3D Model Creation by Sketching Cross-sections

David Hughes[*]
Yale University
Department of Computer Science

Abstract

The question of how to interactively create and edit 3D meshes using a 2D interface is one that has proven difficult to answer. The limitations of the interface require software designers to infer 3D actions from 2D gestures, often in ways that may be unintuitive to users. Similarly, the problem of inferring 3D information from sketched 2D strokes is made difficult by the need to extrapolate depth information in a manner consistent with the user's design intentions. This paper presents a method of creating 3D meshes by connecting a series of cross-sections projected onto a moveable drawing plane, with the goal of simplifying the mesh-creation process to the point where complex objects and surfaces can be created with just a few user strokes.

Keywords: sketching, sketch-based interfaces, mesh creation, mesh editing

1 Introduction

Although the advent of interest in computer-based sketching systems has been fairly recent, sketching designs with a pencil and paper has been an important way of communicating and improving ideas for centuries. The simplicity and freedom of freehand sketching are what make it most compelling as a tool. A sketch provides instant visual feedback to a designer, initiating a feedback loop consisting of a series of refinements that culminates in either a finished design or the input to a later design phase. The system presented in this paper is a framework for making 3D objects by connecting and interpolating cross-sections along an arbitrary 3D curve. While many sketching systems take a 2D sketch as an input and impose internal constraints to determine depth information, this system's constraints are introduced to the user as tools to work with. Because of this, users can interpret their sketches literally as cross-sections of a surface without having to worry about the results of a 3D translation algorithm.

In architectural design, the visual feedback of sketches and generated 3D models is important to have when deciding if and how to modify a design and when presenting a preliminary design to a client. This feedback allows both the architect and client to more fully understand where the design currently is and encourages collaborative speculation about where it is going. Trimwork and detailing are often specified along the curves of doorways, walls, and windows, with fairly to completely uniform cross-sections, making them easy to specify with a small amount of information. Adding such details goes a long way in the visual representation of a design.

Beyond architectural uses, design by cross-section is useful when there is a need to rapidly design complex objects and surfaces. It can take a very long time to specify these objects with triangles and quads, or even with spheres, cylinders, cubes, and other low-level 3D primitives. Even techniques like lathing (extracting an object of rotation from a curve) and extrusion can be difficult tools to make some objects or surfaces with. By interpolating and connecting a series of freehand strokes along a curve, complex objects can be made easily, limited only by the ability of the user to accurately specify object cross-sections with their input strokes. Another advantage of the cross-sectional paradigm is that many different 2D operations can be applied to the curve primitives. Thus, the interpolation scheme can be extended to provide different looks depending on the interpolation function and its parameters. The details of interpolation are described in Section 4.

[*]e-mail:david.s.hughes@yale.edu

2 Previous Work

There have been a number of projects and papers in the sketch-based design that have influenced the work in this paper. Teddy [Igarashi et al. 1999] allows users to sketch the outline of an object and inflates the object into 3D based on a chordal axis. Teddy also supports simple extrusion of a single shape to fill a region specified by another user gesture. Perhaps most importantly, Teddy allows users to create models that have a hand-sketched freeform appearance, a feature this sketching system seeks to emulate. Unfortunately, it is difficult to instantiate and hone an idea for a 3D object in the system, and the model editing capabilities are not very extensive. The inflation algorithm that allows a model to be created in a few seconds also leaves you with a "what you get is what you get" model without a lot of recourse to change it.

SKETCH[Zeleznik et al. 1996] is an interesting system for creating 3D scenes using a number of simple mouse gestures and a small number of primitives that can be combined with CSG. However, the utility of SKETCH is limited by the number of primitives and operations that can be performed on them. SKETCH is best viewed as a means to quickly create scenes with simple primitives, not as a full sketching system.

[Lipson and Shpitalni 1997] takes a series of user-specified lines and interprets them according to statistically obtained features and a coordinate system that is inferred from the directions of lines and the angles between intersecting lines. While the extent to which [Lipson and Shpitalni 1997] can interpret a 2D sketch as a 3D model is impressive, the system is still hindered by an implicit assumption that all lines drawn will be straight and all faces will be planar. Using this assumption, the system doesn't work well with sketches of objects or scenes containing curvy lines or non-planar surfaces.

[Wang et al. 2003] permits sketching using a restricted domain to design garments on a variety of human forms and sizes. The application presented interpolates between cross-sectional curves and between silhouette curves, forming a 3D model by constraining the created mesh to an existing body mesh. [Wang et al. 2003] was an influential paper in developing the ideas in this paper.

[SketchUp] is a commercial sketching system that allows users to form simple models from planes and straight lines. It also has a freehand sketching mode, the use of which is almost discouraged by the system's general plane-based framework. Like [Lipson and Shpitalni 1997], it is a useful tool that is hindered by an assumption that every object created will be made up of straight lines and planes.

[Karpenko et al. 2004] presents a way to specify the 3D positions of curves by specifying their positions in 2D from multiple perspectives. The 2D strokes are projected onto epipolar 3D lines, and the curve's position is updated according to the new positions along those lines. I considered this approach to sketching lines, particularly for sketching the generalized extrusion curves needed in the current system, but ultimately have not included it. A major weakness of epipolar curve specification is that it deals poorly with non-functional curves at certain viewing angles.

