3D Modeling with Silhouettes

Alec Rivers
MIT CSAIL

Frédo Durand
MIT CSAIL

Takeo Igarashi
The University of Tokyo

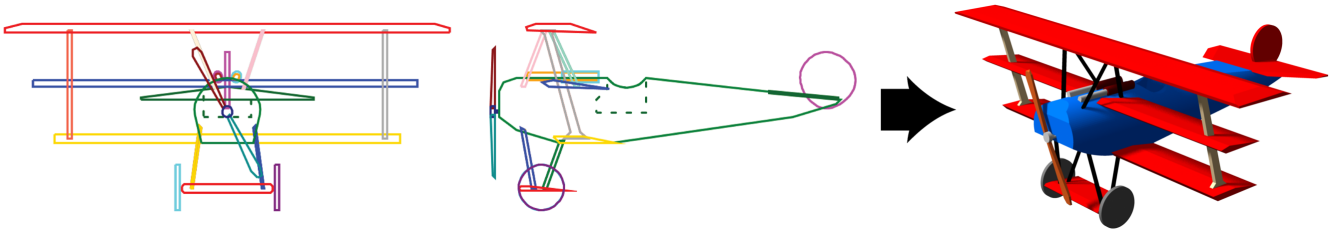


Figure 1: Intersecting 2D silhouettes: *The silhouettes on the left were used to automatically generate the 3D model on the right. Note that the line drawings are not projections of the 3D model, but rather the input that generates the model.*

Abstract

We present a new sketch-based modeling approach in which models are interactively designed by drawing their 2D silhouettes from different views. The core idea of our paper is to limit the input to 2D silhouettes, removing the need to explicitly create or position 3D elements. Arbitrarily complex models can be constructed by assembling them out of parts defined by their silhouettes, which can be combined using CSG operations. We introduce a new simplified algorithm to compute CSG solids that leverages special properties of silhouette cylinders to convert the 3D CSG problem into one that can be handled entirely with 2D operations, making implementation simpler and more robust. We evaluate our approach by modeling a random sampling of man-made objects taken from the words in WordNet, and show that all of the tested man-made objects can be modeled quickly and easily using our approach.

Keywords: 3D modeling, sketch-based modeling, silhouettes, sketching, visual hull, variational surfaces

1 Introduction

We present a new sketch-based modeling approach that facilitates the creation of 3D models of man-made objects and targets amateur and professional users. Our work is motivated by observing the workflow of traditional CAD, where design usually proceeds in two phases. A designer first sketches an object in 2D, often from front, side, and top views, and then uses these sketches as a reference in constructing a 3D model on a computer. While most of the creative designing occurs in 2D, the process of translating those 2D sketches into a 3D model is tedious, and often takes the most time. The 3D position for every element of the model must be specified, and when referring to sketches, the designer must be constantly translating a desired change in a 2D view to the sequence of 3D operations that will produce that change. We propose a new approach that can construct a 3D model directly from 2D drawings of an object from different views, eliminating the transfer step.

The core idea that makes our approach tractable, from both a user interface and implementation perspective, is to focus on 2D object silhouettes. The user specifies a part by drawing its silhouettes in two or more orthographic views, and a 3D shape that matches those silhouettes is automatically generated and displayed. A user can focus on iterating the 2D drawings, allowing for fast experimentation and modeling.

From an algorithmic standpoint, we show that using silhouettes dramatically simplifies the computation of one of the most challenging features in CAD-CAM: Boolean operations, a.k.a. Constructive Solid Geometry (CSG). Building on ideas from the visual hull and silhouette intersection, we introduce simple algorithms that operate entirely in 2D but can generate complex 3D CSG objects. We also enable the creation of smooth shapes that approximate the least-variation-of-curvature surfaces that match their input silhouettes.

In our interface, a user specifies the silhouettes of a part from front, side, or top views. Once two or more sihouettes have been specified, its 3D shape is automatically constructed. Although each part is axis-aligned, parts can be rotated in 3D relative to each other, meaning that the model as a whole need not be axis-aligned. We target the modeling of man-made objects, as they typically can be decomposed into axis-aligned subparts. Organic models are not well suited to this approach.

We illustrate the simplicity and expressiveness of silhouettes as a modeling device by modeling a small random sampling of all man-made objects indexed in WordNet [1995]. We show that all of the man-made objects in our sample can be split into a low number of subparts that are fully specified by their silhouettes from three views. We also show a detailed scene modeled entirely with silhouettes. We discuss the differences between our approach and Google SketchUp. A user study comparing the two is available in [Rivers et al. 2010]. Finally, we provide an executable of our implemented modeler, available at our project website.

2 Related Work

Similar to many CAD modeling systems, e.g. [SolidWorks 2009; Pro/ENGINEER 2009; CATIA 2009], our interface uses three orthographic 2D views and a single 3D view. However, the way these views are used is different. In our approach, these views are not 2D projections of a fundamentally 3D model, but rather canvases

on which the user is free to sketch as he or she wishes, with these sketches then determining the shape of the 3D model. The operations the user performs are therefore different: in a traditional CAD program, the user performs a wide range of modeling and editing operations that directly affect the 3D properties of the model, such as placing and connecting vertices, rotating facets, extruding faces, etc., while in our approach, the only operation the user performs is drawing 2D shapes. This greatly simplifies the interface and reduces the learning curve.

