Sketching, Scaffolding, and Inking: A Visual History for Interactive 3D Modeling

Ryan Schmidt* University of Toronto
Tobias Isenberg† University of Calgary
Pauline Jepp† University of Calgary
Karan Singh* University of Toronto
Brian Wyvill‡ University of Victoria

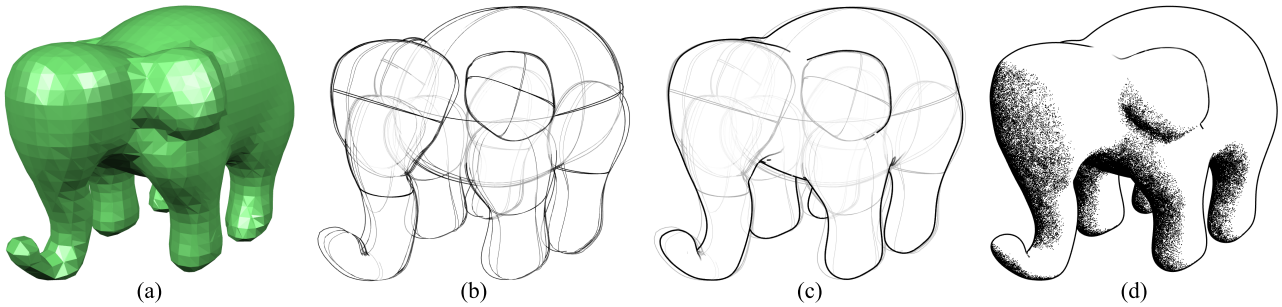


Figure 1: *We begin with a coarse tessellation of an implicit surface (a). The implicit model is defined by a hierarchy of simple components, which are visualized using* geometric massing *(b). These elements of* visual scaffolding *are composited with a real-time pen-and-ink rendering system (d), providing a visual history of the interactive modeling process (c).*

Abstract

Inspired by the comic-book production pipeline, a method is proposed for integrating visual aspects of the sketching process into 3D sketch-based modeling systems. In particular, artist-drawn construction aids called *visual scaffolding* are explored. Two scaffolding components which simulate elements of pencil sketching, *geometric massing* and *eraser marks*, are integrated into a rendering pipeline which also includes a suite of new object-space techniques for high-fidelity pen-and-ink depiction of implicit surfaces. Based on a hybrid, hierarchical approach which leverages both the implicit surface definition and an underlying coarse tessellation, new methods are described for computing silhouettes, suggestive contours, surface stippling, and surfel-based visibility culling. These techniques are real-time but require no pre-computation, allowing them to be applied to dynamic hierarchical implicit surfaces, and are demonstrated in ShapeShop, an interactive sketch-based modeling tool. The result is a real-time display pipeline which composites these novel scaffolding and pen-and-ink techniques to depict a visual history of the modeling process.

Keywords: sketching, visual history, visual scaffolding, geometric massing, eraser marks, interactive pen-and-ink depiction, implicit surfaces, sketch-based modeling

1 Introduction

Sketch-modeling systems, which attempt to replace the standard "menus-and-control-points" 3D modeling workflow with a more natural stroke-based interface, are growing in popularity and capability. Early systems such as Teddy [Igarashi et al. 1999] have been turned into computer games, and recent systems such as ShapeShop [Schmidt et al. 2005] can be used to create a wide range of complex 3D models. One of the goals of sketch-modeling is to more naturally support traditional sketching workflows. Designers often prefer sketching to 3D modeling in part because sketches are more likely to interpreted as "works in progress," rather than final models [Schumann et al. 1996]. Some sketch-based modeling systems have attempted to simulate this property with "sketchy" rendering styles [Igarashi et al. 1999].

However, like all aspects of design, sketching is a process of refinement, compositing, and correction. With traditional sketching, there is an inherent visualization of this process. Sketch-modeling tools do away with this notion - even when non-photorealistic depiction is available, it is cast as just another "rendering mode," and the current 3D model is presented as if it simply popped into existence. This prevents the designer from seeing the previous choices and errors which led to the current model.

Our goal is to improve support for traditional sketching workflows by simulating the visual output produced *during* the sketching process. Since "the sketching process" is difficult to characterize in general, we look to a domain where the pipeline is highly specified - namely, comic books and graphic novels [McCloud 1994]. Classic texts such as Lee and Buscema's "How To Draw Comics the Marvel Way" [1984] lay out a series of steps for producing consistent, high-quality drawings of figures and objects, guidelines which have been applied by generations of comic book artists. The technique prescribes incremental addition of detail, beginning with a coarse *geometric massing* of the basic shape using simple geometric components (Figure 2). Sketch-modeling workflows follow a similar process, where 3D models are constructed by incrementally sketching components. Hence, we visualize the models using a rendering style based on geometric massing (Figure 1).

When sketching with pencil, erasing a line often leaves behind a faint *eraser mark*, which is a useful guide for correcting mistakes. We model this visual effect by faintly rendering deleted components. We characterize geometric massing and eraser marks as elements of *Visual Scaffolding*, which encompasses visual elements that are not part of the "final" surface, but correspond to artist-constructed visual aids and other marks that one would see when sketching on paper. Our display of visual scaffolding reduces the gap between sketching interfaces and traditional sketching.

*e-mail: {rms|karan}@dgp.toronto.edu
†e-mail: {isenberg|pj}@cpsc.ucalgary.ca
‡e-mail: blob@cs.uvic.ca

The final step in the comic-art production pipeline is "inking," where the artist draws a pen-and-ink image over the pencil sketch, in effect using the entire sketch as visual scaffolding. We model this stage as well, by layering a real-time pen-and-ink depiction of the current 3D surface on top of the geometric massing and eraser marks. This allows for simultaneous visualization of both the smooth surface and the underlying internal model structure.

We implement our rendering pipeline in ShapeShop [Schmidt et al. 2005], a sketch-modeling system based on hierarchical implicit volume models [Wyvill et al. 1999]. Algorithms for pen-and-ink rendering of implicit surfaces are non-interactive on complex models [Foster et al. 2005; Stroila et al. 2007], so we develop a suite of novel techniques for real-time pen-and-ink depiction of dynamic smooth surfaces. Briefly, we find low-fidelity silhouette and suggestive contours by applying brute-force object-space algorithms to a coarse *base mesh* which approximates the smooth surface. These contours are incrementally refined and projected to the surface. We adapt surfel techniques to both remove hidden lines and generate stippling. A hierarchical data structure provides efficient visibility culling and dynamic instantiation, and view-adaptive techniques ensure that contours remain smooth when zooming in.

The main benefit of our pen-and-ink renderer is that it provides real-time performance on complex 3D surfaces which are being interactively modified. Previous techniques generally require costly mesh pre-processing which must be repeated each time the surface changes [Hertzmann and Zorin 2000]. Our approach is designed to work with the coarse tessellations found in current 3D modeling systems (Figure 6), and hence can be easily integrated into these tools. Note also that while we describe our algorithms in the context of an implicit representation, they can be easily applied to other functional smooth representations such as subdivision [Stam 1998] and parametric spline [Foley et al. 1996] surfaces.

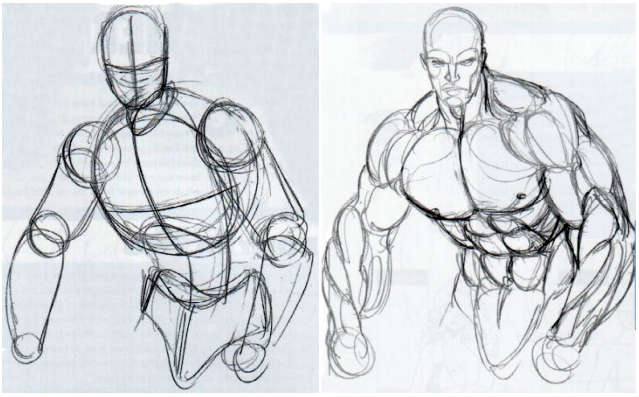


Figure 2: *Character sketching is often an iterative process. The artist begins with a rough geometric "massing" (leftmost image), which helps ensure correct proportions and perspective. The detailed character is then sketched over top of this visual scaffolding. (Images © Wizard Entertainment, Inc).*

2 Related Work

There has been little exploration of what we refer to as visual scaffolding in the graphics literature. 3D construction planes are a basic form of visual scaffolding which appear in many systems. Recently, [Ijiri et al. 2006] employed transient visual scaffolding in the form of 3D concept sketches, which are then progressively refined, in the context of a plant-modeling system. Static visual scaffolding in the form of instructive visual aids has been explored in education research [Khusro et al. 2004], and the concept of visual history has seen some use in 2D document editing systems, particularly for revision visualization and selective "undo" [Kurlander and Feiner 1990; Hardock et al. 1993]. A variety of techniques have been explored for visualizing recent interaction history in the HCI literature [Baudisch et al. 2006; Bezerianos et al. 2006]. Some commercial CAD systems support revision tracking, but generally this is limited to displaying the "difference" between two models.

