using hardware accelerated rendering of a clipped part of the local surface, corresponding approximately to a silhouette. Color blending is used to gracefully diminish the intensity of the strokes as we move away from the initial viewpoint, and to allow reinforcement of intensity due to multiple strokes. We have also introduced a multipass method for stroke intervisibility, which results in a visually pleasing gradual occlusion.

Our approach is built into a system which provides an interface which retains many of the characteristics of traditional 2D drawing. We aid the user in placing the drawing plane in empty space, and positionning it relative to other objects (strokes or geometry) in the sketch.

Future Work

We have chosen a Bézier surface representation for storing silhouette stroke information. This approach is convenient but it can result in a large number of hardware rendered polygons. Other representations could be used instead.

For instance, a volumetric data structure would have allowed us to combine information about the local surface: the more strokes are drawn at a given location in space, the more we are certain that it corresponds to a true surface point. To render strokes from a new viewpoint, we would have used a variant of the marching cubes algorithm to produce a surface estimation, associated with a mesh silhouette detection algorithm to generate new silhouette strokes. The obvious drawbacks of this approach are memory consumption and data loss due to volumetric sampling of user input. A particle-based approach, i.e., strokes composed of particles that try to satisfy a set of constraints such as "stay on silhouette", etc., would produce interesting stroke transformations. However, numerical stability would undoubtedly be an issue as well as computational overhead, which would impede interactivity.

Our user interface is clearly far from perfect. Plane positioning is not completely intuitive, and alternatives should be considered. For example, a computer vision approach in which the user defines two viewpoints for each stroke and implicitly reconstructs positions in space (within an error tolerance) could potentially prove feasible. But it is questionable whether such an approach would truly be more intuitive.

We are investigating various such alternatives with the goal of finding an appropriate combination to improve our current solution, both in terms of stroke rendering quality and user interface.

Acknowledgments

Many thanks to Eric Ferley for his feedback throughout the project, to Laurence Boissieux for creating the landscaping and fashion examples, to Frédo Durand for his advice on the paper, and to Marc Pont for the mannequin model.

Figure 9: *Example of artistic illustration. Three views of the same sketch area are shown.*

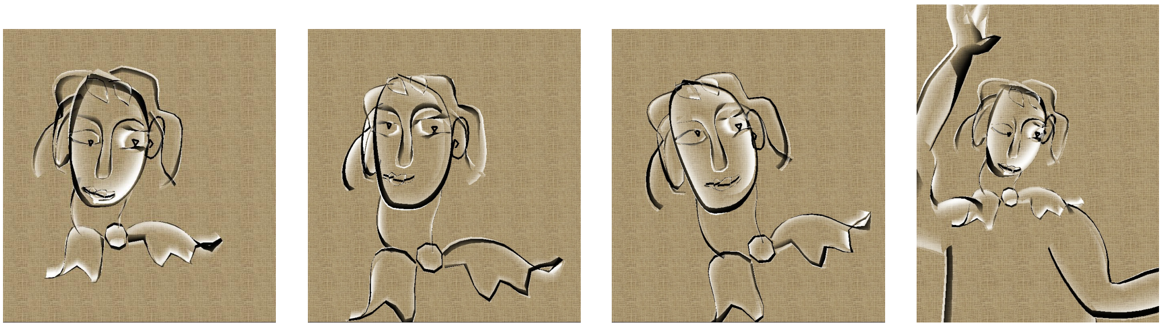


Figure 10: *A second example of artistic illustration. Three views of the same sketch area are shown. The fourth view is the finished drawing.*

References

1. M. Akeo, H. Hashimoto, T. Kobayashi, and T. Shibusawa. Computer graphics system for reproducing three-dimensional shape from idea sketch. *Computer Graphics Forum*, 13(3):C/477–C/488, 1994.
2. V. Branco, A. Costa, and F. N. Ferreira. Sketching 3D models with 2D interaction devices. *Computer Graphics Forum*, 13(3):C/489–C/502, 1994.
3. I. N. Bronshtein and K. A. Semendyayev. *Handbook of Mathematics*, page 554. Springer, 1998.
4. J. Butterworth, A. Davidson, S. Hench, and T. M. Olano. 3DM: A three dimensional modeler using a head-mounted display. *Computer Graphics*, 25(2):135–138, Mar. 1992.
5. J. M. Cohen, J. F. Hughes, and R. C. Zeleznik. Harold: A world made of drawings. In *Proceedings of the First International Symposium on Non Photorealistic Animation and Rendering*, pages 83–90. ACM Press, June 2000.
6. J. M. Cohen, L. Markosian, R. C. Zeleznik, J. F. Hughes, and R. Barzel. An interface for sketching 3D curves. In *Proceedings of the Conference on the 1999 Symposium on interactive 3D Graphics*, pages 17–22. ACM Press, Apr. 1999.
7. M. F. Deering. HoloSketch: a virtual reality sketching/ animation tool. *ACM Transactions on Computer-Human Interaction*, 2(3):220–238, Sept. 1995.
8. L. Eggli, B. D. Brüderlin, and G. Elber. Sketching as a solid modeling tool. In *SMA '95: Proceedings of the Third Symposium on Solid Modeling and Applications*, pages 313–322. ACM, May 1995.
9. L. Eggli, C. Hsu, B. D. Brüderlin, and G. Elber. Inferring 3D models from freehand sketches and constraints. *Computer-aided Design*, 29(2):101–112, 1997.
10. B. J. Hale, R. P. Burton, D. R. Olsen, and W. D. Stout. A three-dimensional sketching environment using two-dimensional perspective input. *Journal of Imaging Science and Technology*, 36(2):188–195, Mar.–Apr. 1992.
11. A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *SIGGRAPH 00 Conference Proceedings*, pages 517–526. ACM Press, Aug. 2000.
12. T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: a sketching interface for 3D freeform design. In *SIGGRAPH 99 Conference Proceedings*, pages 409–416. Addison Wesley, Aug. 1999.
13. H. Lipson and M. Shpitalni. Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Computer-aided Design*, 28(8):651–663, 1996.
14. L. Markosian, M. A. Kowalski, S. J. Trychin, L. D. Bourdev, D. Goldstein, and J. F. Hughes. Real-time nonphotorealistic