To generate the surfaces corresponding to input silhouettes, we build upon and extend the concept of the visual hull and silhouette intersection [Szeliski 1993; Laurentini 1994; Matusik et al. 2000; Franco and Boyer 2003; Lazebnik et al. 2007]. Whereas the visual hull has previously been used mainly to reconstruct 3D shapes from video or image, we introduce its use as a modeling tool. When used to reconstruct a 3D shape from the silhouettes of an entire object, the visual hull is limited in the complexity of the models it can produce, and cannot generate concavities. In our approach, however, a model is assembled out of an arbitrary number of subparts, each one of which is separately defined by its silhouettes. The overall complexity of the model is therefore unlimited (see Figure 1).

Subparts of a model can also be subtracted from each other, as in Constructive Solid Geometry (CSG) modelers, simplifying the modeling of concavities. Tilove [1980] describes a divide-and-conquer approach to computing CSG solids constructed out of simple primitives; we present a way to compute the boundary representation of a CSG solid of silhouette cylinders by applying a similar divide-and-conquer approach in 2D on just the planes on which surface facets may lie.

Our approach shares similarities with systems developed to convert drafted wireframe representations of objects to CAD models [Sakurai and Gossard 1983; Wang and Grinstein 1993; Chen and Perng 1988]. These approaches were designed to be applied to existing blueprints with no explicit correspondences between shapes in different views, and therefore focused largely on the issue of resolving ambiguity, typically with user guidance. Furthermore, they restricted input drawings to a subset of shape types, limiting the complexity of the generated model. Our approach, by comparison, is interactive, generates just one 3D model without ambiguity, can support any input shapes, and can generate smooth objects.

Related methods have been proposed that interactively infer a 3D shape from a single drawing [Lipson and Shpitalni 1996; Masry et al. 2007; Gingold et al. 2009]. They share with our approach the goal of using 2D drawings to generate a 3D model. The main difference is that they work from a single view, which results in ambiguities which must be manually resolved by the user. Using a single view also limits the complexity of shapes that can be generated.

Creating 3D models that match desired appearances from different views has recently been approached as an artistic challenge. Sela and Elber [2007] describe how a 3D model can be created that resembles two different input models from different views. Mitra and Pauly [2009] describe shadow art, a novel form of sculpture in which a 3D shape is constructed that matches a set of input silhouettes. While the main application is artistic, they also describe how mutually contradictory input silhouettes can be deformed to eliminate contradictions while retaining the originally intended appearance, which could potentially be useful in our approach to 3D modeling.

2D sketching has recently been leveraged as a powerful tool for 3D modeling [Zeleznik et al. 1996; Igarashi et al. 1999; Igarashi and Hughes 2001; Karpenko et al. 2002; Schmidt et al. 2005; Karpenko and Hughes 2006; Dorsey et al. 2007; Kara and Shimada 2007; Nealen et al. 2007; SketchUp 2009] and editing [Nealen et al. 2005]. In these approaches, the interface consists of a single 3D view which can be rotated by the user. 2D sketching operations are used to construct and modify the 3D model, which is represented in one of many possible ways, such as floating 3D curves, collections of edges and surfaces, or volumetric blobs.

The major differences between our approach and these methods are that we use fixed 2D views, as opposed to a 3D view, and that we require just 2D drawing as input. Using fixed 2D views eliminates the need to establish a 3D context for 2D sketching operation, and therefore the need for constant rotation and positioning of the scene by the user. By essentially restricting ourself to one tool, we also reduce the learning curve. Finally, unlike some of these interfaces, in our approach the sketched input (silhouettes) is persistent, and can be revisited and edited in any order.

As with Prasad et al. [2005], we wish to find a smooth shape that matches a set of input silhouettes. To do this, we build on the algorithms presented in FiberMesh [2007], which introduced a way to compute a smooth mesh that connects a set of 3D curves. We introduce a special form of FiberMesh-style smoothing that smooths the mesh while maintaining the original silhouettes. We use this method to obtain a smooth shape that matches the input silhouettes.

The fact that our system requires solely 2D input and interaction is the major difference between our approach and the majority of existing sketch-based and CAD modelers, and greatly reduces the required interface complexity. Despite the simple nature of the input, we believe that our approach can handle a wide range of models, and allows for unlimited complexity, while also eliminating the problem of ambiguity present in some of the previous approaches.

3 User Interface

In our approach, a 3D model is assembled out of parts. Each part is specified by two or three silhouettes from front, side, or top views. As the user creates and refines the silhouettes of a part, the corresponding 3D model is automatically generated and displayed. For example, if the user sketches a triangle in the front and side views, and a square in the top, a pyramid will appear in the 3D view. The calculation is fast enough to provide immediate feedback.

A shape with complex or occluded segments is built up by composing it out of multiple parts. This matches a natural 2D drawing approach of splitting objects into parts, and also sidesteps what would otherwise be a major problem of underspecification. Each part has its silhouettes drawn separately, and our interface makes associations between parts and silhouettes explicit by only allowing the user to draw silhouettes for the currently selected part. Often, a part is fully specified with only two silhouettes; in these cases, the third silhouette need not be drawn.

A part's silhouette in one view imposes certain constraints on the part's silhouettes in the other views. For example, the height of a part's silhouette in the side view should match the height of the part's silhouette in the front view. While a part is selected, these constraints are illustrated as guidelines showing the maximum and minimum extents of the silhouette in the X and Y directions. However, these constraints are not enforced during the drawing process, as they are often temporarily broken as the user is editing the model, e.g. after moving a model's silhouette in one view but before moving it in another. Because the shape generated for each part is the intersection of the silhouettes' extrusions, the shape of a part with mutually inconsistent silhouettes will be limited by the more restrictive silhouette. The guidelines serve to make it clear when this is occurring.