Depicting 3D shapes with line and contour drawings has a long history in computer graphics. CAD systems often employ feature lines to visualize 3D models [Requicha and Voelcker 1982]. Technical limitations initially made this necessary, however current CAD systems still include these line-based rendering modes, as they make it possible to visualize internal structure in a way that is difficult to mimic with shaded surfaces. While transparency can be useful for this task, transparent objects become difficult to differentiate when many are overlapping [Harrison et al. 1995].

Early line-drawing techniques for representing 3D surfaces attempted to simulate the lines that an artist would draw, focusing on silhouette and feature lines [Saito and Takahashi 1990]. These methods evolved into more general techniques for simulating pen-and-ink drawing, a traditional method of artistic depiction. In addition to feature lines, pen-and-ink techniques include stippling [Deussen et al. 2000; Secord 2002] and hatching [Salisbury et al. 1994; Winkenbach and Salesin 1994] to depict shading and texture of the surface. Extensive work on generating pen-and-ink-style renderings [Winkenbach and Salesin 1996; Salisbury et al. 1997; Hertzmann and Zorin 2000; Zander et al. 2004; Foster et al. 2005] has resulted in very high-quality images [Isenberg et al. 2006]. Most of these techniques require significant computation time, making them inapplicable to interactive modeling. Real-time methods are available [Freudenberg et al. 2001; Praun et al. 2001; Fung and Veryovka 2003], but involve pre-computed surface-parameterizations which limit them to static models.

Object-space mesh silhouettes are a key component of our system, and a variety of existing techniques are available [Isenberg et al. 2003]. We use the simplest brute-force methods, as more advanced algorithms either require costly pre-processing [Hertzmann and Zorin 2000], or are randomized [Markosian et al. 1997] and hence cannot guarantee frame-coherence. Kirsanov et al. [2003] describe an interesting method in which a high-resolution mesh is progressively simplified, allowing silhouettes to be efficiently computed on the low-resolution mesh and then mapped back onto the high-resolution mesh. Again, the mesh simplification is pre-computed, limiting this technique to static models. We take a similar approach, although we do not initially have a high-resolution mesh, but instead an implicit surface.

A variety of works have addressed the problem of computing silhouettes, feature curves, and other pen-and-ink elements on implicit surfaces. Bremer and Hughes [1998] used incremental techniques to trace silhouette loops around simple implicit models, while Elber [1998] projected points from a base mesh onto the implicit surface and created strokes using this point distribution. We adapt this projection method, initially described by [Meier 1996], in Section 4. Foster et al. [2005] built on these tracing and particle-based techniques, creating a full pen-and-ink rendering system for complex hierarchical implicit models. Jepp et al. [2006] have extended this system using flocking techniques to render additional surface contours. Stroila et al. [2007] describe a robust mathematical framework based on implicit-surface intersection to compute silhouette and suggestive contours, although C^3 continuity is assumed. None of these techniques are guaranteed to find all silhouettes, doing so on an implicit surface requires interval algorithms [Plantinga and Vegter 2006], however this is very costly. Burns et al. [2005] do provide real-time techniques for contour extraction on high-

resolution volume data, but the 3D sampling required to convert our implicit models into this representation is very time-consuming.

While some of these techniques are capable of interactive performance, few scale to complex models, and none would maintain real-time performance on complex implicit models which are being interactively modified. Particle-based methods can attempt to interactively "track" the surface, but significant changes (particularly topology change) are still very costly.

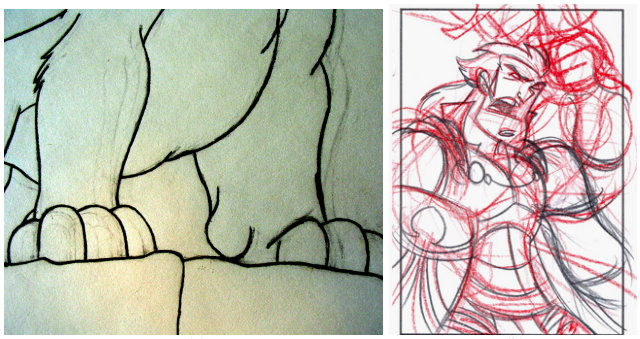
3 Visual Scaffolding

Most work in computer-generated artistic depiction focuses on simulating the artist's finished work - the watercolor painting, cartoon rendering, or pen-and-ink image. However, artists rarely produce final pieces "from scratch." Generally a constructive process is followed, often beginning with "concept sketches" of varying levels of detail. Even at the sketching stage, there are often conventions for building up the drawing. An interesting case study is the domain of comic books and graphic novels [McCloud 1994].

The production of comic book art follows a well-defined construction pipeline prescribed by classic texts such as [Lee and Buscema 1984]. Human figures begin as a sketched assembly of geometric shapes, similar to the "massing" models common in architectural design. The artist then sketches over top of this first pass with incrementally more detailed sets of contours, using the geometric shapes as a guide to maintain proportions and perspective (Figure 2). The initial geometric sketch is not part of the final drawing, parts of it which remain visible are usually erased. Hence, it can be considered an artist-generated visual construction aid, a particular element of what we call *visual scaffolding*.

After the highly-detailed sketch is complete, it is turned into a pen-and-ink image and possibly colored. The inking step is often performed by a separate artist, the "inker," who does not simply trace over the initial sketch, but rather uses it as a guide to produce a compelling pen-and-ink depiction. So, to the inker, the entire sketch is visual scaffolding.

The faintly visible marks left by erasing pencil-drawn lines are another element of visual scaffolding. If the design space of a sketch is thought of as a tree, then these *eraser marks* show branches which were partially explored and then abandoned. This information is useful to the artist, as previous errors are visible and can be used to make iterative adjustments to a sketch. An example is shown in Figure 3a. Some artists also carry out design experiments simply by over-sketching, as in Figure 3b.



(a) (b)

Figure 3: *Erased "mistakes" are still visible in the scanned drawing in (a) (image contrast has been increased to make the faint lines more visible). In (b), a red pencil has been used to explore possible variations on the sketch (image (b) © Mike Krahulik).*

The main point is that visual scaffolding is a key element of the design process, used by the artist at all stages except when displaying the final work. In addition, this exploration and refinement of design space is often iteratively accumulated in a single image. It is interesting to compare this process to traditional 3D modeling pipelines, where such visual history is very rare. That mistakes can be made to instantly disappear is a ubiquitous "feature" of computer graphics tools. Similarly, there is very little support for visualizing the intermediary construction states of a 3D model, particularly in an in-line fashion. One reason for this is that smooth shaded surfaces are difficult to adapt to the task. Visual scaffolding is much easier to display in combination with pen-and-ink depiction, where there is more "white space" to work with.

3.1 Geometric Massing

Automatically generating a *geometric massing* image such as Figure 2 for an arbitrary 3D model is non-trivial, in part because most surface representations do not contain an intrinsic decomposition into simple geometric shapes. However, many sketch-based modeling systems take a constructive approach, allowing the designer to incrementally assemble a model by drawing individual pieces. This approach lends itself well to the display of geometric massing.

After examining a range of samples such as Figure 2, we have formulated some guidelines for geometric massing. Hidden lines tend not to be drawn on individual massing elements, but separate elements are composited without any hidden line removal. In addition to silhouette contours, planar contours passing through the "center" of the geometric element are common. Generally only a subset of these contours are drawn. Guidelines for this decision are difficult to characterize, but some heuristics are described below.