[Shesh and Chen 2004] presents a sketching system that can be used to build and manipulate simple planar objects. It allows gesture-based extrusion and also has a gesture for constraining or joining one face to another. Additionally, it has a method of dealing with oversketching that allows curves to be extended, joined, or highlighted depending on the position of the oversketched stroke. Unfortunately, this is another system that assumes a planar, straight-line view of objects that will be created, and doesn't work with arbitrary curves as primitives.

3 System Components

The cross-section system currently uses the following major components:

- Positions
- Spines
- Drawing Planes
- Freehand Curves
- Frames
- Objects
- Tracing and History

The following subsections will explain the fundamentals of each of the objects listed above.

3.1 Positions

A position is a simple structure that holds a 3D vertex and three orthogonal vectors describing forward, side, and up directions. The position structure is used to orient components of the system. It is easy to interpolate between positions using either the vertex position alone, the angles alone, or a combination of both.

3.2 Spines

A spine is the most basic component of the system, and its operations act as a foundation for the other components in the system. Spines are simple vectors of points that are drawn as the line segments between those points. Indeed, the spine can be queried to get information about any point along itself, including the Position at every point.

To construct the basis for semi-smooth interpolation, the Position of each line segment along the curve is calculated. The Positions of adjacent line segments are then averaged to calculate per-vertex Positions (and the end-point vertices are assigned the Positions of the segments at their respective ends). Then any point's Position can be interpolated between its adjacent vertices.

With a semi-smooth interpolation basis in place, a spine is useful for animating objects moving along it, or even for providing virtual camera views. All that is necessary is to query the spine for the Position of a point along it (specified by absolute length) and to use that information to orient the object in question.

3.3 Drawing Planes

The drawing plane in the current implementation is a grid that moves along the currently selected Spine. It has a certain position along the Spine that can be controlled by the mouse wheel, and it uses the Position querying described above to orient itself. The actual drawing plane is a full plane, extending beyond the edges of the grid that represents it.

3.4 Freehand Curves

Freehand curves are perhaps the most important components of the system, as they specify the actual cross-sections that will become the objects the user is designing. These curves, like Spines, comprise vectors of points that are drawn as a segmented curve. However, these curves extend the features of simple curves because of how they are used in the system. For example, they need to be represented in both 2D and 3D for purposes of interpolation and connection described in Section 4, and need to be able to convert easily between the two representations.

Freehand curves are taken as input from the Draw Cam view, which projects points onto the Drawing Plane in its current position. These points are then converted to 2D by finding their distances from the Up and Left vectors of the Drawing Plane's Position and doing an angle test to fully determine the correct x-y point. Likewise, conversion from 3D to 2D involves multiplying the x-y point coordinates by the Up and Left vectors of each curve's Position along the spine and summing the results. The ability to convert to and store points in 2D makes the freehand curve component much more portable, since all it needs is a Position structure to align it along any other spine or at any point along the same spine.

An important aspect of the freehand curve is that it allows the user complete freedom to express any cross-sectional design. It doesn't force the use of straight lines, and it isn't changed or interpreted to fit a planar context (although it is constrained to the drawing plane it was projected onto).

3.5 Frames

A frame is a wrapper that contains a point along a Spine and one or more freehand curves. The current implementation only allows one curve per frame, but ultimately many curves should be able to share a frame. Frames are stored in a linked-list structure corresponding to a Spine, and are sorted on the basis of their position along it. Such a FrameList is used to specify the order of connection and interpolation between curves.

3.6 Objects

The ultimate goal in this system is to create objects. These are collections of primitives that form surfaces between cross-sectional curves. The Global Object is used to create all other objects, which can be stored in a hierarchy. The hierarchical structure is conducive to selection rendering in OpenGL, which the system is currently