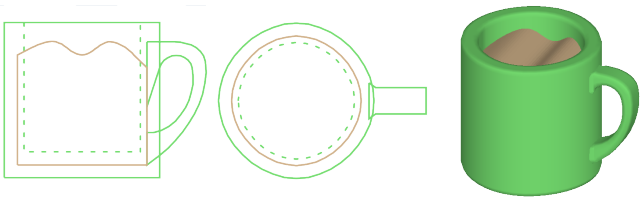


Figure 2: Drawing a subtraction: *A coffee mug is assembled by drawing the silhouettes of the hole separately (the dashed lines), and subtracting the resulting shape from the mug.*

Figure 3: *A non-axis-aligned model generated by our approach.*

Rotation: Once a part has been drawn into the model, it can be rotated in 3D relative to the rest of the model. In Figure 3, the earphones and microphone have been rotated relative to the rest of the headset. When a part is rotated relative to the current view of the editor, its silhouettes are no longer orthogonal and can therefore not be edited; in the interface, the 3D bounding box of the rotated part is shown instead. Double-clicking a rotated part causes the editor's view to switch to that part's orthographic space; the silhouettes are now orthogonal and can be edited, while the other parts of the model are no longer orthogonal and are shown as bounding boxes. In this way, complex models can be constructed with non-orthogonal parts, with the only requirement being that each subpart must by itself be modeled by locally orthogonal silhouettes. However, we have found that for simple models, non-orthographic parts are rarely needed; e.g., none of the models in Table 1 required them. Note that the limitation to three orthogonal silhouettes is not algorithmic, but is intended to ease the user's cognitive load.

3.1 Drawing Concavities

By default, if a user draws parts that intersect, they simply overlap in the 3D model. However, the user can optionally have a part act as a cut-away volume from a part. Under the hood, this is performing a Boolean subtraction [Requicha 1977]. This allows the user to easily draw in concavities, for example the hole in a coffee cup (see Figure 2). Parts that have been rotated relative to one another can still be subtracted from one another.

3.2 Smoothing

Shapes constructed purely by intersection and subtraction of orthographic silhouettes always have sharp edges. However, we allow the user to toggle for each part whether the desired shape is sharp or smooth. A smoothed part generates an approximation of the smoothest possible shape that satisfies the input silhouettes. This is illustrated in Figure 4.

Smoothing operations are performed strictly after subtractions: a smoothed object cannot be subtracted from any other, but an object that is composed out of parts and subtractions can be smoothed.

2D Smoothing Smooth surfaces by default minimize the variation of curvature in both principal curvature directions. However, the user can specify that a part should minimize variation of curvature only around one axis. The primary purpose of this is to allow the construction of solids of revolution, though it supports a slightly more general class of shapes, such as solids of revolution that have been stretched in one axis.

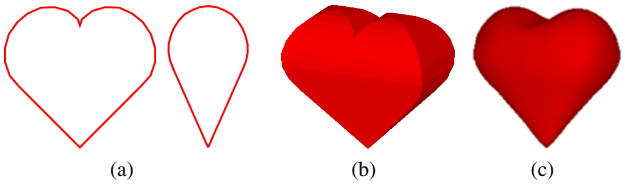


Figure 4: Smoothing: *Input silhouettes (a) are intersected to find a sharp shape (b) which is then smoothed while preserving the specified silhouettes (c).*

4 Algorithms

Models in our approach are composed out of parts defined by their silhouettes, which the user can combine using Boolean operations. A model before smoothing can therefore be represented by a CSG tree applied to silhouette cylinders. A silhouette cylinder is the orthographic equivalent of a silhouette cone: it is the infinite extrusion of a silhouette in the viewing direction. A single part's shape is simply the intersection of its silhouette cylinders; this is the same as the visual hull. Subtraction parts are cut away by intersecting the subtraction part's silhouette cylinders, and subtracting the result from the original part.

Traditionally, calculating Boolean operations on arbitrary polyhedra involves many special cases, and an error in one part of the mesh can make it impossible to generate the correct surface elsewhere. It is notoriously difficult to produce a robust implementation.

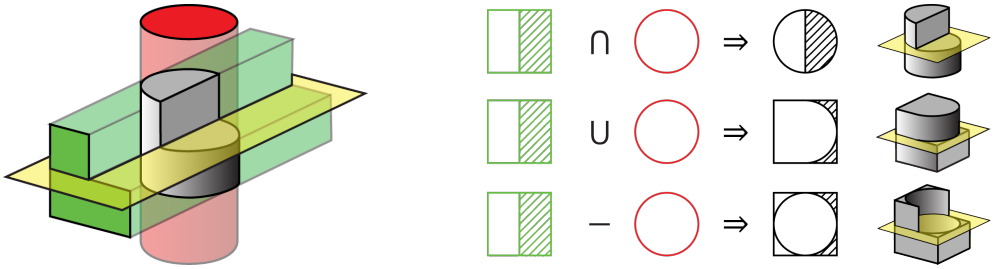
In this section, we present a simple algorithm that can calculate the solid resulting from an arbitrary CSG tree applied to silhouette cylinders. Our algorithm builds on the work of Matusik et al. [2000], who showed that the visual hull (the intersection of silhouette cylinders) could be computed using 2D operations. We show that the computation of full CSG trees over silhouettes can similarly be performed using only 2D operations. Because 2D Boolean operations suffer from far fewer special cases than 3D Boolean operations, and each facet of the resulting CSG solid is computed independently, this algorithm is easy to implement robustly.