We render goeometric massing by applying the contour-extraction algorithms described in the next section. Each element has its own base mesh, which is generally static and of low complexity. An oriented bounding box is fit to the base mesh to determine the major axes, from which the planar contours can be pre-computed. We keep only the planar contours which have extents approximately equal to the respective bounding box dimension. In addition, we cull planar contours which are nearly co-planar with the view plane (and fade them in as they become visible, to avoid "popping"). Silhouettes are extracted using the techniques described in Section 4.1. To mimic the "sketchyness" typically found in geometric massing, we draw each contour several times with random perturbations to scale, angle, line width, and darkness.

An example of our geometric massing is shown in Figure 4. We have also experimented with showing depth ordering of the internal lines by modulating darkness based on layers of occlusion, which is visible in Figure 4. There is some motivation for this in traditional sketching, as some texts suggest that internal lines be erased as the user adds detail. However, we apply it primarily because we have found that in a 3D context it greatly assists with determining the relative spatial position of the massing elements.

3.2 Eraser Marks

Eraser marks provide an interesting visualization of a sketch's history. Providing a similar sort of visual feedback in an interactive modeling system is both unprecendented and somewhat daunting. The problem is one of determining what data is useful to display, without overloading the user. For example, since paper does not support "drag-and-drop," we do not show eraser marks when the user makes incremental changes to the model elements. Instead, eraser marks are only shown when nodes in the implicit model are deleted. Likewise, eraser marks on paper show what was drawn

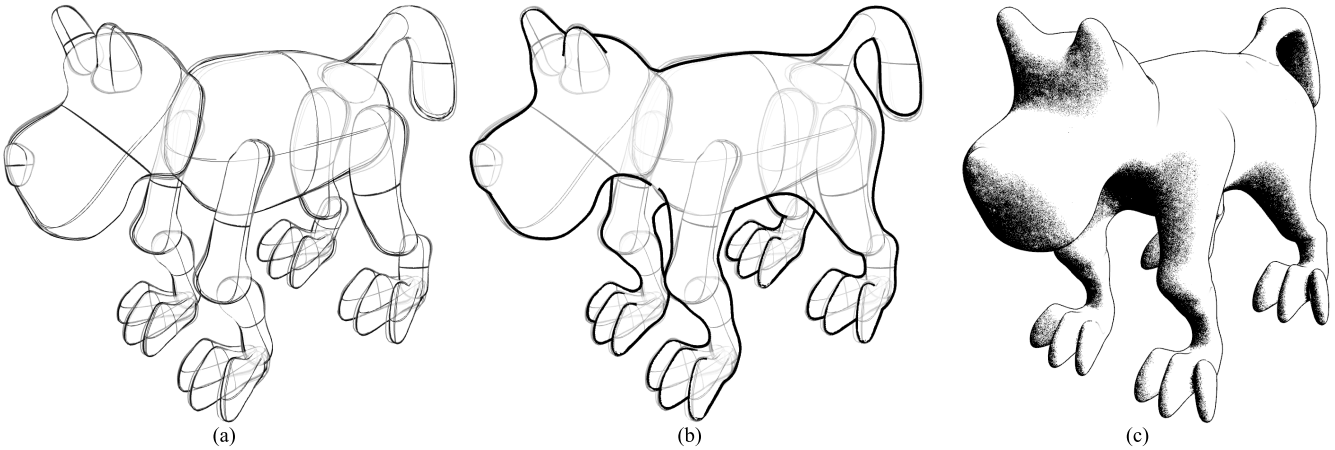


Figure 4: *Geometric massing of dog model (a) and overlaid with silhouette (b). The contrast of the massing lines has been increased to improve visibility in print. In (c), a high-quality pen-and-ink image is shown with suggestive contours and stippling.*

with the pencil. In our system, the user does not directly draw the surface silhouette, but instead draws 2D contours which define 3D objects that in turn modify the 3D surface. Hence, instead of showing eraser marks for the surface, we show eraser marks for what was actually drawn - namely, the underlying implicit primitive. This more closely reflects the intent of eraser marks, which is to assist designers when trying to correct mistakes. While these design choices seem most appropriate for ShapeShop, there are a variety of other ways eraser marks could be used, particularly with respect to interactive surface manipulation. Exploration of these issues is left for future work.

An eraser mark is depicted in Figure 5. We currently show the eraser mark from all viewing directions, which is satisfactory for simple models, but may result in excessive visual clutter during a long modeling session. In that case we could display an eraser mark only when the viewing direction is near the one it was drawn from. Another option would be to assume that the relevance of eraser marks decreases over time, and fade them out to invisibility. This is somewhat physically accurate, as the stray graphite that sketches build up over time tends to eventually obscure eraser marks.

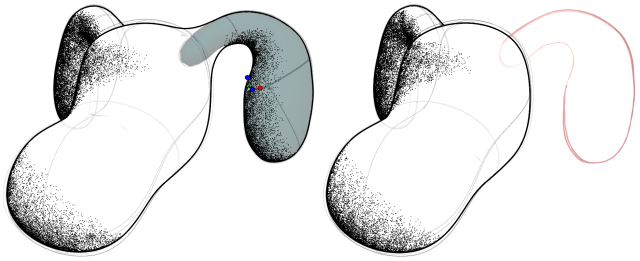


Figure 5: *The smooth implicit surface depicted in the left image is the result of blending three components. On the right, the selected ear component has been erased, leaving behind an eraser mark.*

4 Real-Time Pen & Ink on Implicit Surfaces

Conceptual 3D design tools are a promising application area for interactive pen-and-ink visualization. In this context, it is critical to support high-fidelity visual comprehension of smooth surfaces, while also conveying the impermanent nature of the current 3D shape [Schumann et al. 1996]. By simplifying the visual representation, pen-and-ink renderings support both of these goals. However, current interactive algorithms for pen-and-ink rendering of 3D models assume that the usage scenario is exploration of a static, high-resolution surface mesh [Hertzmann and Zorin 2000; Zander et al. 2004]. This is problematic on two fronts - first, interative modeling tools generally attempt to guarantee fast visual feedback during model editing by tessellating smooth surfaces at low resolutions. For example, standard NURBS tessellation resolutions from the commercial tool Maya are shown in Figure 6. The argument can be made that interactive resolution will scale with computing power, but, historically, this has been counter-balanced by increasing model complexity. The second issue is that, even if high-resolution surfaces could be generated, these pen-and-ink algorithms involve a significant amount of pre-processing. If the designer is interactively manipulating the surface, this pre-processing must be repeated each frame - a significant computational burden.

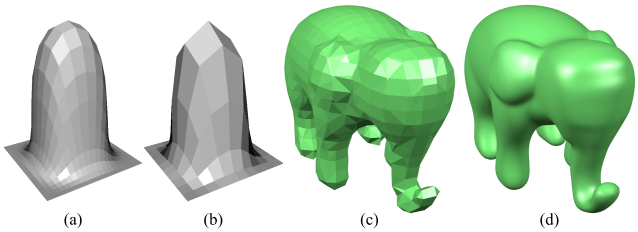


Figure 6: *Standard mesh resolutions used to interactively display a NURBS patch in Maya at high (a) and medium (b) resolution, and interactive mesh resolution (c) for a moderately complex ShapeShop implicit model (d).*

The meshes in Figure 6 approximate the underlying smooth surfaces rather poorly, and the visual fidelity of any mesh-based pen-and-ink algorithm is similarly limited by coarse tessellations. More problematic is that the designer relies heavily on iterative visual feedback to create the desired shape, and must estimate the smooth-surface shape from these rough meshes. We address both of these problems at once, by developing a suite of hybrid techniques which leverage both the base mesh and the functional smooth-surface definition. These techniques provide interactive pen-and-ink rendering as the designer edits the model, as well as more accurately representing the shape of the smooth surface.

4.1 Silhouettes

Given an arbitrary smooth surface $\mathcal{S} \in \mathbb{R}^3$, any point $\mathbf{p} \in \mathcal{S}$ is considered "on" the silhouette if it satisfies the equality

$$\mathbf{n} \cdot \mathbf{v} = 0 \tag{1}$$

where $\mathbf{n}$ is the surface normal at $\mathbf{p}$, and $\mathbf{v} = \mathbf{c} - \mathbf{p}$ is the *view vector* to the camera position $\mathbf{c}$. Note that Equation 1 fails at creases, where $\mathbf{n}$ is not defined, but in that case one can consider a a limiting process which approaches the crease on either side.

