EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2007)
M. van de Panne, E. Saund (Editors)

CrossSketch: Freeform Surface Modeling with Details

Alexis Andre, Suguru Saito and Masayuki Nakajima

Tokyo Institute of Technology, Japan

Abstract

This paper presents a novel technique to model a three dimensional freeform surface, in its global shape and small details, using a sketching interface, from a single point of view. In the past, most modeling systems that used sketches as input reconstructed the shape from the silhouette, and the user had few control on the inner parts of the result. In our system, we generate a grid of co-planar lines from a small number of strokes that the user drew, then we estimate the normal vector where it is constrained, and we form the surface by propagating this information to the whole grid. As a result, smaller strokes act locally to add detail, while longer strokes modify the whole surface. Our system gives a new approach to the modeling from sketches problem, and is intended to be a part of a more complex modeling system.

Categories and Subject Descriptors (according to ACM CCS): J.6 [Computer-aided engineering]: Computer-aided design I.3.5 [Computational Geometry and Object Modeling]: Modeling packages I.4 [Image Processing and Computer Vision]: Applications

1. Introduction

Three dimensional modeling is a tedious process. Traditional tools put the user in front of multiple projections of a three dimensional world, and the user needs to position every object with respect to these projections. However, the human visual system is able to recognize even complex shapes from single line drawings, quite instantaneously. The underlying process is based on a lot of various mechanisms, from a local understanding of the shape to a global gestalt of all visual clues. Line drawings are often associated to contours, or silhouettes, and a few carefully chosen strokes suffice to convey shape in a general meaning. Hoffman [Hof00] showed that the process is really complex, but some rules that the brain follows when reconstructing the shape from the visual stimuli have however been proposed.

From the pioneer Teddy [IMT99] to more recent work, for example [CSSJ05], the use of two dimensional sketches to model objects received a lot of attention from the modeling community.

The problem of using the drawing of an object to reconstruct its shape is challenging, as the map from surfaces to drawings is not a bijection. One silhouette drawing is the projection of a infinity of various shapes. Existing systems produce *natural shapes*, where the contours determined the boundaries of a volume that is then inflated like a balloon. While the inflation method, as well as the nature of the surface, varies, for a given system, the same contour always leads to the same shape, while a infinity of shapes shares the same silhouette. We think there is a need to give control on the inner shape to the user in order to produce more complex surfaces, and in order to do that, we need to find other sources of information to construct the inner part of the surface.

We propose here a method able to construct one surface with a few strokes, from a single frontal view, where the user specified the inside of the surface. In this paper, we do not inflate the shape, but rather construct the surface from the drawing. We chose to use only one view to stay close to the way people draw on paper.

The main contributions of this paper can be stated as follows.

First, we propose an algorithm to reconstruct three dimensional surfaces from a grid of lines, where two of them are planar and perpendicular to each other, based on simple rules of the visual intelligence (Section 4).

Second, we describe an interface that covers key points of sketch drawing, from stroke processing (Section 5.1) to the inclusion of various visual rules that gives more insight to

Sketch-Based Interfaces and Modeling 2007, Riverside, CA, August 02-03, 2007.
© 2007 ACM 978-1-59593-913-5/07/0008 $5.00

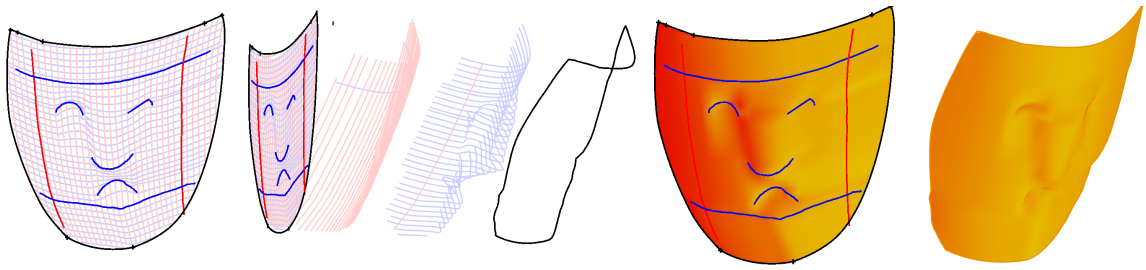

Figure 1: *A surface (right) modeled by our system from the lines drawn on the left, without the help of contour information.*

the drawing, that produces grids of lines (Section 6) with the properties needed for the previous algorithm.