We also introduce a simple algorithm for smoothing a shape constructed from silhouettes. This smoothing operation generates a surface that approximates the surface minimizing the variation of Laplacian magnitude, as in [Nealen et al. 2007], but that still has the original input silhouettes.

4.1 Computing CSG Solids

Our algorithm for computing a CSG solid from silhouette cylinders works by examining each potential plane on which a facet of the final 3D solid may lie. Each segment of each input silhouette determines one such plane: it is the plane on which the segment lies and which has a normal orthogonal to both the segment and the silhouette's viewing direction. (Note that we discretize all curves in the silhouettes as polylines.)

We examine each plane independently. For each plane, we wish to find the intersection of the plane with the surface of the CSG solid. This intersection determines one surface facet of the final



(a) (b)

Figure 5: Finding the visual hull: *(a) Two silhouette cylinders defined by a circle and an L shape are intersected to produce a 3D shape. The yellow plane is one plane on which a surface facet of the resulting solid may lie; it is generated by a segment of the L-shaped silhouette. (b) Calculating the surface facet on the yellow plane of various CSG solids formed by a Boolean operation applied to the input silhouette cylinders. The 2D polygons are the intersection of the plane with the solid (*$\mathbf{i}$*), with the dashed portions being just the part of the plane that is on the surface of the solid (*$\mathbf{s}$*). The solids generated by the different Boolean operations are at far right.*

polyhedron. The boundary representation of the final CSG solid is simply the collection of all surface facets generated on all the planes.

Because we perform all our calculations on one plane at a time, we can work in 2D, making calculations simpler and more robust. When the polygon that corresponds to the intersection of the CSG solid's surface with the plane has been found, we project it back into 3D to obtain a surface facet.

Our algorithm for computing the CSG solid builds on the insight that we can intersect all of the silhouette cylinders with the plane, and then perform 2D Boolean operations on the resulting 2D polygons to find the intersection of the derived CSG solid with the plane, essentially turning a 3D CSG operation on solids into a 2D CSG operation on polygons. However, some extra tricks are required to keep track not only of the intersection of the CSG solid with the plane, but the intersection of the CSG solid's *surface* with the plane, as we only wish to generate polygons along the surface of the solid.

Our algorithm for finding the surface facet corresponding to the plane proceeds as follows: first, we find the intersection of each silhouette cylinder c with the plane. We label these intersection polygons $\mathbf{i}_c$. For each silhouette, we also find the intersection of the *surface* of the silhouette cylinder with the plane (as opposed to the interior), which we label $\mathbf{s}_c$, and which is a subset of $\mathbf{i}_c$. Then, we apply 2D Boolean operations to the intersection and surface polygons corresponding to a set of silhouette cylinders to compute $\mathbf{i}$ and $\mathbf{c}$ for the solid resulting from a Boolean operation applied to the corresponding silhouette cylinders. In general, we show that Boolean operations can be computed for any two solids for which $\mathbf{i}$ and $\mathbf{s}$, the interior and surface polygons, are known for every plane, to yield the $\mathbf{i}$ and $\mathbf{s}$ of the resulting solid for each plane, allowing further operations to be applied to the derived solid.

4.1.1 Calculating the Silhouette Cylinder Intersections

Given a plane, for each silhouette cylinder c, we first find $\mathbf{i}_c$, the intersection of the silhouette cylinder and the plane. If a silhouette's viewing direction does not lie on the plane, $\mathbf{i}_c$ is simply the 2D projection of the silhouette onto the plane. If the viewing direction does lie on the plane (as with the green silhouette and the yellow plane in Figure 5), finding the intersection is slightly more involved: we project the plane into silhouette space to obtain a line, which we then intersect with the silhouette, and finally take the resulting one or more line segments into plane space and extrude them infinitely in the viewing direction to yield $\mathbf{i}_c$. In this case $\mathbf{i}_c$ will consist of one or more infinitely long rectangular polygons (which we show truncated in Figure 5 (b) for clarity).

Next, we find $\mathbf{s}_c$, the intersection of the *surface* of the silhouette cylinder with the plane. This is a simpler operation: first, we observe that only silhouette cylinders with viewing directions that lie on the plane have a non-zero-area intersection of their surface with the plane. For each such silhouette, we project the plane into silhouette space to obtain a line, as before. Then, the silhouette segments that lie entirely on this line are found, and we take these into plane space and extrude them infinitely in the viewing direction to yield $\mathbf{s}_c$. In the left two columns of Figure 5 (b), we illustrate $\mathbf{i}$ and $\mathbf{s}$ for the red and green silhouette cylinders, with the dashed portions representing $\mathbf{s}$.

4.1.2 Computing Boolean Operations

Similar to the approach given by Tilove [1980], we now show how Boolean operations on a single plane can be computed for two solids A and B for which for which $\mathbf{i}$ and $\mathbf{s}$, the interior and surface polygons, are known, to yield a solid C, defined by its own interior and surface polygons.

Intersection:

$$\mathbf{i}_C = \mathbf{i}_A \cap \mathbf{i}_B \quad (1)$$

$$\mathbf{s}_C = (\mathbf{s}_A \cup \mathbf{s}_B) \cap \mathbf{i}_C \quad (2)$$

This follows intuitively, as the interior of the intersection of two solids is the intersection of their interiors, while the surface of the intersection is any part of the surfaces of A or B that lies within C.