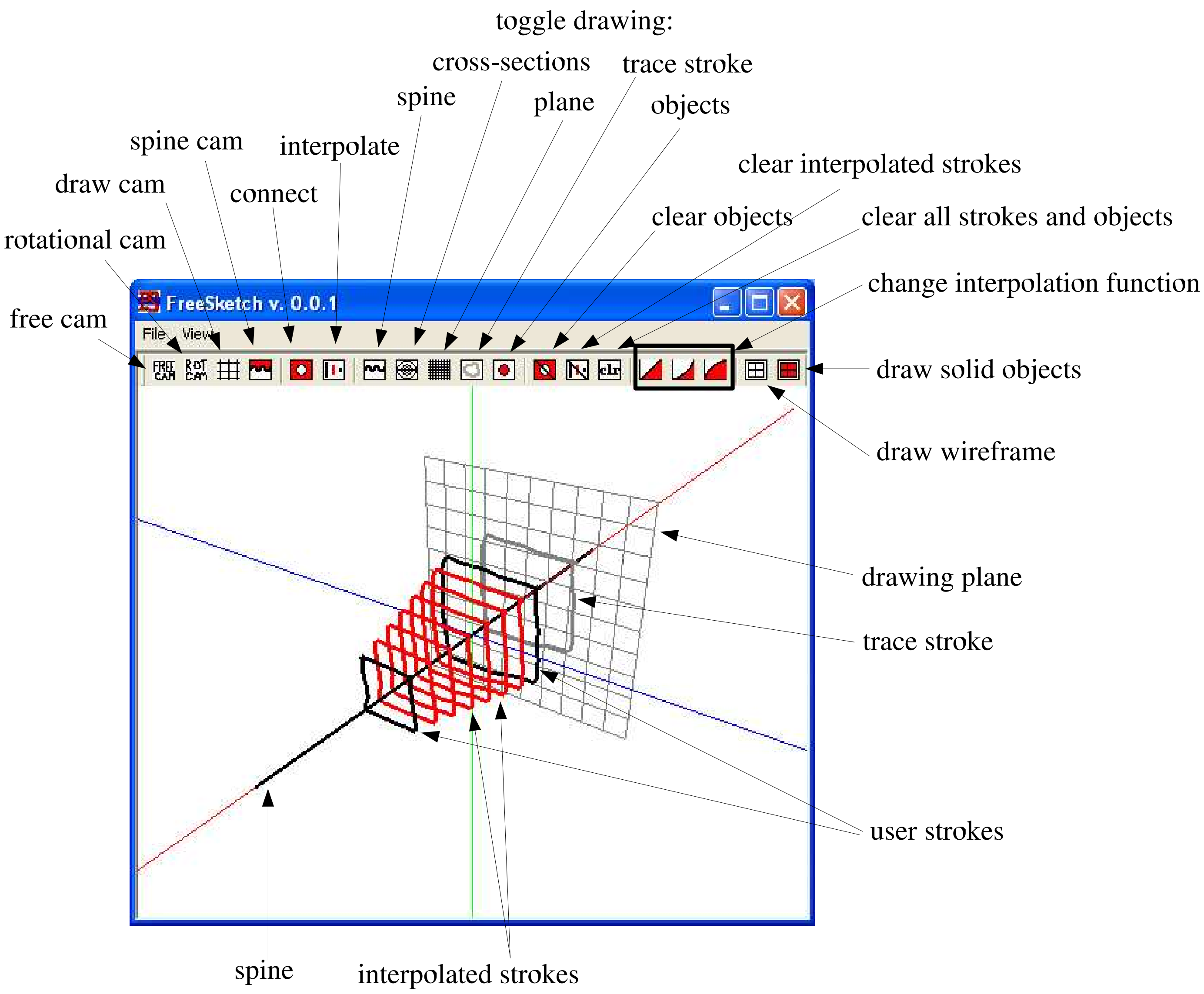


Figure 1: Labeled screenshot of all buttons and major system components

implemented in. The goal is not only to allow selection of objects and components, but to eventually allow strokes to be projected onto the geometry as well.

The main primitives of the objects are triangles and quads, and triangle strips have also been recently added.

Besides having ways of connecting cross-sections to form objects, the Global Object can also use a single curve to create an object that has been “lathed” around the y-axis. Although this isn’t integrated into the current user-interface, it would be almost trivial to add lathing functionality.

3.7 Tracing and History

The tracing and history system allows users to keep track of how their current strokes compare with past strokes. The trace stroke is automatically copied from the last stroke drawn, and can be a helpful basis for the shape of the next stroke. Additionally, the history capabilities allow a user to set their trace stroke to any past stroke by searching through the current frame list. The up and down arrow keys search backwards and forwards respectively, and an exact copy of the current trace stroke can be left in the drawing plane’s current position by pressing the space bar. Beyond giving the user a context for the next stroke, history and tracing make it easy to repeat patterns in a design, yielding results similar to an iterative approach, but with greater control and interactivity.

4 Connecting Cross-sections

The methods by which a series of cross-sectional sketches are sewn together to form a 3D object are the most important part of the system’s algorithms.

4.1 Sampling

The first step in connecting or interpolating is to sample the curves in question. The curves need to be sampled in order to normalize the number of points that we connect, so that each curve has a standard number and sections fit together properly. Currently, the number of samples is determined by the average number of points in all curves along the current Spine. A new list of points is generated by sampling at regular intervals along the length of a freehand curve. These points are then connected between adjacent curves and used to generate a uniform object mesh. However, it isn’t completely straightforward to find an appropriate sampling of points from two curves, since certain orderings of sampled points may lead the system to connect in boneheaded ways. The easiest example of this is connecting two curves that are drawn in opposite directions. The easiest sampling method leaves us with interconnections that intersect, which is probably not what the user intended. These simple samplings often result in objects that intersect themselves frequently. Since sampling alone is not enough to ensure that the object turns out well, we can use a minimization technique to find the best sampling order.

4.1.1 Smarter Sampling

To get smarter samplings as the inputs to later interpolations and connection stages, an initial sampling is taken and then the order of points is cycled. The algorithm chooses the best ordering (the ordering that results in the lowest sum of distances between

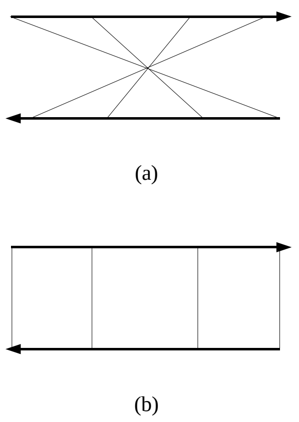


Figure 2: Illustration of problems with simple curve sampling: Both figures depict a scenario where two curves that were drawn in opposite directions are connected. Figure 2(a) shows the incorrect parameterization given by the simplest algorithm, where the direction and proximity of corresponding points on both curves isn’t accounted for. Figure 2(b) shows the connection parameterization that was probably intended, which results from the more complex algorithm described in Section 4.1.1.