Finding solutions to Equation 1 on a complex smooth surface is not feasible in an interactive system. And, as we have noted, simply using the interactive mesh, which we will call the *base* mesh, is unsatisfactory. Hence, we use both - our general approach is to find coarse silhouette contours on the base mesh, and then project them onto the smooth surface. While this is only an approximation, we have found it to be very effective.

The first step is to find object-space silhouette contours on the base mesh. We use the basic brute-force approach, evaluating $\mathbf{n} \cdot \mathbf{v}$ at each vertex of the mesh and extracting all edges whose vertices have opposite signs. It is critical to our method that $\mathbf{n}$ be accurately computed from from the smooth surface definition - normals approximated from the coarse base mesh are much too noisy.

These initial edges are a very coarse approximation to the silhouette contour, so a simple refinement is to find the zero-crossing along each edge, and connect the "mid-edge" points with lines lying across the triangle faces [Hertzmann and Zorin 2000]. These *sub-polygon* silhouettes provide an accurate approximation of the base mesh silhouette, an example is shown in Figure 7a.

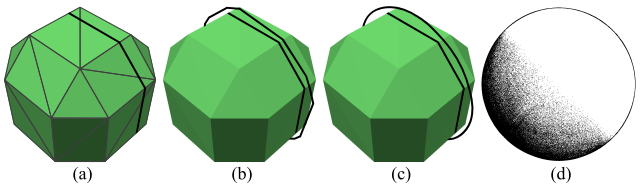


Figure 7: *Smooth-surface silhouettes are approximated by computing sub-polygon silhouettes on a coarse base mesh (a), projecting the vertices of this coarse silhouette onto the actual smooth surface (b), and subdividing (c). The resulting silhouette is shown from the camera viewpoint in (d), with stippling.*

The sub-polygon silhouette is represented as a list of edges, one per triangle. The next step is to project the vertices of these edges onto the smooth surface, as shown in Figure 7b. The details of this projection depend on the smooth surface representation. On a NURBS patch, one maps from the 3D mid-edge points into the uv-space of the patch, and then back onto the NURBS surface. Our implementation focuses on implicit surfaces, where a scalar function $f(\mathbf{p})$ defines a scalar field over $\mathbb{R}^3$, and the equality $f(\mathbf{p}) = v$ defines the *implicit surface*, where v is the *iso-value*. The 3D vector $\nabla f(\mathbf{p})$ of partial spatial derivatives of f, known as the *gradient*, points in the same direction as the surface normal when $\mathbf{p}$ lies on the surface, and otherwise points towards the surface. Hence, to project a point onto the surface, we "walk" in the direction of the gradient, updating $\mathbf{p}$ using the following *convergence iteration*:

$$\mathbf{p} \leftarrow \mathbf{p} + \frac{(f(\mathbf{p}) - v)\nabla f(\mathbf{p})}{\|\nabla f(\mathbf{p})\|} \tag{2}$$

Once the coarse silhouette has been been projected onto the surface, we refine it by iteratively subdividing each silhouette segment until a minimum-length threshold is met (Section 4.5), projecting new vertices to the surface as they are generated (Figure 7c).

The convergence properties of Equation 2 are difficult to characterize in general [Schmidt 2006]. Since we are interested in a fast approximation, we assume f is well-behaved and only iterate twice for the initial projection, and once during subdivision. We also check that the initial projections are valid - if $|f(\mathbf{p}) - v|$ has grown, we try stepping by half the initial distance, and if that fails, we discard the segment, rather than risk displaying an invalid result.

In general, our projection/refinement technique is reasonably robust, even with coarse, low-quality base meshes. Some visual comparisons are shown in Figure 8, where our refined silhouettes are compared to sub-polygon silhouettes on a high-resolution mesh. The approximations are quite accurate, with no major errors, even though the input data (the coarse mesh silhouette) generally contains significant deviations from the desired result (Figure 8d).

4.2 Suggestive Contours

Silhouette contours alone provide only basic information about object shape. Artists frequently convey the shape of more subtle surface curvature by drawing additional contours. Attempts to formalize these contours led to the notion of a *suggestive contour* [DeCarlo et al. 2003]. Informally, suggestive contours are curves which would be silhouette contours in nearby viewpoints. Mathematically, consider the projection of the view vector $\mathbf{v}$ onto the tangent plane at $\mathbf{p}$. This gives a vector $\mathbf{w}$ which, along with $\mathbf{n}$, defines the *radial plane* at $\mathbf{p}$. Taking the intersection between this plane and the surface near $\mathbf{p}$ produces a 2D curve lying in the radial plane, the curvature of which is is the *radial curvature* κ_r. Suggestive contours can then be defined as points where $\kappa_r = 0$ and the directional derivative of κ_r in the direction of $\mathbf{w}$ is positive:

$$D_{\mathbf{w}}\kappa_r > 0 \tag{3}$$

Like our silhouette contours, we find suggestive contours in object space. We evaluate κ_r and $D_{\mathbf{w}}\kappa_r$ at base mesh vertices (using the scalar field f, as mesh-based techniques are too noisy on our base meshes), and then apply the sub-polygon and projection-refinement techniques. Finding good suggestive contours does tend to require a base mesh with somewhat higher resolution, but generally our results are similar to those found using a high-resolution mesh (Figure 9). Computing $D_{\mathbf{w}}\kappa_r$ from our scalar fields is problematic because we do not have the high-order continuity that is assumed in [Stroila et al. 2007]. Our fields are only C^1, so we have developed a modified finite difference approximation, see Appendix A.

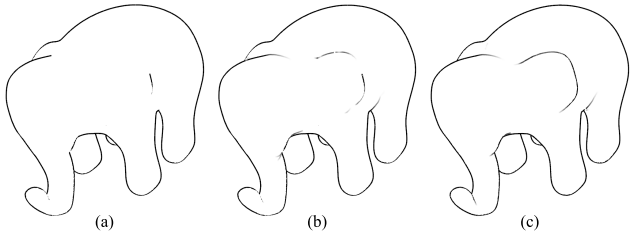


Figure 9: *Comparison of suggestive contours computed using (b) low-resolution base mesh and (c) high-resolution mesh. Image (a) shows result without suggestive contours.*

4.3 Stippling and Visibility Culling

Artists working in the pen and ink medium use various techniques to depict interior shading on smooth surfaces, including small dots

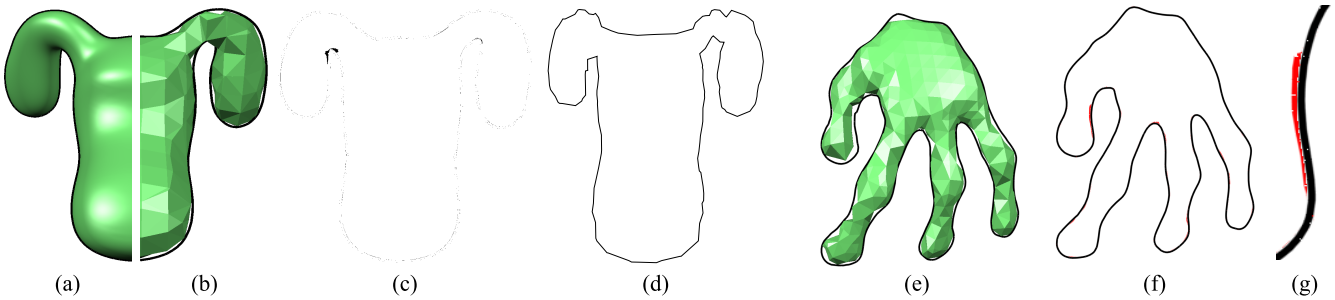


Figure 8: *An accuracy comparison on a simple model. The high-resolution mesh silhouette is shown in (a), and our technique in (b). The difference image of the two silhouette contours is shown in (c). The coarse mesh's sub-polygon silhouette before refinement is shown in (d). In (f), the high-resolution silhouette is rendered in black, and our refined silhouette is shown underneath in red. The largest difference is visible on the inside of the thumb, see detail in (g).*

of varying density, called *stippling*, and short *hatching* strokes. We support such elements in our rendering system by covering the surface with small area-elements, called *surfels* [Pfister et al. 2000], which can be thought of as local canvasses onto which stipples and strokes can be drawn. We focus on object-space stipple-shading, distributing stipple points evenly across each surfel and performing visibility culling at run-time. Ideally, a distribution such as Poisson-disk [Dunbar and Humphreys 2006] would be used, however we found that a jittered regular distribution is generally adequate. The surfel distribution is far more critical, as explained below.