Union:

$$\mathbf{i}_C = \mathbf{i}_A \cup \mathbf{i}_B \quad (3)$$

$$\mathbf{s}_C = (\mathbf{s}_A - (\mathbf{i}_B - \mathbf{s}_B)) \cup (\mathbf{s}_B - (\mathbf{i}_A - \mathbf{s}_A)) \quad (4)$$

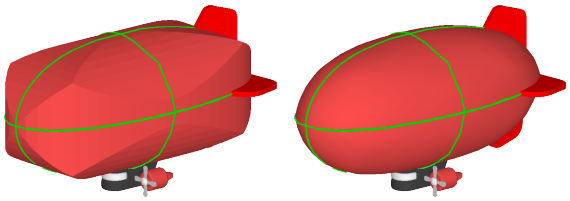
Equation 4 follows from the intuition that the surface of the solid C produced by a union of two shapes A and B is the part of A's surface that does not lie strictly within B, combined with the part of B's surface that does not lie strictly within A.

Subtraction:

$$\mathbf{i}_C = \mathbf{i}_A - (\mathbf{i}_B - \mathbf{s}_B) \quad (5)$$

$$\mathbf{s}_C = (\mathbf{s}_A - (\mathbf{i}_B - \mathbf{s}_B)) \cup (\mathbf{s}_B \cap \mathbf{i}_A) \quad (6)$$

Equation 6 follows from the intuition that the surface of a solid C produced by the subtraction of solid B from solid A is the part of A's surface that does not lie entirely within B, combined with the part of B's surface that does lie within A.



(a) (b)

Figure 6: Rim calculation: *The rims are calculated based on the unsmoothed shape (a), and then locked in place while smoothing occurs (b).*

XOR: XOR can be implemented as a combination of union and subtraction operators: $\mathbf{A} \oplus \mathbf{B} = (\mathbf{A} - \mathbf{B}) \cup (\mathbf{B} - \mathbf{A})$.

Using these equations, we apply the CSG tree to the $\mathbf{i}$ and $\mathbf{s}$ of the silhouette cylinders on the plane to yield $\mathbf{S}$, the intersection of the final CSG solid's surface with the plane, which we then project back into 3D to get a surface facet of the final solid. By combining the facets generated in this way from all planes, we assemble the final CSG solid's polyhedron. In Figure 5 (b), we illustrate the calculation of $\mathbf{I}$ and $\mathbf{S}$ on a single plane for the above Boolean operations applied to two silhouette cylinders.

If desired, surface normals can be automatically generated as part of this step. This is done by recording separately for each solid separate copies of $\mathbf{s}$ corresponding to surface facets with a positive normal relative to the plane and with a negative normal relative to the plane. For a single silhouette cylinder, these polygons can be easily separated by observing the normals of the silhouette segments that generate $\mathbf{s}_C$. For solids that are the result of Boolean operations, we carry out the above surface operations twice for each plane, once on $\{\mathbf{i}_A, \mathbf{i}_B, \mathbf{s}_A^+, \mathbf{s}_B^+\}$ to generate $\mathbf{s}_C^+$ and once on $\{\mathbf{i}_A, \mathbf{i}_B, \mathbf{s}_A^-, \mathbf{s}_B^-\}$ to generate $\mathbf{s}_C^-$. When adding facets to the mesh, we can separately project $\mathbf{S}^+$ into 3D to generate a facet with a positive normal and $\mathbf{S}^-$ into 3D to generate a facet with a negative normal. Optionally, we can subtract $(\mathbf{S}^+ \cap \mathbf{S}^-)$ from $\mathbf{S}^+$ and $\mathbf{S}^-$ to avoid generating infinitely thin strips in the final model.

4.2 Smoothing

In this section we present an algorithm for smoothing a part defined by silhouettes that preserves the original input silhouettes. We build on the smoothing algorithm of Nealen et al. [2007].

The first step in smoothing is to remesh the part's 3D polyhedron into collection of small facets, suitable for smoothing. This is done by intersecting each facet with a regular grid of squares projected flat onto the facet's surface. We take care to ensure that the generated mesh remains watertight along seams between facets.

Ensuring that a mesh maintains its silhouettes as it smooths is difficult. One way a mesh could fail to satisfy its silhouettes is if it expands beyond its silhouettes during smoothing. We preclude this possibility by only allowing each vertex to move in the direction opposite to its normal. This leaves the more tricky problem of a mesh shrinking to the point that it no longer satisfies a silhouette.

Enforcing Rims A single point on a given silhouette defines a line in 3D space that passes through that point and extends infinitely in the viewing direction of that silhouette. We refer to this as the point's rim line. A mesh has shrunk too much if there exists a silhouette point for which the rim line never touches the mesh. To ensure that a mesh continues to match its silhouette as it is smoothed, we explicitly calculate a point on each rim line and lock it in place before smoothing. We call these points the rim points, or collectively the rims, an established term for the positions at which a smooth shape touches its visual hull.

Our algorithm for calculating rim points is straightforward. Conceptually, every point along each silhouette should correspond to a rim point on the visual hull. In practice, because the mesh is composed of discrete triangles, this is not necessary. We consider only rim lines that pass through a vertex of the remeshed model. For each of these rim lines, we calculate its intersection with the unsmoothed model, resulting in a set of 3D line segments. We then place a rim point at the center of each of these segments. The mesh must then be re-tessellated to include these points. Although this rim-finding system may seem simplistic, we have found it to work well in practice (see Figure 6).