corresponding points) and returns two “smart-sampled” curves. The algorithm works as follows:

```
C1, C2 are arrays of size $numsamples$ containing the original “dumb” sampling
$minDistance \leftarrow \infty$
$reverseOrder \leftarrow$ false
for $i = 0$ to $numSamples$ do
    $currentDistance \leftarrow 0$
    for $j = 0$ to $numSamples$ do
        $D \leftarrow$ distance between C1[$(i + j)$ mod $numSamples$] and C2[$j$]
        $minDistance \leftarrow minDistance + D$
        if $currentDistance < minDistance$ then
            $minDistance \leftarrow currentDistance$
            $startingPosition \leftarrow j$
            $reverseOrder \leftarrow$ false
        end if
    end for
    for $j = 0$ to numSamples do
        $D \leftarrow$ distance between points C1[$(i - j)$ mod numSamples] and C2[$j$]
        $minDistance \leftarrow minDistance + D$
        if $currentDistance < minDistance$ then
            $minDistance \leftarrow currentDistance$
            $startingPosition \leftarrow j$
            $reverseOrder \leftarrow$ true
        end if
    end for
end for
```

The distances are also calculated in reverse order, which is particularly helpful for determining the correct orientation of curves that were drawn in opposite directions. When the minimum distance is calculated and we know the starting position for the ordering, we can obtain two optimal curves to connect or interpolate:

```
Curve1, Curve2 are new curves to be synthesized from the information above
if $reverseOrder$ then
    for $i = 0$ to $numSamples$ do
        Curve1 ← C1[($startingPosition - i$) mod $numSamples$]
5:      Curve2[$i$] ← C2[$i$]
    end for
else
    for $i = 0$ to $numSamples$ do
        Curve1 ← C1[($startingPosition + i$) mod $numSamples$]
10:     Curve2[$i$] ← C2[$i$]
    end for
end if
```

It is important to note that this sampling is done in both 2D for interpolation and in 3D for connecting freehand curves. Now we have two curves that can be used to build objects (Curve1 and Curve2 from above are returned as results).

4.2 Interpolation

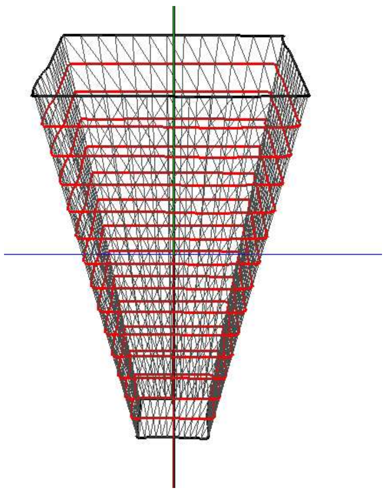
Interpolation is important because it allows a user to draw a minimal number of cross-sections and use those sketches to create more curves automatically. Interpolation takes place between 2D shapes and between points on the current spine, with a certain density. New frames containing the interpolated strokes are inserted into the current frame list at regular intervals along the spine.

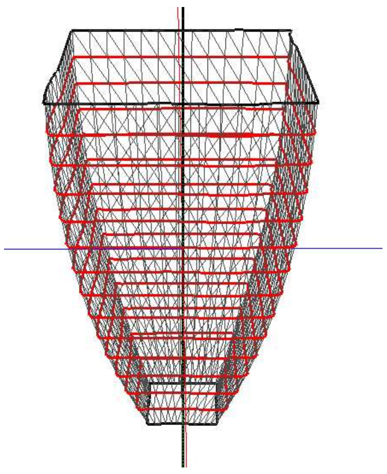
The 2D shapes are sampled using the method in Section 4.1.1 and new curves are interpolated based on a parameterization of the 2D difference vector. For example, to insert a stroke in a frame halfway between two frames with the simplified 2D “curves” (0, 0) and (0, 2), the algorithm would make a frame at half the distance between their parameters on the spine and insert the “curve” (0, 1) into it. Linear interpolation between curves is the default in the current system, but other interpolation functions are supported. These take the distance between interpolated frames, scale it to the domain [0..1], and return the value of a function on this domain. This returned value is then multiplied by the 2D difference vector to yield the final interpolated point.

The interpolation function extensions allow more interesting forms to be created than would be created by strictly linear interpolation. This is especially good for creating objects that seem cartoonish or otherworldly. The important thing about these functions is that they be of the form $f(x)$ where f is on the domain [0..1], $f(0) = 0$, and $f(1) = 1$. The latter two rules ensure that the interpolation maintains the form specified by the user’s strokes. The range of interpolation functions can even exceed the [0..1] range to yield objects that are bowed out between connected strokes. Figure 3 shows the results of three simple functions. In all figures, and in the current implementation, user-strokes are represented by thick black lines and interpolated strokes are thick red lines.

4.3 Connection

When the connection button is pressed, the system makes connections between each pair of adjacent frames in the frame list corresponding to the selected curve, forming the faces that will make up the resulting object. Connection takes a “smart” sampling and simply iterates through the points, either adding them to a new object as a triangle strip structure or as individual triangles and quads.