2. Related Work

The creation of three dimensional objects from two dimensional input is a really challenging topic that various researchers have already heavily investigated.

SKETCH [ZHH96] is one of the first attempts to combine mouse gestures and non-photo realistic rendering to create and modify three dimensional models. Their system uses a set of gesture strokes that are then interpreted as basic shapes. The resulting objects are afterward positioned where the user wants them to be.

The pioneer work by Igarashi et al., Teddy [IMT99], is an intuitive interface that creates round shapes from closed contours, then allows the user to extrude, cut, or bend the initially created shape. Here, the user draws at each step the contour of the object he wants to model, and the system presents a plausible inflation of the shape. Such blob-like creation systems, [IMT99, KH06, KHR02, SWSJ05] are easy to use, as they provide good-looking shapes with a few strokes, but the main issue here is that the nature of the produced shape is fixed by the system, as the user has little control on the inner aspect of the shape. Furthermore, to model crease and sharp features often requires a cut operation. We would like to by-pass this cut operation at the creation step. The following systems allow some control on the inner shape.

"Harold, a world made of Drawings" [CHZ00], proposed one simple way to generate terrain by drawing a stroke starting and ending on the ground, indicating the silhouette of the desired terrain. Our system takes a similar view of modeling, by drawing planar strokes that lie on the desired surface.

Cherlin et al. [CSSJ05] used various techniques to generate interesting shapes. Their approach was inspired by real drawing techniques, such as the spiral method, where the shape depiction comes from a spiral stroke bounded by the silhouette. Their system is able to generate complex shapes with few strokes, but only one part at a time. As a result, the creation of complex objects takes some time, but most of it is used to assemble the various parts.

Ijiri et al. [IOOI05] present a sketch-based system focused on plants, especially leaves and petals, in a well-thought combination of free-form modeling and bending operations. While the system can create realistic models of various flowers, the purpose is too specific for the system we want to build.

Some sketching interfaces uses additional information such as shade and shadow to reconstruct the object with more details, for example [SLKM04], where the amount of shade is used to inflate more or less the volume. Narrower shadows result in more elongated shapes.

Another related work can be found with the sketch based interface for mesh editing by Nealen et al. [NSACO05]. They provide a framework able to modify existing meshes using various types of strokes while preserving the details. The available operations can deform one part of the silhouette, or modify one line drawn on the mesh. Our system shares common points with this work, as we also allow detail editing. Our main focus, though, is to create the surface from scratch.

Another approach is to construct the shape once the user has finished his drawing. 3D Sketch [MSK00] takes the sketch of an object, maps the strokes to the edges of a cube, and inflates the corresponding cubic form, while preserving the style of the strokes. The system provides a nice interface to sketch a whole range of objects, as long as they can be mapped to a cube. Such limitation makes the strength and the weakness of their system. We want a system able to model any shape, without any constraint.

While it is not strictly the same domain of application, single-view reconstruction of images (with some user interaction) present many similarities with our objective. Prasad et al. [PF06] mapped the visible edges of an object to a plane in the 3D contour generator domain, then inflate the shape with the constraints on the contours. The method works well with objects of cylindrical topology and of various genus. The approach we use is different, as we do not use the edge or contour information at all.

We also refer the reader to a survey of sketch-based modeling [CPCN05] for a broader overview.

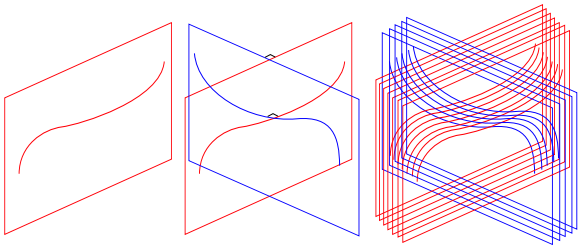

Figure 2: *One line is seen as a planar stroke. Two lines are seen orthogonal to each other, and similar strokes appear similar, in this case, parallel in 3D.*

3. Vision of a grid of lines

3.1. Reconstruction problem

We will consider the following framework for our system. The user draws strokes on the plane $z = 0$, under an orthographic projection, that is, the x and y coordinates of each stroke point are known, but not the z one. The main problem is now to find the desired z coordinate for every point in the sketch, on the whole surface. We chose to use in the beginning an orthographic projection, and we hope to deal with other cases in the future.