Our stipple visibility culling attempts to simulate diffuse shading, so we define the vector $\mathbf{l}$ which points towards the "light," and cull stipples based on the value of $\mathbf{n} \cdot \mathbf{l}$. The stipple normal $\mathbf{n}$ is taken from the surfel it lies on. We define an upper threshold $t_u \in [0, 1]$, and cull entire surfels if $\mathbf{n} \cdot \mathbf{l} \geq t_u$. If a surfel passes the culling test, we then need to determine which stipples to render. The density should vary based on $\mathbf{n} \cdot \mathbf{l}$, so we assign a random value $r \in [0, 1]$ to each stipple when it is created, and only render stipples if:

$$r^k > (1 - \mathbf{n} \cdot \mathbf{v}) \tag{4}$$

where k controls the shading falloff (we use $k = 2$).

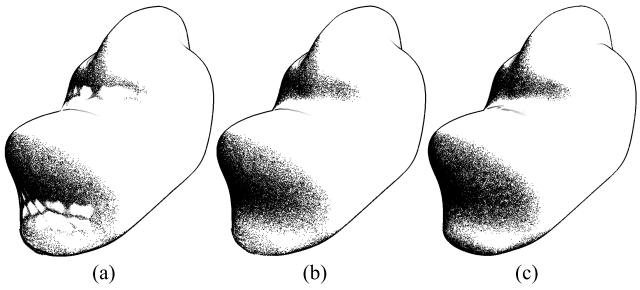


Figure 10: *Z-buffer visibility culling using the low-resolution base mesh (a), surfels (b), and high-resolution mesh (c) for comparison. Artifacts occur primarily in concave regions of (a).*

Each stipple and contour segment must also be tested to determine whether it is currently occluded by some portion of the surface nearer to the eye. A common mesh-based solution is to render the mesh into the z-buffer, and rely on depth-testing hardware to determine visibility. Unfortunately this approach gives very poor results with our low-resolution base mesh (Figure 10a). Another option is ray-casting, which is accurate but too expensive for interactive use with implicit surfaces. Instead, we adopt Foster et al.'s [2005] approach of rendering the surfels into the z-buffer. Foster et al. [2005] used particle repulsion to distribute surfels across the implicit surface, seeding the algorithm with rather costly ray-surface intersections. More robust algorithms are available [Meyer et al. 2005], however they are not fast enough for interactive use on complex deforming surfaces. Instead, we adapt the method proposed in [Meier 1996] of distributing surfels on the base mesh triangles and projecting them to the surface.

Our main challenge in surfel generation is efficiently producing a distribution of surfels which both covers the surface (for proper visibility culling) and also has a semi-regular distribution (to avoid stippling patterns). First, we must produce a suitable surfel distribution on the base triangles. Meier [1996] randomly distributed points, however this does not guarantee coverage of the surface. Again, a Poisson-disk distribution would be ideal, but we are highly constrained by computation time. We initially tried a subdivision scheme, doing 1-to-4 splits with an area threshold, but this lead to highly varying density across neighbouring triangles (Figure 11b). Varying density is a visual cue for surface curvature, so these artifacts are likely to mislead the viewer. We achieve smoother results (Figure 11c) by projecting the triangle into 2D and rasterizing it, replacing pixels with surfels, which allows many surfels to be generated very quickly. Jittering can be added, however we have not noticed major visual improvements by doing so.

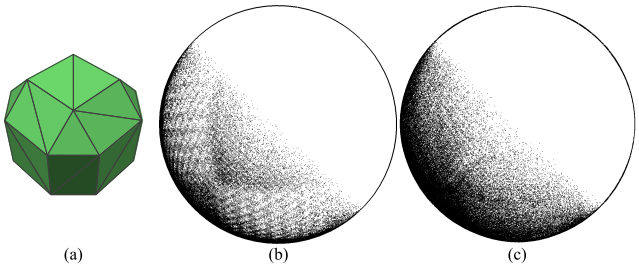


Figure 11: *Varying stipple density arising from non-uniform surfel distribution of a low-resolution base mesh (a) can produce undesirable patterns (b). Surfel "rasterization" results in more uniform sampling (c).*

If the surfels are distributed in the planar triangle using a regular grid with spacing $\triangle$, they will "spread out" when the are projected onto the smooth surface, resulting in gaps (Figure 12a,c). This also occurs with the random distribution of [Meier 1996]. To correct for this, we rely on a simple heuristic. Along each edge of the triangle, we know the chord length c and the angle θ between the vertex normals. Hence, we can approximate the length of the projected

edge using a circular segment with arc length s (Figure 12a):

$$s = \left(\frac{c}{2\sin(\theta/2)}\right)\theta \tag{5}$$

The ratio s/c is computed for each triangle edge, and the largest value gives us a uniform scaling factor for the triangle density. However, the spreading is non-uniform, so we must correct for this with non-uniform grid spacing. If our grid is aligned with one edge of the triangle, as in Figure 12b, then we step along the X axis as follows:

$$x \leftarrow x + \triangle\left(1 - k_{\triangle}\left(1 - t_x^2\right)\left(\frac{s}{c} - 1\right)\right) \tag{6}$$

Here, $t_x = |x - x_m|/(x_m - x_l)$ is used to apply a quadratic falloff to the scaling factor $s/c - 1$ as x gets further from the midpoint of the projection of the triangle on the x-axis (see Figure 12b). Since the projection is not necessarily circular, we oversample to ensure that the surface remains covered. The constant $k_{\triangle}$ controls oversampling, we use $k_{\triangle} = 4$. This scaling must also be applied in the y-direction.

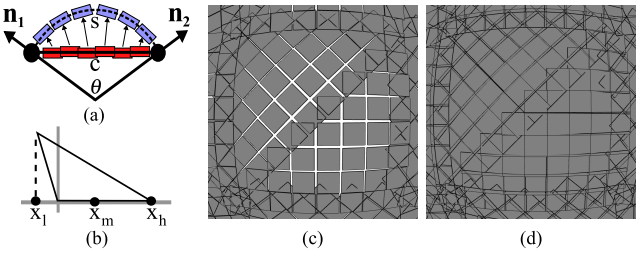


Figure 12: *An overlapping surfel sampling on the base triangle can contain gaps when projected onto the surface (a,c). Non-linear scaling of the sampling density minimizes this effect (d).*

4.4 Hierarchical Culling with Dynamic Expansion

The pen-and-ink rendering pipeline we have described involves a large amount of data. We have a low-resolution base mesh, dynamic contours, surfels, and stipple particles, all of which must be re-generated each frame if the model is changing. To minimize the amount of data generated and processed, we organize the various elements using a hierarchical data structure. At the root of the hierarchy is the base mesh, which contains a set of base triangles (Figure 13a). Each base triangle stores a set of surfels, and each surfel in turn stores its stipple particles (Figure 13b,c).

To optimize rendering, we perform visibility culling at all levels of this hierarchy. We assume surfaces are closed and cull at the triangle level any faces which are completely back-facing. Off-screen and occluded triangles can also be culled. Triangles passing the culling tests have their surfels rendered into the depth buffer. After the z-buffer has been filled, we extract and render contour lines, and then perform lighting-culling at the surfel level using Equation 4. Finally, the stipples lying on the remaining surfels are displayed if they pass the lighting test (Equation 4).

This per-frame hierarchical culling provides a significant performance benefit - we observe 20% to 80% reductions in rendering time when the model is being rotated, with denser stippling generally dictating the rendering cost. To further reduce the computational load, we also use a lazy-evaluation scheme when generating surfels and stipples. Initially we do not expand the hierarchy beyond the base triangle level. If a triangle passes the culling test, its surfels are generated on-the-fly. Likewise, during the stipple-rendering pass, stipples are generated as needed for surfels that pass the light-culling test. We have found that it is also performance-critical to avoid dynamic memory allocation, so surfels and stipples are stored in pre-allocated linear memory buffers which grow in fixed blocks. Using this dynamic expansion scheme, we observe up to 50% reduction in surfel generation time for frames where the base mesh changes. This is a major optimization, as it minimizes the number of surfels which must be projected to the surface, which is the most expensive step in our algorithm. There is a small cost, in that later frames can be slightly more expensive when additional branches of the hierarchy must be expanded.