(a) Linear shape interpolation


(b) Square shape interpolation

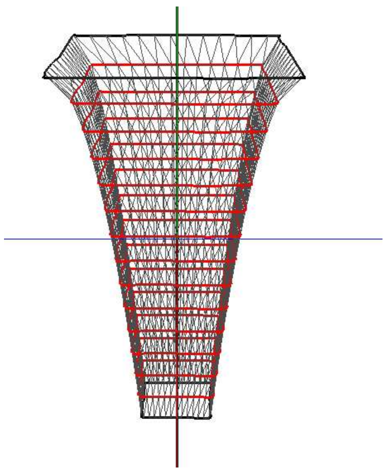

(c) Square-root shape interpolation

Figure 3: Interpolating between shapes

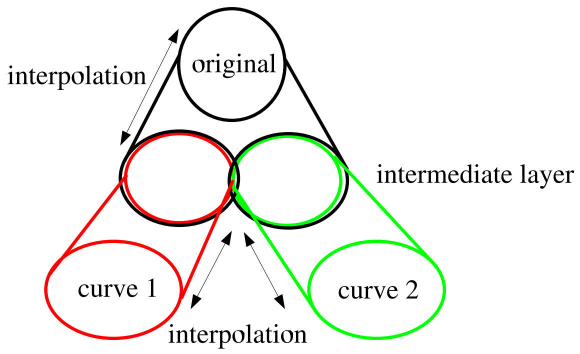


Figure 4: Illustration of algorithm to connect between frames with multiple curves. The original curve connects with a curve that traverses the boundaries of both individually interpolated curves. Then the individual curves are used for interpolating and connecting beyond the point of tangent intersection.

4.3.1 Multi-connection

One interesting and helpful feature that is not included in the current implementation is the ability to connect between frames with different numbers of curves. For example, a tree may branch off from its trunk into two main branches, each of which would be specified as a separate stroke in the same frame. In another case, a single gentle hill may divide into two maxima at its peak, each of which will be represented by separate strokes.

One solution to this problem is to initially interpolate between individual curves and then to find the point where the individual interpolations intersect tangent to one another. At this point, we form (for example)three curves, one that traverses the boundary of the intersecting areas and two others that are the individual interpolations. We can interpolate between our original single curve and the boundary-traversing curve, then interpolate individually between the other curves. This is a simple solution to the problem and makes use of existing system capabilities. An example is shown in Figure 4.

5 Results

The results obtained from using the system have been encouraging, even with the relatively low level of sophistication in the current system's drawing toolbox. The limitation to freehand strokes isn't necessary, but there is currently no way to draw perfectly straight lines or shapes like ellipses and rectangles that are popular in traditional drawing programs. Despite shortcomings in the currently implemented feature set, the geometry that can be created is still somewhat impressive, and serves as a proof-of-concept to justify further work.

5.1 Simple Terrain

Figure 5 shows two views of a volcanic mountain created in the system along a vertical spine.

Figure 6 depicts a simple terrain that was connected along a horizontal spine. It can be difficult to design a complex terrain like this one in the current system because of how prior strokes are overlaid in the view, something that is discussed in Section 6. However, a complex terrain can usually be specified easily in a few strokes.

5.2 A Castle Wall

The castle wall depicted in Figure 7 is a reasonably complex geometric model that was build using a total of four different strokes, which were repeated along a straight horizontal spine. Despite the short amount of time that was spent on the model, it looks reasonably good and has a bit of a hand-sketched look to it. The rendering not only provides good visual feedback to the designer, it does so in 3D, so that every aspect of the wall and gate can be seen and reflected on. If the wall is too narrow, the gate too wide, or the supports too tall or short, the designer can edit the component curves to change shapes and dimensions easily. Also, portions of the wall can be changed to make the entire design look less uniform and more individualized at every point along it. Alternatively, a long stretch of wall could be specified by a spine stroke and a set of rules for placing the cross-sectional strokes along it.

5.3 Elliptical Extrusion

Figure 8 shows the extrusion of a simple shape, which is pointed on the top and rounded on the bottom, along an elliptical spine. Although most of the examples shown were generated using a straight vertical or horizontal spine, any curve can be used as a spine to extrude and connect cross-sections along.

6 Discussion

This section will be divided into a discussion of extensions to the current system and a discussion of the general usefulness of the techniques presented in this paper.

6.1 Extensions and Future Work

Although there are a fair number of features to the current system implementation, the objects that can be made are still relatively simple. A full implementation of a design-by-cross-section system would include more drawing tools, allowing users to design cross-sections with the full set of tools that are included in traditional bitmap and vector-based graphical editors, along with a number of other tools appropriate for the domain. These include:

- Drawing tools - easy rectangle, ellipse, and line generation, along with the ability to keep a library of standard strokes that can be selected in an alternate "history" list.
- Curve editing tools - to exploit the potential of this system, users should be able to freely join strokes and erase and edit sections. The current implementation only allows single, continuous freehand strokes. Also, 2D selection and editing would allow a lesser number of strokes to go further in a design, allowing rotations, translations, and easy scaling.
- Object editing tools - in order to create complex scenes, there need to be controls for placing objects, along with scaling and rotating them. Constructive solid geometry and cutting techniques like those used in [Igarashi et al. 1999] and [Shesh and Chen 2004] would also be useful additions.

- Automatic spine generation - the ease of using this spine-based sketching system is fully dependent on the quality of the spines. There are a number of ways that novel spines can be inferred from existing geometry; a spine can be generated by connecting the 2D centroids of adjacent frames, by tracing an edge or intersection, or by projecting a user-stroke onto existing geometry.