4.2.1 Minimizing Variation of Laplacian Magnitude

We take the approach to smoothing described in FiberMesh [Nealen et al. 2007] and apply it as a sequence of simple smoothing iterations. Each iteration updates the positions of the vertices as follows:

$$\mathbf{c}_i = \|\mathbf{x}_i - \frac{1}{|\mathbf{N}_i|} \sum_{j \in \mathbf{N}_i} \mathbf{x}_j\| \quad (7)$$

$$\mathbf{c}'_i = \frac{1}{|\mathbf{N}_i|} \sum_{j \in \mathbf{N}_i} \mathbf{c}_j \quad (8)$$

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{c}'_i \mathbf{n}_i \quad (9)$$

where $\mathbf{x}_i$ is the position of vertex i, $\mathbf{N}_i$ is the set of neighbor vertices connected to vertex i, and $\mathbf{c}_i$ is the estimated curvature at vertex i (simply the magnitude of the discrete graph Laplacian). Rim vertices are locked in place. This update step is applied repeatedly until the greatest change of position for a vertex in the mesh is less than some threshold. This converges to an approximation of the surface that minimizes the variation of Laplacian magnitude across the mesh.

2D smoothing as described in Section 3.2 is achieved by limiting $\mathbf{N}_i$ for each vertex to the subset of connected vertices that lie at the same depth relative to the smoothing plane.

5 Limitations and Results

In our approach, each part of a model is specified by up to three orthographic silhouettes. This puts a limit on the complexity of each individual part: specifically, an unsmoothed part cannot have facets that are not perpendicular to at least one primary axis. However, this limitation is offset by two factors: first, each part can be independently rotated once created, allowing off-axis facets in a model composed of multiple parts; and secondly, parts can be smoothed, which includes the flexibility to generate all solids of revolution, as well as a more general class of smooth shapes.

In practice, therefore, our main limitation is with solids where a majority of facets share no common axes. This property applies mainly to organic shapes. A small fraction of man-made objects, such as drill bits, also share this property. However, we note that these types of shapes are difficult to achieve in existing modelers as well, including traditional CAD systems, which typically do not support organic shapes and provide a special-purpose tool to generate helices.

For man-made objects, we believe that almost every object can be modeled using our approach. This is because man-made shapes can generally be decomposed into simple subparts, each of which has facets that share common axes (even if the axes are not shared between parts). The failure cases amongst man-made objects are

Figure 7: Complex scene: *Every model in this image was designed and composited using our implemented modeler. Rendering was performed separately in Maya.*

primarily those that have organic-like shapes (e.g., aerodynamic car hoods, sculptures, clothes), which, again, are difficult even with traditional modelers.

We performed two informal evaluations to test our approach: one to test its range (what models it is able to create) and one to test its scalability (how complex the models can be). We also describe the differences between our approach and Google SketchUp. A user study in which novice users were asked to create 3D models with our approach and with Google SketchUp, which were then ranked by a second group of users, is also available in a separate technical report [Rivers et al. 2010].

Range Evaluation: To test whether the majority of man-made objects can in fact be modeled simply using silhouettes, we modeled a random sample of man-made objects using our approach. We used WordNet to generate a list of random hyponyms of the word "artifact", in the meaning of a physical entity created by man, which we believed would fairly sample all man-made objects. We chose to generate this list automatically and randomly to prevent the possibility of "cherry-picking" models that work well with our approach. We then used Google search to select an image of the given object, selecting from the first page of either web or image search results, which we used as a reference. (The purpose of this was again to preclude the possibility of choosing a representation of the word that unfairly favors our modeler; having an image is not itself necessary for the modeling process.) We then created a 3D model of the object. We proceeded in order down the list, only skipping words that did not correspond to concrete physical objects (8 of the first 32 words generated: the rejected words were "fencing material", "weld", "photographic print", "aisle", "seidel", "dedicated file server", "knobble", and "dialect atlas").

We show a table of the results on the final page. Models were created by two individuals, neither of whom had significant previous 3D modeling experience. The average time to create a model was 21.7 minutes.

As is shown, all of the first 24 randomly generated objects could be modeled quickly and well using silhouettes, and in the majority of cases the number of subparts needed is very low. The most difficult cases were, as expected, those with organic and indivisible shapes, such as the Mao jacket and Abaya, though we believe that these too have passable results. We believe that this shows that the great majority of man-made objects are amenable to being modeled with silhouettes.

Scalability Evaluation: We also wished to test whether modeling with silhouettes put a limit on the maximum complexity achievable in a model. To test whether this was the case, we chose to model a single scene with high-fidelity using our approach. We modeled an office, which we populated with objects from our real office. We show the result in Figure 7. Every object was created and composited in our implemented modeler. We found that we could generate arbitrarily complex models, and that adding details did not become more difficult as the models became more complex.

Comparison to Google SketchUp: Google SketchUp [2009] is recent sketch-based modeler that tackles the same challenge as our approach – namely, to make it easy for novice users to design man-made objects. However, there are several differences in the way that we approach this challenge.

One key difference between our approach and SketchUp is the nature of the lowest-level building blocks of a model. In SketchUp, the most basic elements of a model are edges and surfaces, and 3D volumes exist only as boundary representations constructed out of edges and surfaces. By comparison, the lowest-level building block of a model in our approach is a 3D volume, specified by the intersection of 2D silhouettes. We believe that working with volumes and silhouettes is more intuitive from a conceptual point of view, and bypasses a variety of difficulties that novice users may have in working with a boundary representation: for example, in SketchUp, a user must manually ensure the coplanarity of lines when generating a surface, be aware of and avoid T-junctions, ensure watertightness, and so on. These concepts are particularly challenging for novice modelers.