However, a given line drawing is the projection of an infinite number of objects, so how do we reconstruct the correct surface? In order to do so, we use the following guidelines that have been established about the way people see and reconstruct lines.

3.2. Perception of lines

Stevens [Ste82], as well as Hoffman [Hof00] provide a set of rules that our visual system follows. The rules of interest for the problem we consider are the following:

1. Similar lines stay similar.
2. One curved stroke is seen as lying on a plane.
3. Two intersecting lines are seen as orthogonal to each other.

These rules hold under the general viewpoint assumption [Fre94]. This assumption is to suppose that the drawing is drawn from a generic view, where the drawing is stable when the viewpoint slightly moves. Linked lines (in 3D) stay linked even under a rotation of the viewpoint, while coincidental joints, due to the perspective, split. We suppose that the user draws from a generic viewpoint. We believe this is a safe assumption as the user wants to model one surface, and they do their best not to draw lines with ambiguity.

These rules allow us to reconstruct the shape from a grid of two sets of similar lines. Figure 2 gives the outline of the idea. One line is viewed as a planar line. Another line, intersecting the first one, is viewed as a different planar line, and the angle between the two is a right angle. One representative of each set of similar lines, the **backbone** of the other group, can then be reconstructed in 3D, by first computing the planes on which they appear to lie. Similar lines, close to the representative line of the set are reconstructed by translating the plane of the corresponding representative, and projecting the new lines onto it.

We will in the next section present the details of our reconstruction algorithm, and in Section 5 present a system able to produce the input grids of similar lines with a few user-drawn strokes.

4. Reconstruction

4.1. Projection of the normal vector

We suppose that we have a drawing of a grid, similar to the one on Figure 2, and one representative of each set of lines. From now on, we will refer to such strokes as "hatching" lines. We use the term hatching, however such strokes are not to be confused with cross-hatching strokes used to depict shade or shadows, and whose orientation may not be related to the underlying object.

Our algorithm depends on two main strokes chosen in the hatching structure, the backbones described in the previous section. As stated before, the two strokes appear orthogonal. However, this is not enough to know the orientation of the two planes. We will use here the geometric properties of the projection of a corner of a cube to estimate the normal vector to the surface at this particular point.

4.2. Orientation of the backbones

Cubic Corners, studied by Perkins [Per68] while investigating the perception of cubic shapes from the point of view of human vision, are easy to visualize and to draw with three right angles. His main interest was to understand when human beings see the corner of a cube in three concurring lines. Meanwhile, he established the relation between the angles of the lines and the perceived depth under an orthographic projection. This result has been used in [MVS05] to reconstruct objects where the orthogonality prevails.

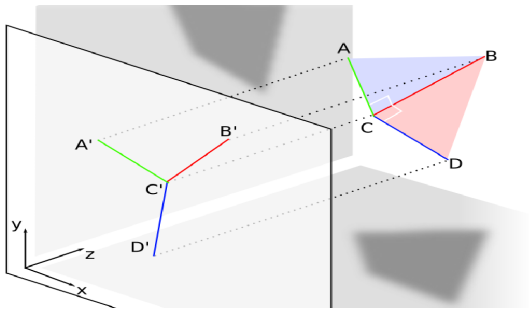

Figure 3: *A cubic corner.*

Using the notations of the Figure 3, we have:

$$z_C = z_D \pm L_{C'D'} \tan(\arcsin(\sqrt{\cot \widehat{A'D'C'} \cot \widehat{B'D'C'}})), \quad (1)$$