- Automatic curve specification - since the cross-section strokes are specified in 2D, a number of different methods may be used to automatically generate them. For example, a 1D noise generator could be parameterized by a length along a spine and an x-value to produce a height field suitable for creating random terrains along the spine. Similarly, a circle primitive could be purturbed by a noise function to create an extruded object with a random appearance. Finally, users should be able to input a pattern that is iterated over the length of the curve to form a repeating design.

An interesting property of the system is that the list of cross-sections can potentially be transferred from one spine to another. This could lead to interesting applications in animation, where the geometry of a character's body is specified by cross-sections and re-rendered at each frame attached to a moving skeleton of spines. Another application would be in warping existing objects, bending a pillar created on a straight vertical spine by transferring it's list of cross-sections onto a curved spine.

Beyond the features mentioned above, support for smooth shading needs to be added to give created objects a more realistic appearance. Smooth shading is absent from the system at this point because it was difficult to compute vertex normals with the current object representation. The representation is not a connected mesh, but rather a series of separate triangles, quads, and triangle-strips hierarchically organized within a series of objects. It would be helpful in the future to convert this representation to one more conducive to most mesh operations.

6.2 Evaluating Design-by-Cross-section

A recurring question in my mind during this research has been that of whether or not this sort of design is intuitive and easier than creating models by other means. While I'm not convinced of this necessarily, Section 5 and the accompanying figures show a number of examples made in the system that would be difficult or impossible to make with other sketching systems. The goal of sketching with the current implementation of the system is not to construct a full 3D model from a single 2D sketch, as was the goal in many previous projects. It is to rapidly build a prototype of an object, something it accomplishes quite well. However, there are a few major hurdles that this paradigm will have to overcome.

The first difficulty involves the presentation of existing curves through the user interface. The drawing view is good for drawing, but it does not have good indicators of distance between points on the curve, especially if the curve happens to be more complicated than a straight line. An alternative to this is to draw cross-sections in the drawing view and place them along a spine using the rotational camera or some other camera that expresses a curve's spatial context well. The tracing/history functionality provides a platform for creating objects this way, but it is also clunky in that it often requires one hand to switch between controls (the arrow keys and the mouse/space bar).

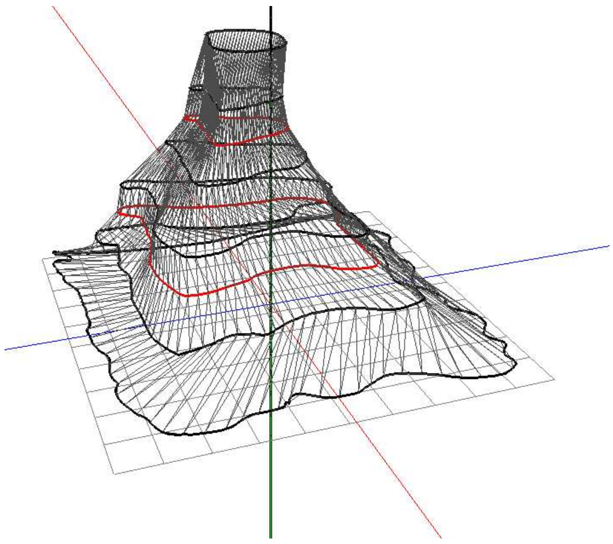
A second difficulty with the user interface is the clutter caused by rendering curves behind and in front of the current drawing plane. While this can be helpful for context, it is ultimately distracting and in some cases can be a huge hindrance. The solution to this problem may lie in rendering the drawing plane with slight alpha blending, giving the appearance of transparency and making the other curves less prominent. Changing the color or darkness of the curves may also be helpful in clearing up the confusion that this problem raises.

The more dangerous difficulty as I see it is whether or not the cross-sectional paradigm connects well with how designers work, and whether it is really conducive to design thinking. There are a lot of advantages from a programming perspective to the way the system's components work, but this may be all for nothing if it is not ultimately useful as a design tool. Personally, I have decided that the cross-sectional method is a sort of middle ground between a full sketching system and more primitive ways of specifying models. It seems ludicrous to suggest that our first ideas in the early phases of designing an object are of cross-sections of that object. However, there are plenty of examples in architecture where the shape of a single extruded cross-section is all that is necessary to create amazing details on existing walls, window arches, and doorways. Despite the fact that people don't "think in cross-sections", I think the potential exists within the paradigm to allow easier, more interactive, and more expressive model creation than do most of the sketching systems I researched.

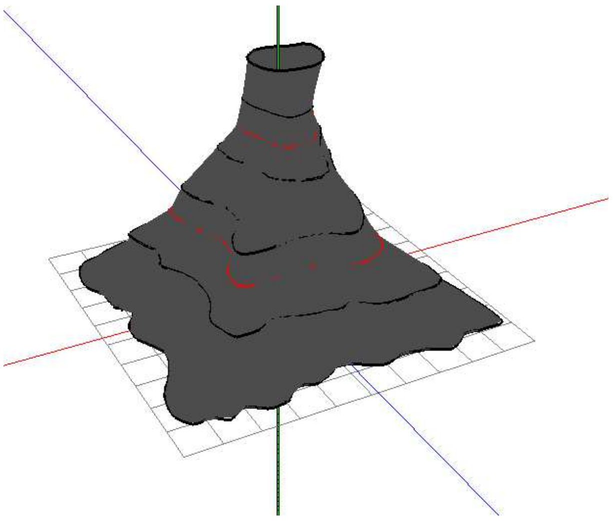
7 Conclusion

Modeling by cross-section has yielded some interesting results, with a high number of promising extensions. The example figures were easy to generate, and only the castle wall took more than a few seconds. As I mentioned previously, this approach is sketching more in the sense that it is a way of rapidly prototyping 3D objects which allows a level of imperfection that is conducive to the competing divergent and convergent processes of creative thinking. Instead of interpreting a single 2D sketch and a series of gestures to infer a 3D model, this system allows 3D models to be quickly sketched out with minimal attachment to the transient appearance of the particular shapes created. This provides the rapid visual feedback needed in early exploratory design stages, with the advantage that objects can be reprojected from any viewing angle, allowing a better understanding of their forms and properties.