Another difference is that SketchUp's interface presents the user with a single 3D view, and as a result all 2D sketching operations performed by the user are ambiguous with regards to depth. This can lead to mistakes where the user performs a modeling operation that appears reasonable in the current view, only to rotate and find that the actual shape was greatly distorted. In our interface, by comparison, all shapes are drawn in at least two views, preventing this type of ambiguity.

Please see [Rivers et al. 2010] for the results of a user study comparing our approach with Google SketchUp.

6 Conclusion

In this paper, we proposed the use of 2D silhouettes as a modeling tool. We showed that by limiting ourselves to 2D input, we are able to simplify the modeling interface, even relative to sketch-based modelers. We presented algorithms for robustly calculating CSG operations on shapes defined by 2D silhouettes, and for smoothing such shapes in a way that preserves the original silhouettes.

In our range evaluation, we demonstrated that despite limiting ourselves to 2D input, the great majority of man-made objects could be modeled quickly and easily with silhouettes. We also modeled a high-fidelity office scene to show that our approach puts no restrictions on the complexity of models that can be generated.

We believe that the simplicity of our approach is illustrated by the 2D silhouettes of the models in Table 1. Aside from smoothing settings, these drawings completely specify the corresponding 3D models. Therefore, any user who can draw those 2D silhouettes

can create those 3D models. We encourage the reader to try our implemented modeler, available on our project page.

7 Acknowledgments

Thanks to Danielle Magrogan for providing many of the models in the range evaluation, and the reviewers of the MIT pre-deadline. Ilya Baran provided helpful discussions and suggestions on the structure of the paper. This work was supported by funding from the MathWorks Fellowship, the Singapore-MIT Gambit Game Lab, and Intel.

References

CATIA. 2009. Dassault Systemes.

CHEN, Z., AND PERNG, D.-B. 1988. Automatic reconstruction of 3D solid objects from 2D orthographic views. *Pattern Recogn. 21*, 439–449.

DORSEY, J., XU, S., SMEDRESMAN, G., RUSHMEIER, H., AND MCMILLAN, L. 2007. The Mental Canvas:A Tool for Conceptual Architectural Design and Analysis. *PG*.

FRANCO, J.-S., AND BOYER, E. 2003. Exact Polyhedral Visual Hulls. In *Proceedings of the Fourteenth British Machine Vision Conference*, 329–338.

GINGOLD, Y., IGARASHI, T., AND ZORIN, D. 2009. Structured annotations for 2D-to-3D modeling. *ACM Transactions on Graphics (TOG) 28*, 5, –18.

IGARASHI, T., AND HUGHES, J. F. 2001. A suggestive interface for 3D drawing. *Symposium on User Interface Software and Technology*.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A Sketching Interface for 3D Freeform Design. *International Conference on Computer Graphics and Interactive Techniques*.

JOSHI, P., AND CARR, N. 2008. Repousse: Automatic Inflation of 2D Artwork. *Eurographics Workshop on Sketch-Based Modeling*.

JUDD, T., DURAND, F., AND ADELSON, E. 2007. Apparent ridges for line drawing. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, 19.

KARA, L. B., AND SHIMADA, K. 2007. Sketch-Based 3D-Shape Creation for Industrial Styling Design. *IEEE Comput. Graph. Appl. 27*, 60–71.

KARPENKO, O. A., AND HUGHES, J. F. 2006. SmoothSketch: 3D free-form shapes from complex sketches. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM, New York, NY, USA, 589–598.

KARPENKO, O., HUGHES, J. F., AND RASKAR, R. 2002. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum 21*, 585–594.

LAURENTINI, A. 1994. The Visual Hull Concept for Silhouette-Based Image Understanding. *IEEE Trans. Pattern Anal. Mach. Intell. 16*, 150–162.

LAZEBNIK, S., FURUKAWA, Y., AND PONCE, J. 2007. Projective Visual Hulls. *Int. J. Comput. Vision 74*, 137–165.

LIPSON, H., AND SHPITALNI, M. 1996. Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Journal of Computer Aided Design 28*, 8, 651—-663.

MASRY, M., KANG, D., AND LIPSON, H. 2007. A freehand sketching interface for progressive construction of 3D objects. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, ACM, New York, NY, USA, 30.

MATUSIK, W., BUEHLER, C., RASKAR, R., GORTLER, S. J., AND MCMILLAN, L. 2000. Image-based visual hulls. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 369–374.

MILLER, G. A. 1995. WordNet: A Lexical Database for English. *Communications of the ACM 38*, 11, 39–41.

MITRA, N. J., AND PAULY, M. 2009. Shadow art. *ACM Transactions on Graphics (TOG) 28*, 5.

NEALEN, A., SORKINE, O., ALEXA, M., AND COHEN-OR, D. 2005. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph. 24*, 1142–1147.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. FiberMesh:designing freeform surfaces with 3D curves. *ACM Transactions on Graphics (TOG) 26*, 3.

PRASAD, M., ZISSERMAN, A., AND FITZGIBBON, A. W. 2005. Fast and Controllable 3D Modelling from Silhouettes. *Proceedings of the 26th Annual Conference of the European Association for Graphics (Eurographics)*, 9—-12.

PRO/ENGINEER. 2009. Parametri Technology Corporation.

REQUICHA, A. A. G. 1977. *Mathematical Models of Rigid Solid Objects*. Production Automation Project.