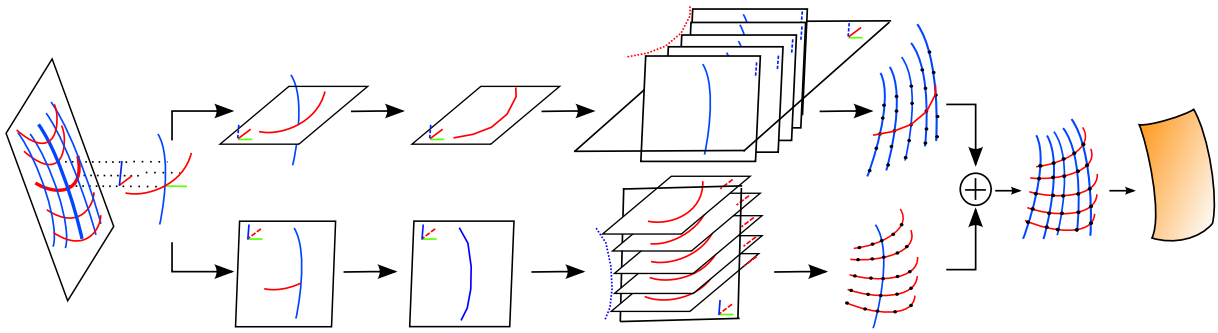

Figure 4: *For a given set of hatching lines, and from the normal vector, we reconstruct the two backbone strokes (second from the left). We then use these 3D strokes to create a series of projection planes for the other group of strokes, resulting in two groups of 3D hatching lines. Finally, we take the mean of the two groups to reconstruct the object.*

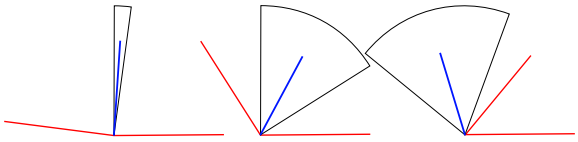

Figure 5: *For the two vectors in red, the third vector must be inside the black regions. We choose the bisector of that region to estimate the normal vector.*

where z_X represents the z-coordinate of the point X, L_{AB} is the length of the segment AB. The two possible results come from the *Necker*'s reversion. Moreover, not all triplets of lines can be the projection of a cubic corner, as Perkins remarked. The projective consistence imposes that all three angles around the corner must be greater than $\Pi/2$ for a Y-junction, and that two angles around the corner, each less than $\Pi/2$, must sum to greater than $\Pi/2$ for a W-junction. For a given pair of vectors, the third vector must lie in a specific zone in order to be a possible corner. Moreover, as Stevens [Ste82] remarked, the available domain, depending on the two others vectors, may be strongly restricted. He suggested the use of the bisector of the available domain, when strongly restricted, as an approximation of the third vector. We apply the same approach, however, we do not have the liberty to look for strongly constrained crossings, as we have only one point where the two representatives of each group cross. In most cases, however, taking the bisector at that particular point still produces acceptable results, see Figure 5. In the event of a bad result, the user is allowed to draw the normal vector on the intersection. The choice of the backbones is capital for this method to work well. As a result, the system is difficult to manipulate in the beginning.

4.3. Propagation along the backbones

Once the normal vector is known, we can extend the 3D information to the whole surface. We first approximate these two strokes as planar strokes. As we know the projection of all the points on the drawing plane, reconstructing the stroke is straight forward once the original plane is specified. The orientation of the strokes is now fixed, and the strokes are reconstructed (Figure 4, second from the left). From these two backbones, we will inflate the other hatching strokes in a similar way. For simplicity, we will describe one group, the process being similar for the other.

Now that the backbone for a particular group is known, (for example, the red backbone on the Figure 4, top line). All the blue strokes (the opposite hatching group) are reconstructed in 3D by projecting them to planes orthogonal to the backbone, that is, all the strokes present the same orientation. Once all the hatching strokes of the two groups have been reconstructed, the original intersections points of the two hatching groups are averaged to produce the shape.

5. Interface

In this section, we describe the steps needed to create grids of lines that suits the previous algorithm. Strokes are essential in this paper, as they contain all the information available. We therefore apply specific pre-processing to the strokes in order to get the most meaningful information. First, we need to filter out the stroke from the raw input data.

5.1. Stroke Sampling

Our system accepts as input strokes ordered sets of points, coming from any conventional 2D input device. A pen tablet, for example, was used to produce all the figures of this paper. We implemented a slightly improved version of the sampling method of [SSD01], where corners of the strokes are extracted from the set of local minima of speed (the user tends to slow down at corners) and from the set of local maxima of curvature (corners present high curvature profiles). Each segment of the stroke is then interpolated as a succession of Bezier curves until the error between the input points and

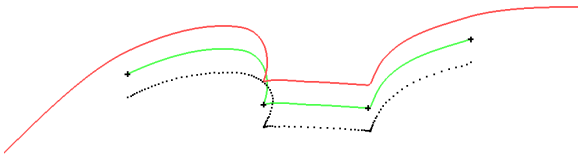

Figure 6: *The raw data points in black, the approximated set of Bezier lines and the corresponding corners in green, the extension result in red.*

the resulting approximation is below an small threshold. We then enforce C_2 continuity on the succession of curves for each segment, by slightly adjusting the control points. This gives smoother strokes, and the error stays small. Without the C_2 continuity, the resulting shape would present irregularities that would appear on the shading. Figure 6 shows one original point set and its corresponding stroke interpolation and extension.