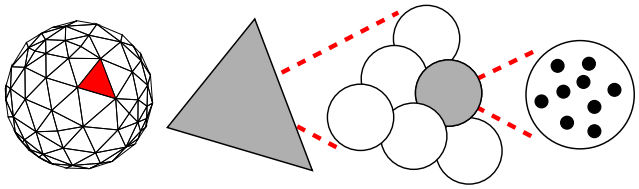


Figure 13: *Each base mesh triangle contains a set of surfels, which in turn contain sets of stipples. This hierachical data structure efficiently supports visibility culling and dynamic data generation.*

4.5 View Independence

One of our goals is to improve the user experience by avoiding the visual faceting one often sees with smooth shading (Figure 6). The intent is similar to view-dependent meshing [Hoppe 1997], but instead of mesh resolution, we automatically control contour subdivision and stipple density as the user zooms in and out.

Visible "faceting" in the silhouette and suggestive contours can be avoided by changing the subdivision error tolerance. It is too costly to determine the optimal subdivision level on a per-segment basis, so we rely on a global measure. We take two 3D points, each a small distance from the model origin and lying in a plane perpendicular to the viewing direction, and project them into 2D pixel coordinates. The distance ρ between the 2D pixels is used as a global view-scaling metric. The subdivision error tolerance is then set to $(1/\rho)k_{sil}$, where k_{sil} is a constant depending on screen resolution (we use $k_{sil} = 2$, for a 1680×1050 display).

There are two other components to consider - surfels and stipples. The same screen-space scaling metric can be used, with ρ controlling surfel and stipple density. We apply this approach for stippling, attempting to maintain a consistent level of intensity as the user zooms in (Figure 14). Our approach requires an integral number of stipples-per-surfel, which we compute as $1 + (\rho/k_{stip}) * k_{dark}$, where k_{stip} is a system-wide hand-tuned constant, and k_{dark} is a user-controllable value that determines the stipple density (and hence the apparent "darkness" of the shading). For reference, we use $k_{stip} = 50$ and begin with $k_{dark} = 2$, with model scale generally on the order of a unit cube.

Currently, we do not dynamically modify the surfel density due to performance limitations. In our application, surfel density is also less critical - primarily the only errors a user will see when zooming in are visibility culling errors in the silhouettes. This is undesirable, but the effect is not too severe, and in some sense is consistent with our desire for "sketchy" output.

4.6 Limitations

Our technique rests heavily on the coarse base mesh. While we do not require that the base mesh have high quality, it must have accurate normals and generally reflect the overall shape of the surface. The exact requirements are highly dependent on the scalar fields in

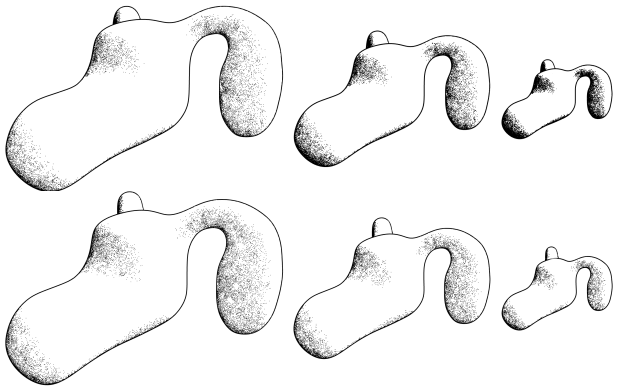


Figure 14: *In the top row, the stipple density remains constant as the camera zooms out, resulting in an increase in apparent darkness from left to right. In the second row, the density is dynamically adapted, resulting in a more uniform appearance at different scales.*

use - roughly, it is necessary that any point on the implicit surface be reachable by a monotonic gradient walk from the base mesh. In addition, it is critical that the functional surface topology is captured by the base mesh. This is problematic, as an implicit-surface meshing algorithm which both guarantees topology, and provides real-time performance, has yet to be discovered. ShapeShop makes no guarantees about mesh quality, relying on the user to increase meshing accuracy when features are missed.

Likewise, there is no guarantee that a point which is on the mesh silhouette will correspond to a silhouette point on the smooth surface, nor is it guaranteed that the projected point will lie on the smooth-surface silhouette. Various iterative correction techniques can be applied [Bremer and Hughes 1998; Stroila et al. 2007], however we found no major visual difference when doing so.

Inter-frame coherence is a problem for any interactive pen-and-ink technique. Since we rely completely on object-space techniques, and store all generated data, contours and stippling are frame-coherent when the base mesh is static (i. e., during rotation). However, the marching-cubes tessellation can change significantly during model manipulation. This usually leaves contours unchanged, but we have no general solution for maintaing stipple coherence. Often the changes are local in nature, so when generating stipples for a triangle we seed the random number generator using the triangle vertex coordinates. This provides coherence in areas where the mesh does not change. Another coherence issue occurs when dynamically adapting the stipple resolution - since we take integer steps, there is visible popping. These problems are unfortunate, but do not seriously impair the usability of the system.

5 Discussion

Current sketch-based 3D modeling tools have focused mainly on "input", allowing designers to leverage their existing sketching skills to enhance the modeling experience. With visual scaffolding, we have begun to explore another dimension of the sketching process. By simulating the visual "output" produced while sketching, we hope to both make computer-based sketching feel more like pencil-and-paper sketching, while also providing superior visualization and interaction capabilities. There are many other aspects of visual scaffolding, such as coarse silhouettes and construction lines (Figure 15), which remain to be explored.

Geometric massing and eraser marks are useful visual guides, and may also result in significant interface improvements. Like most solid modeling tools, one of the key limitations of ShapeShop's interface is that the model is defined by a tree which is difficult to visualize in-line with the shaded-surface display. Geometric massing inherently exposes the set of underlying components which make up the model, and could be adapted to support display (and direct manipulation) of the model tree. Similarly, eraser marks provide a tangible visual history of deleted objects, which could allow the designer to selectively "undo" editing operations. Another direction is the computation of geometric massing for existing mesh models - recent work on ellipsoidal surface approximation may be a reasonable starting point [Simari and Singh 2005].

We also describe a new approach to pen-and-ink depiction of smooth surfaces, including algorithms for finding silhouette contours, suggestive contours, surfel-based visibility testing, and stippling. Our renderer is novel in part because it is the first pen-and-ink technique that can be applied in real-time to smooth surfaces which are being interactively deformed. We have had extensive experience using ShapeShop with this renderer - an early implementation was released as part of ShapeShop v002 in July 2006. The result of one editing session using the techniques described in this paper is shown in Figure 16.

One key advantage of the pen-and-ink renderer is that it enables the designer to visualize the actual smooth surface during interactive editing, something which is not otherwise possible (accurate, high-resolution surface meshing is completely non-interactive). Since our techniques are directly applicable to NURBS and subdivision surfaces, it would be interesting to explore pen-and-ink visualization in interactive modeling tools for those representations.

We have not done extensive evaluation of the impact of visual scaffolding on interactive modeling, but plan on exploring this in the future. The response of ShapeShop users to the pen-and-ink renderer has been quite positive. After a demo video which contained a pen-and-ink editing session was posted to the Internet, users sent e-mails requesting that this feature be released (it already had been, but the menu item that enabled it was rather cryptic). ShapeShop is publicly available, and can be downloaded at http://www.shapeshop3d.com/. The implementation described in this paper can be acquired by contacting the first author.

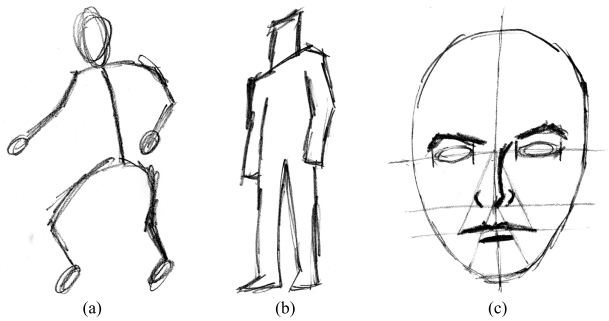


Figure 15: *Sketching textbooks suggest a variety of other visual scaffolding techniques, including stick figures (a), coarse geometric silhouettes (b), and various types of construction lines to assist with drawing complicated objects such as human faces (c).*

Acknowledgements

The authors would like to thank Mario Costa Sousa and Kevin Foster for their suggestions and encouragement. The students of the Graphics Jungle and DGP labs, as well as the anonymous reviewers, also provided invaluable feedback.