Cross-sectional sketching is not a replacement for traditional 3D programs, especially those whose uses don't require more than straight, orthogonal lines and flat planes. But it has proven useful in constructing models with more control than in systems like [Igarashi et al. 1999], and freer forms than can be made in [Lipson and Shpitalni 1997] and [Shesh and Chen 2004]. It is just one way of restricting the 3D domain in a way that is useful for explicitly specifying geometry of arbitrary shapes, and as such it has its limitations. Nevertheless, the system has the potential to expand in many directions, and the paradigm may prove to be rather useful.

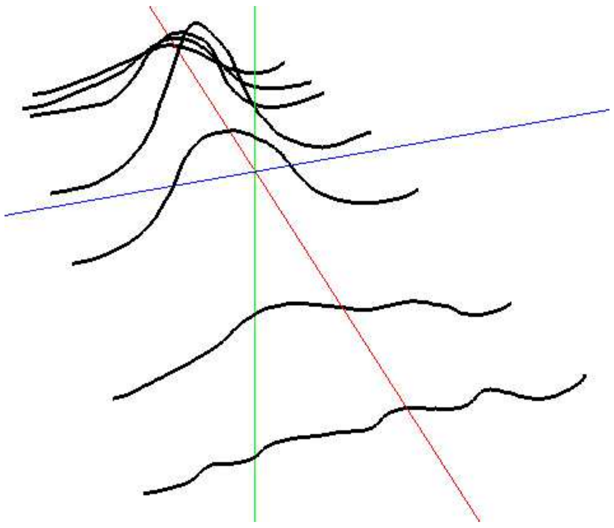


(a) A wireframe rendering of a "volcano" created in a minute or so, with user-drawn and interpolated lines.

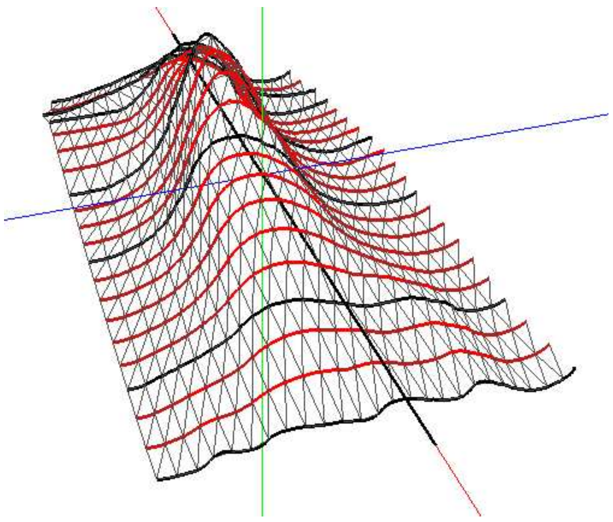


(b) Solid rendering of the volcano from a different angle

Figure 5: Another freeform object example created along a vertical spine.

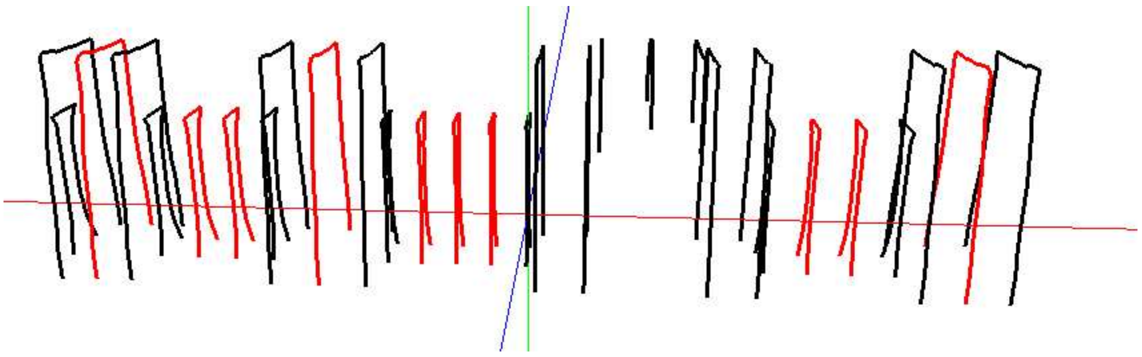


(a) User-strokes used to build terrain.