One particular advantage of this method is that the stroke is sampled between *corners*. This allows the user to draw strokes with sharp angles. As the reconstruction method is able to deal with such angles, this gives more freedom to the category of shapes this system can reconstruct.

When drawing with a pencil, there is no *modes* or anything that holds information about the type of strokes being drawn. Once the drawing is done, anyone can classify the strokes between two groups, silhouette (or edges) and hatching (two directions), without any doubt for most cases. We want here to reach a similar level of *simplicity*, that is the user must be able to draw his sketch without specifying anything.

5.2. Classification of the strokes

The user must start by drawing the edges of the surface he wants to model. While he is drawing strokes, we construct a graph where the nodes correspond to corners and ending points of each stroke, and links correspond to the strokes.

Internally, a stroke is classified into one of the following groups: unknown, edge, hatching (unknown), first group of hatching lines, second group of hatching lines. Groups of hatching lines are valid for any surface of the object, so we need first to determine possible surfaces. To do so, we look for shortest loops in the set of strokes of type unknown or edge using Djikstra's shortest path algorithm. For any loop, we suppose there is a surface whose edges correspond to the loop in the graph. Each of these edges is then classified as a edge.

For hatching lines, we use the following approach. If a stroke is not linked with any known edge, we look for a surface that may contain a big portion of the stroke (greater than a fixed threshold). If such surface exists, we suppose that the stroke is a hatching stroke, and we classify it as a stroke of type hatching (unknown) for the given surface.

5.3. Hatching strokes

Now we need to separate all the hatching strokes into two and only two groups, as we made this assumption. We will create the two groups incrementally, i.e. when a new hatching stroke is found, we classify it with the information we have to this point. When the stroke crosses an existing hatching stroke, the new stroke belongs to the other hatching group.

However, when no intersection point can be found, the new stroke is matched against all other hatching strokes already present in the same surface, and the new stroke is categorized as the same type of its closest match. In the initial case, the first hatching stroke is labeled as first group of hatching lines, then as long as new strokes are similar to this group, they end up in the same group. If the new stroke's minimum similarity value to the first group is higher than a threshold, the stroke is then categorized as second group of hatching lines. After that, the closest group or the one that the new stroke crosses is chosen.

As a result, two strokes from the same hatching group can not intersect, but such cases are not included in the domain of shapes we aim to reconstruct.

5.4. Similarity between two strokes

We need a way to measure the similarity of two strokes. We use a modified version of the Hausdorff distance. The Hausdorff distance is suitable for measuring the distance between two sets of objects, in our case, sample points of the stroke. The most common version of the distance is not suited for our problem, as we look for *similar* strokes, i.e. the closest stroke from the *distance* point of view may not at all be related to the closest stroke from the *shape* point of view. The version we use includes a translation of the first set in order to find the best match.

We use the following definition to calculate the similarity S of the sets A and B:

$$S(A,B) = \min_{t} \max_{a\in A} \min_{b\in B} \|a-b+t\|, \qquad (2)$$

where t is the translation vector, and $\|.\|$ is some norm in the considered space. We use the L_2 norm in the drawing plane. Each stroke is re-sampled as a set of equidistant segments, and the extremities of these segments form the set of points needed for computation. The reader may refer to Huttenlocher et al. [HKR93] for a more complete study on this particular measure.

6. Populating the surfaces

Once we have some samples of the hatching strokes inside one surface, we need to extend them to the whole surface. The problem has two aspects. First we need to elongate the strokes until they reach the surface boundaries, and also to

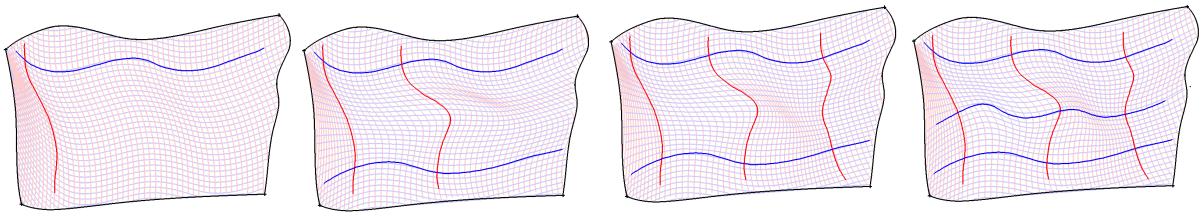

Figure 7: *Example of surface population. At first, with only two strokes, the hatching structure is constructed with extended and translated copies of the original strokes. As the number of strokes increases, the hatching lines between two strokes of the same group are interpolated.*

match the length of the other strokes. We then need to approximate the hatching strokes where the user did not draw anything.