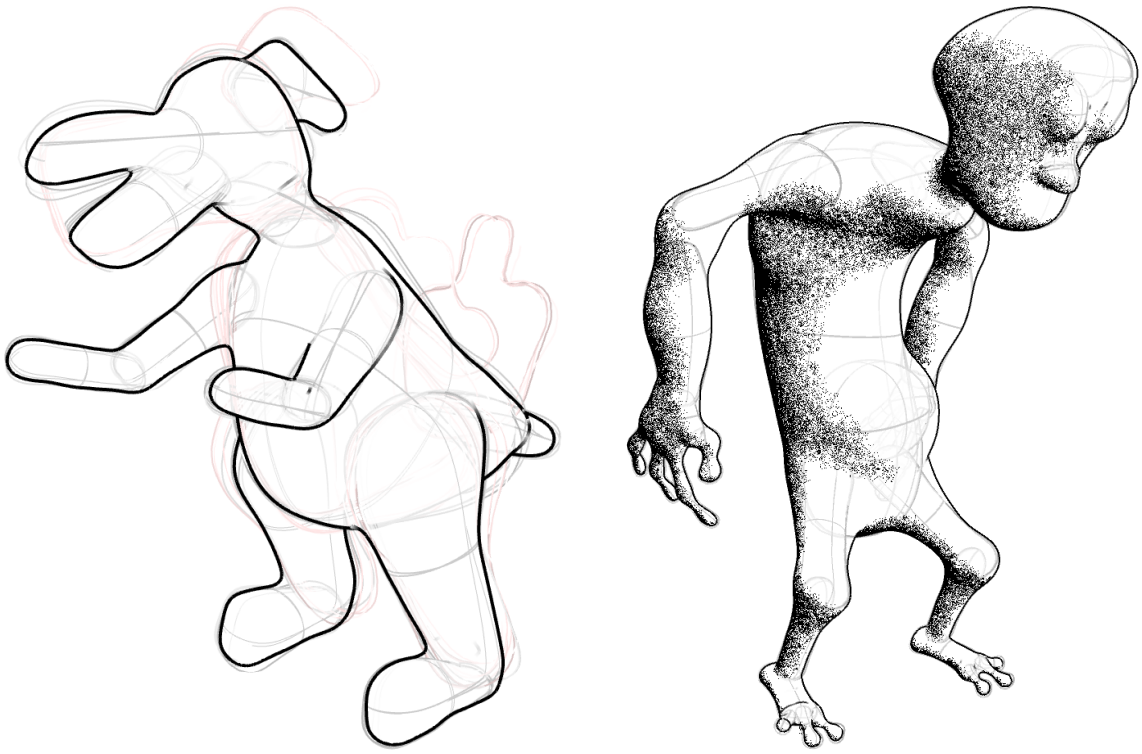


Figure 16: *Visual construction history combined with interactive pen-and-ink contours on complex models. The left figure was interactively modeled using our rendering pipeline. A variety of eraser marks show some of the ideas that the designer considered but discarded.*

References

BAUDISCH, P., TAN, D., COLLOMB, M., ROBBINS, D., HINCKLEY, K., AGRAWALA, M., ZHAO, S., AND RAMOS, G. 2006. Phosphor: Explaining Transitions in the User Interface using Afterglow Effects. In *Proc. of UIST 2006*, 169–178.

BEZERIANOS, A., DRAGICEVIC, P., AND BALAKRISHNAN, R. 2006. Mnemonic Rendering: An Image-Based Approach for Exposing Hidden Changes in Dynamic Displays. In *Proc. of UIST 2006*, 159–168.

BREMER, D., AND HUGHES, J. 1998. Rapid Approximate Silhouette Rendering of Implicit Surfaces. In *Proc. of Implicit Surfaces 1998*, 155–164.

BURNS, M., KLAWE, J., RUSINKIEWICZ, S., FINKELSTEIN, A., AND DECARLO, D. 2005. Line Drawings from Volume Data. *ACM Trans. on Graph. 24*, 3, 512–518.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive Contours for Conveying Shape. *ACM Trans. on Graph. 22*, 3, 848–855.

DECARLO, D., FINKELSTEIN, A., AND RUSINKIEWICZ, S. 2004. Interactive Rendering of Suggestive Contours with Temporal Coherence. In *Proc. of NPAR 2004*, 15–24.

DEUSSEN, O., HILLER, S., VAN OVERVELD, C., AND STROTHOTTE, T. 2000. Floating Points: A Method for Computing Stipple Drawings. *Comp. Graph. Forum 19*, 3, 40–51.

DUNBAR, D., AND HUMPHREYS, G. 2006. A Spatial Data Structure for Fast Poisson-Disk Sample Generation. *ACM Trans. on Graph. 25*, 3, 503–508.

ELBER, G. 1998. Line Art Illustrations of Parametric and Implicit Forms. *IEEE Trans. on Vis. and Comp. Graph. 4*, 1, 71–81.

FOLEY, J. D., VAN DAM, A., FEINER, S. K., AND HUGHES, J. F. 1996. *Computer Graphics: Principles and Practice*, 2nd ed. Addison-Wesley.

FOSTER, K., JEPP, P., WYVILL, B., COSTA SOUSA, M., GALBRAITH, C., AND JORGE, J. A. 2005. Pen-and-Ink for BlobTree Implicit Models. *Comp. Graph. Forum 24*, 3, 267–276.

FREUDENBERG, B., MASUCH, M., AND STROTHOTTE, T. 2001. Walk-Through Illustrations: Frame-Coherent Pen-and-Ink Style in a Game Engine. *Comp. Graph. Forum 20*, 3, 184–191.

FUNG, J., AND VERYOVKA, O. 2003. Pen-and-Ink Textures for Real-Time Rendering. In *Proc. of Graphics Interface 2003*, 131–138.

HARDOCK, G., KURTENBACH, G., AND BUXTON, W. 1993. A Marking Based Interface for Collaborative Writing. In *Proc. of UIST 1993*, 259–266.

HARRISON, B. L., KURTENBACH, G., AND VICENTE, K. J. 1995. An Experimental Evaluation of Transparent User Interface Tools and Information Content. In *Proc. of UIST 1995*, 81–90.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating Smooth Surfaces. In *Proc. of SIGGRAPH 2000*, 517–526.

HOPPE, H. 1997. View-Dependent Refinement of Progressive Meshes. In *Proc. of SIGGRAPH 1997*, 189–198.

HUGHES, J. F. 2003. Differential Geometry of Implicit Surfaces in 3-Space – A Primer. Tech. Rep. CS-03-05, Department of Computer Science, Brown University, Providence, RI, USA.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A Sketching Interface for 3D Freeform Design. In *Proc. of SIGGRAPH 1999*, 409–416.

IJIRI, T., OWADA, S., AND IGARASHI, T. 2006. Seamless Integration of Initial Sketching and Subsequent Detail Editing in Flower Modeling. *Computer Graphics Forum 25*, 3, 322–328.

ISENBERG, T., FREUDENBERG, B., HALPER, N., SCHLECHTWEG, S., AND STROTHOTTE, T. 2003. A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE Comp. Graph. and Appl. 23*, 4, 28–37.

ISENBERG, T., NEUMANN, P., CARPENDALE, S., COSTA SOUSA, M., AND JORGE, J. A. 2006. Non-Photorealistic Rendering in Context: An Observational Study. In *Proc. of NPAR 2006*, 115–126.

JEPP, P., WYVILL, B., AND SOUSA, M. C. 2006. Smarticles for Sampling and Rendering Implicit Models. In *Proc. of EGUK 2006*, 39–46.

KHUSRO, K., MUNYOFU, M., SWAIN, W., AUSMAN, B., LIN, H., AND DWYER, F. 2004. Effect of Visual Scaffolding and Animation on Students' Performance on Measures of Higher Order Learning. Tech. Rep. ED485130, Assoc. for Educational Communications and Technology.