RIVERS, A., DURAND, F., AND IGARASHI, T. 2010. A User Study Comparing 3D Modeling with Silhouettes and Google SketchUp. *MIT Technical Report*.

SAKURAI, H., AND GOSSARD, D. C. 1983. Solid model input through orthographic views. *SIGGRAPH Comput. Graph. 17*, 243–252.

SCHMIDT, R., WYVILL, B., SOUSA, M. C., AND JORGE, J. A., 2005. ShapeShop: Sketch-Based Solid Modeling with BlobTrees.

SELA, G., AND ELBER, G. 2007. Generation of view dependent models using free form deformation. *The Visual Computer 23*, 3 (January), 219–229.

SKETCHUP. 2009. Google.

SOLIDWORKS. 2009. Dassualt Systemes.

SZELISKI, R. 1993. Rapid octree construction from image sequences. *CVGIP: Image Underst. 58*, 23–32.

TILOVE, R. B. 1980. Set Membership Classification:A Unified Approach to Geometric Intersection Problems. *IEEE Transactions on Computers 29*, 10.

WANG, W., AND GRINSTEIN, G. G. 1993. A Survey of 3D Solid Reconstruction from 2D Projection Line Drawings. *Comput. Graph. Forum 12*, 137–158.

ZELEZNIK, R. C., HERNDON, K. P., AND HUGHES, J. F. 1996. SKETCH: An Interface for Sketching 3D Scenes. *International Conference on Computer Graphics and Interactive Techniques*.

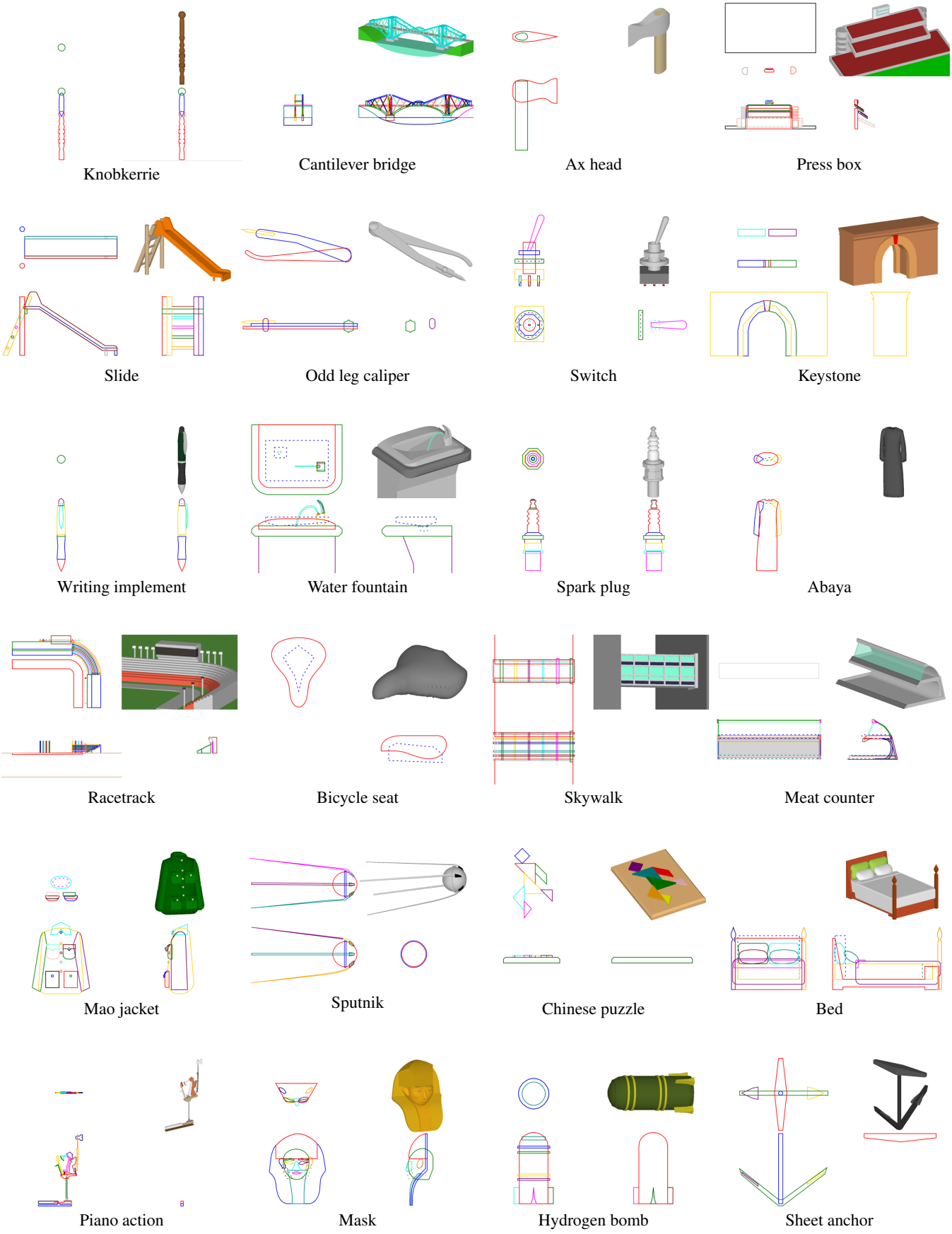


Table 1: Evaluation: *A random sampling of man-made objects modeled using our approach, shown with the silhouettes that generate them. Dashed silhouettes represent subtracted shapes. The average time to create a model was 21.7 minutes.*