6.1. Stroke extension

In order to elongate the strokes in a natural manner, we compute the curvature along the strokes using a triangular approximation of a circle. We then take the average of the curvature in the neighborhood of the ending points of the strokes, then we extend the strokes with new points, keeping the curvature equal to the minimum of the mean curvature and the last curvature calculated, as the strokes tend to present higher curvature near the ending points. We add segments of the same length, and the angle with the previous segment of the stroke is chosen such that the curvature of the newly added part of the stroke matches the desired curvature. In order to add more realism, we slightly decrement this desired curvature along the extension points, resulting in strokes that tend to straight lines. See Figure 6 for one example.

6.2. Interpolation of hatching lines

In this part, we interpolate new strokes that follow the global shape of the input strokes, in order to cover the whole surface with hatching information to provide feedback to the user. The way strokes are interpolated influences the resulting shape. Our approach is similar to the method described in [RK00], when building hatching lines from the principal directions of curvature. On the contrary, we want here to interpolate hatching lines from the input lines, in a natural way.

For a given set of hatching strokes, each stroke is resampled to a series of angles, that is, each stroke is made of segments of equal length, and we know the angle between one segment and the next one. Then, along the corresponding stroke (backbone), we interpolate the strokes (ribs) between the reference strokes (the strokes the user drew) using a linear interpolation of the series of angles.

Figure 8 shows the process. The stroke on the left is sampled as the set $\{A_i | i = 0...n\}$ while the stroke on the right comes as $\{B_i | i = 0...n\}$. Each stroke in the middle of those two strokes in created as $\{X_i = (1-t)*A_i + t*B_i | i = 0...n\}$, where t is the corresponding sampling parameter of the backbone stroke, and n the length of the smallest stroke, but we extended the strokes in order to cover the whole surface in the previous section. For strokes on the outer sides, we use the last user-given stroke in that direction, and we translate it to populate the sides.

This method gives natural results (seen on Figure 7), while other methods of interpolation, for example Coons' patches, would produce very different results, as every interpolation would have been done on four-sided-surfaces, with a number of sample points that needs to be the same along corresponding edges. The method we use here allows us to generate hatching structures with only two or three strokes. However, since we interpolate the strokes using the arc-length of the ones the user draws, unexpected results may occur, for example when one really long stroke, shaped like a V, used to depict a bump, is interpolated with a straight stroke half the length. The bump will spread along the stroke direction instead of a more natural decrease in amplitude.

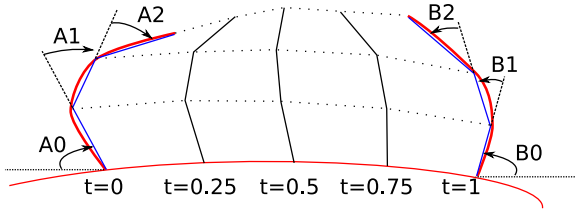

Figure 8: *The ribs generation. For a given backbone stroke (the bottom line), we sample the crossing lines as a series of segments of constant length, taking the relative angles for the calculation of the approximation along the sampled backbone.*

6.3. Detail addition with small strokes

The previous interpolation is done as soon as there is enough information to create hatching strokes on the whole surface. Of course, the generated strokes may not be right for complex surfaces, so we allow the user to redraw on the surface to specify details, in a *oversketching* way. As the user

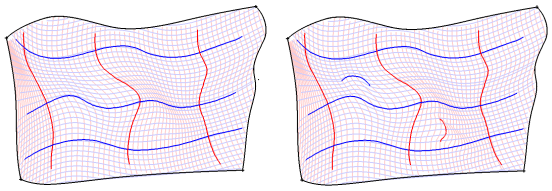

Figure 9: *Adding detail by oversketching.*

is given interpolated hatching lines over the whole surface, he is able to see exactly what and where he needs to provide more detail.