Figure 11: *An example of annotation in 3D: annotating a heart model during an anatomy course. The text displayed is view-dependent. An unimplemented solution would consist in drawing it on a billboard, or using more sophisticated schemes*[16].

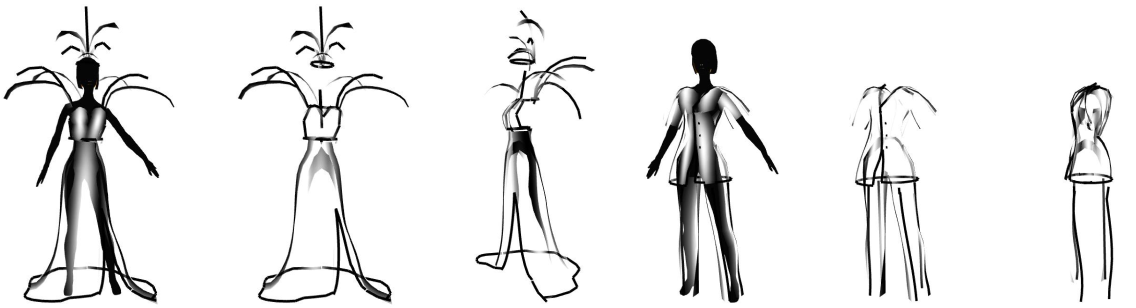


Figure 12: *Using a 3D model as a guide can be useful in fashion applications.*

rendering. In *SIGGRAPH 97 Conference Proceedings*, pages 415–420. Addison Wesley, Aug. 1997.

15. T. B. Marshall. The computer as a graphic medium in conceptual design. In R. Kensek and D. Noble, editors, *Computer Supported Design in Architecture (The Association for Computer Aided Design in Architecture ACADIA'92)*, pages 39–47, 1992.

16. B. Preim, A. Ritter, and T. Strothotte. Illustrating anatomic models — A semi-interactive approach. *Lecture Notes in Computer Science*, 1131:23–32, 1996.

17. D. Pugh. Designing solid objects using interactive sketch interpretation. *Computer Graphics*, 25(2):117–126, Mar. 1992.

18. R. Raskar and M. Cohen. Image precision silhouette edges. In *SI3D 99 Conference Proceedings*, pages 135–140. ACM Press, Apr. 1999.

19. E. Sachs, A. Roberts, and D. Stoops. 3-draw: A tool for designing 3D shapes. *IEEE Computer Graphics and Applications*, 11(6):18–26, Nov. 1991.

20. P. J. Schneider. An algorithm for automatically fitting digitized curves. In A. S. Glassner, editor, *Graphics Gems*, pages 612–626. Academic Press, 1990.

21. P. J. Schneider. Solving the nearest-point-on-curve problem. In A. S. Glassner, editor, *Graphics Gems*, pages 607–611. Academic Press, 1990.

22. T. Strothotte, B. Preim, A. Raab, J. Schumann, and D. R. Forsey. How to render frames and influence people. *Computer Graphics Forum*, 13(3):C/455–C/466, 1994.

23. I. E. Sutherland. Sketchpad: A man-machine graphical communication system. In *Proceedings AFIPS Spring Joint Computer Conference*, volume 23, pages 329–346, Detroit, Michigan, May 1963.

24. O. Tolba, J. Dorsey, and L. McMillan. Sketching with projective 2D strokes. In *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology*, pages 149–158. ACM Press, Nov. 1999.

25. R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. SKETCH: An interface for sketching 3D scenes. In *SIGGRAPH 96 Conference Proceedings*, pages 163–170. Addison Wesley, Aug. 1996.