KIRSANOV, D., SANDER, P. V., AND GORTLER, S. J. 2003. Simple Silhouettes for Complex Surfaces. In *Proc. of SGP 2003*, 102–106.

KURLANDER, D., AND FEINER, S. 1990. A Visual Language for Browsing, Undoing, and Redoing Graphical Interface Commands. In *Visual Languages and Visual Programming*, S. Chang, Ed. Plenum Press, 257–275.

LEE, S., AND BUSCEMA, J. 1984. *How to Draw Comics the Marvel Way*. Simon & Schuster, Inc., New York.

MARKOSIAN, L., KOWALSKI, M. A., TRYCHIN, S. J., BOURDEV, L. D., GOLDSTEIN, D., AND HUGHES, J. F. 1997. Real-Time Nonphotorealistic Rendering. In *Proc. of SIGGRAPH 1997*, 415–420.

MCCLOUD, S. 1994. *Understanding Comics: The Invisible Art*. HarperCollins Publishers, Inc., New York.

MEIER, B. J. 1996. Painterly Rendering for Animation. In *Proc. of SIGGRAPH 1996*, 477–484.

MEYER, M. D., GEORGEL, P., AND WHITAKER, R. T. 2005. Robust Particle Systems for Curvature Dependent Sampling of Implicit Surfaces. In *Proc. of SMI 2005*, 124–133.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface Elements as Rendering Primitives. In *Proc. of SIGGRAPH 2000*, 335–342.

PLANTINGA, S., AND VEGTER, G. 2006. Computing Contour Generators of Evolving Implicit Surfaces. *ACM Trans. on Graph. 25*, 4, 1243–1280.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-Time Hatching. In *Proc. of SIGGRAPH 2001*, 581–586.

REQUICHA, A. A. G., AND VOELCKER, H. B. 1982. Solid Modeling: A Historical Summary and Contemporary Assessment. *IEEE Comp. Graph. and Appl. 2*, 2, 9–24.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible Rendering of 3-D Shapes. *ACM SIGGRAPH Computer Graphics 24*, 3, 197–206.

SALISBURY, M. P., ANDERSON, S. E., BARZEL, R., AND SALESIN, D. H. 1994. Interactive Pen-and-Ink Illustration. In *Proc. of SIGGRAPH 1994*, 101–108.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *Proc. of SIGGRAPH 1997*, 401–406.

SCHMIDT, R., WYVILL, B., SOUSA, M. C., AND JORGE, J. 2005. ShapeShop: Sketch-Based Solid Modeling with BlobTrees. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 53–62.

SCHMIDT, R. 2006. *Interactive Modeling with Implicit Surfaces*. Master's thesis, Department of Computer Science, University of Calgary, Canada.

SCHUMANN, J., STROTHOTTE, T., RAAB, A., AND LASER, S. 1996. Assessing the Effect of Non-photorealistic Rendered Images in CAD. In *Proc. of CHI 1996*, 35–42.

SECORD, A. 2002. Weighted Voronoi Stippling. In *Proc. of NPAR 2002*, 37–44.

SIMARI, P., AND SINGH, K. 2005. Extraction and Remeshing of Ellipsoidal Representations from Mesh Data. In *Proc. of Graphics Interface 2005*, 161–168.

STAM, J. 1998. Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values. In *Proc. of SIGGRAPH 1998*, 395–404.

STROILA, M., EISEMANN, E., AND HART, J. C. 2007. Clip Art Rendering of Smooth Isosurfaces. *IEEE Transactions on Visualization and Computer Graphics*. To appear.

WINKENBACH, G. A., AND SALESIN, D. H. 1994. Computer-Generated Pen-and-Ink Illustration. In *Proc. of SIGGRAPH 1994*, 91–100.

WINKENBACH, G. A., AND SALESIN, D. H. 1996. Rendering Parametric Surfaces in Pen and Ink. In *Proc. of SIGGRAPH 1996*, 469–476.

WYVILL, B., GUY, A., AND GALIN, E. 1999. Extending the CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Comp. Graph. Forum 18*, 2, 149–158.

ZANDER, J., ISENBERG, T., SCHLECHTWEG, S., AND STROTHOTTE, T. 2004. High Quality Hatching. *Comp. Graph. Forum 23*, 3, 421–430.

A Suggestive Contours on Implicit Surfaces

The appendices of [DeCarlo et al. 2004] provide a thorough overview of the differential geometry needed to describe suggestive contours. Only the key equations are repeated here. We follow their notation, where the vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ are the perpendicular *principal curvature directions* at $\mathbf{p}$, while κ_1 and κ_2 are the associated *principal curvatures*. Then $K = \kappa_1\kappa_2$ is the *Gaussian curvature*, and $H = (\kappa_1 + \kappa_2)/2$ is the *mean curvature*. We will also require the directional derivatives of the principal curvatures along the principal directions, denoted P, Q, S, and T:

$$P = D_{\mathbf{e}_1}\kappa_1, \quad Q = D_{\mathbf{e}_2}\kappa_1, \quad S = D_{\mathbf{e}_1}\kappa_2, \quad T = D_{\mathbf{e}_2}\kappa_2 \tag{7}$$

The main value that must be computed is the radial curvature κ_r:

$$\kappa_r = \kappa_1 \cos^2\phi + \kappa_2 \sin^2\phi \tag{8}$$

where ϕ is the angle between $\mathbf{w}$ and $\mathbf{e}_1$. At points where $\kappa_r = 0$, the vector $\mathbf{w}$ is re-written as $\mathbf{w} = u\mathbf{e}_1 + v\mathbf{e}_2$ (where $u = \mathbf{w}\cdot\mathbf{e}_1$ and $v = \mathbf{w}\cdot\mathbf{e}_2$), and the directional derivative is then:

$$\frac{D_{\mathbf{w}}\kappa_r(\mathbf{w})}{\|\mathbf{w}\|} = \frac{C(\mathbf{w},\mathbf{w},\mathbf{w})}{\|\mathbf{w}\|^3} + 2K\cot\theta \tag{9}$$

where $\theta = \cos^{-1}(\mathbf{n}\cdot\mathbf{v})$. The quantity $C(\mathbf{w},\mathbf{w},\mathbf{w})$ is defined as

$$\frac{C(\mathbf{w},\mathbf{w},\mathbf{w})}{\|\mathbf{w}\|^3} = \frac{Pu^3 + 3Qu^2v + 3Suv^2 + Tv^3}{\|\mathbf{w}\|^3} \tag{10}$$

Our implementation considers implicit surfaces, hence we must express these values in terms of partial derivatives of f. The curvature values can be computed from the Hessian matrix of second partial derivatives [Hughes 2003]. Unfortunately our implicit surfaces only have analytic derivatives of first order, because we apply a hierarchical spatial caching scheme which approximates f with only C^1 continuity, and certain composition operators are also only C^1 [Schmidt 2006]. Hence, we approximate the necessary second partial derivatives by central-differencing the analytic ∇f.

The directional derivatives of principal curvatures require third partial derivatives of f, which we have found to be too inaccurate when approximated by finite-differencing. Instead, we take advantage of the fact that the "implicit surface" is simply an iso-contour of a volumetric scalar field, and the notion of "curvature" is defined at all points in this scalar field (assuming it is continuous in an ϵ-ball around $\mathbf{p}$). This allows us to directly estimate $D_{\mathbf{x}}\kappa_1$ in any direction $\mathbf{x}$ by taking finite differences of $\kappa_1(\mathbf{p})$ along the line $\mathbf{p} + t\mathbf{x}$. For example, the quantity $P = D_{\mathbf{e}_1}\kappa_1$ is estimated as:

$$D_{\mathbf{e}_1}\kappa_1(\mathbf{p}) = \frac{\kappa_1(\mathbf{p}+\delta\mathbf{e}_1) - 2\kappa_1(\mathbf{p}) + \kappa_1(\mathbf{p}-\delta\mathbf{e}_1)}{\delta^2} \tag{11}$$

In our tests, this directional finite differencing results in suggestive contours which are visually more stable and more closely resemble the suggestive contours computed from mesh-based derivatives. However, we have not performed any formal analysis of this technique.