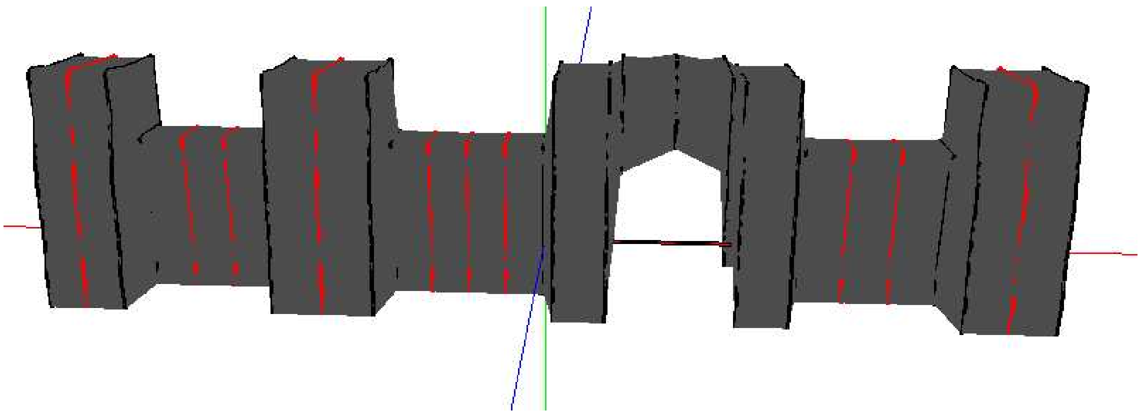


(b) Wireframe rendering of terrain object.

Figure 6: A terrain example, this time generated along a horizontal spine.



(a) User-drawn and interpolated curves forming a castle wall and gate



(b) Solid rendering of connected castle wall and gate

Figure 7: An example of simple scene geometry that was generated in about a minute using the current system.

References

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 409–416.

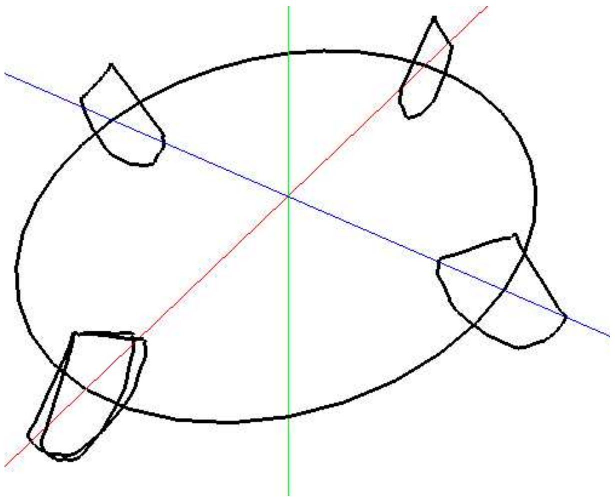
KARPENKO, O., HUGHES, J. F., AND RASKAR, R. 2004. Epipolar methods for multi-view sketching. *Eurographics Workship on Sketch-Based Interfaces and Modeling*.

LIPSON, H., AND SHPITALNI, M., 1997. Conceptual design and analysis by sketching.

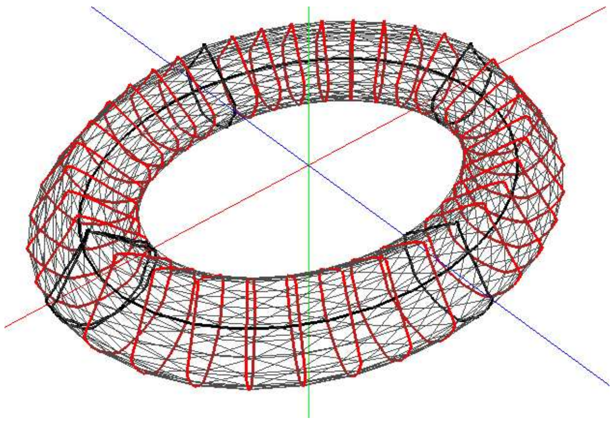
SHESH, A., AND CHEN, B. 2004. Smartpaper: An interactive and user friendly sketching system. *Eurographics 2004 23*, 3.

WANG, C. C. L., WANG, Y., AND YUEN, M. M. F. 2003. Feature based 3d garment design through 2d sketches. *Computer Aided Design 35*, 7 (June), 659–672.

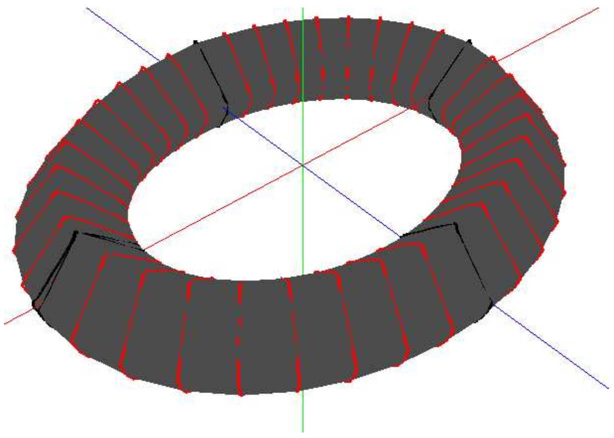
ZELEZNIK, R. C., HERNDON, K. P., AND HUGHES, J. F. 1996. Sketch: an interface for sketching 3d scenes. In *Proceedings of the 23rd annual converence on computer graphics and interactive techniques*, ACM Press, 163–170.



(a) Elliptical spine with four cross-sections



(b) Ellipse with interpolated cross-sections and wireframe



(c) Ellipse with interpolated cross-sections and solid rendering

Figure 8: Demonstrates connection along an elliptical spine.