When the user inputs a small stroke (where the length of the stroke is less than the mean of the other strokes times a reducing factor, 0.3 in our implementation), the system will categorize the stroke as a correction stroke, that is, the user wants to modify the local structure of the surface rather than the whole surface. We then localize the stroke in the previously created hatching structure, and we recreate a whole hatching stroke using the previously interpolated data in the beginning, the new input stroke in the middle, then one more time the previously interpolated data. Then the stroke is processed as if it were a complete hatching stroke.

This allows oversketching in a natural way. When the stroke does not match the user's wishes, the user can redraw on the stroke, and it will be corrected locally. This also removes the need for an undo function. This allows step-by-step modeling of the surface. The first strokes decide the global shape of the surface, then small strokes can be added to add small details to the shape.

7. Results and Discussion

The described method was implemented in Java-JoGL. On a AMD Atlhon 64 3400+ with 1GB of memory running Linux, the system is interactive. The main time-consuming parts are the numerous re-sampling steps of the strokes, as well as the interpolations. The reconstruction of the 3D shape is fast in comparison, and most of the time is spent computing the intersection of the surface with the edges of the surface. We present here some results obtained by our system. Figure 1 shows a mask modeled in less than two minutes (drawing and processing time included) by a trained user, with three edges, four complete hatching strokes and eight correction strokes. Figure 10 shows a single leaf modeled with just two strokes to specify the inner shape, resulting in a simple shape. While the strokes needed to model such a leaf are similar to the ones needed to model a similar leaf using the cross-sectional blending surface method of [CSSJ05], our method was meant to allow the modeling of objects on the spot, with the correct orientation. Moreover, the stroke needed by [CSSJ05] to specify the cross-section seems to be drawn from a different viewpoint, while the strokes needed in our case are consistent with the viewpoint. Figure 11 shows a stairs-like shape, where sharp angles are dominant.

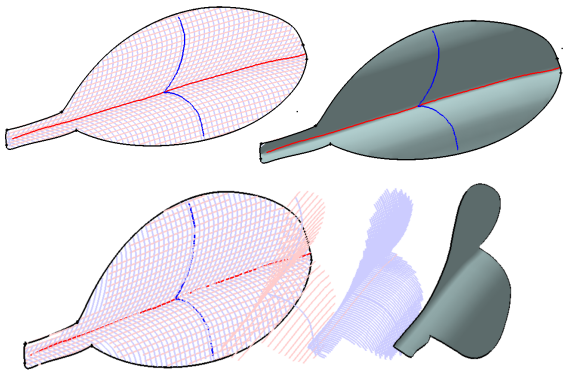

Figure 10: *A leaf modeled with three strokes*

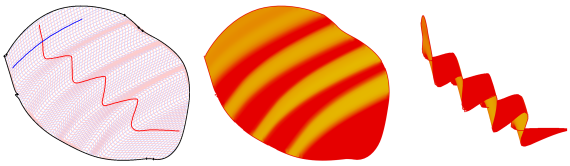

Figure 11: *A shape with sharp angles.*

The rules about our visual system are merely guidelines, and they do not hold under any circumstances, and not for all drawings. For surfaces close to a plane, or a cylinder, the proposed guidelines apply, and as a result, we can reconstruct the surface with confidence. In the case of a sphere, however, the grid that complies with the rules is not natural to draw. As a result, it is difficult in the current state of the system to model objects of spherical topology.

Another limitation of the system is the fact that only frontal geometry is reconstructed, and self-occluding parts are impossible. As a result, the models designed with this system are meant to be viewed from a viewpoint close to the one used during the modeling. Moreover, animations of the models are difficult, as changing the orientation of the models is not easy with the current method.

The system, in its current state, is far from the level of precision that other projects that allow multi-view sketching reach, even if we believe that it can scale to complex surfaces. Our main focus was to allow complex modeling from a single drawing, with small details. As our method works for a single view point, we look forward to implement it in a wider system, where our system could be used to specify local details or global changes. We also believe that an understanding of the reconstruction of one single drawing can lead to a more direct link between artists, used to draw on one piece of flat paper, and modelers, used to think in a 3D world.

8. Conclusion and Future Work

We presented a new approach to the modeling from sketches problem, using inner lines in order to apprehend the inner shape of the current drawing. We showed that simple surfaces can be modeled in a few strokes, and that the resulting system is usable.

Moreover, our system focuses on drawing from a unique point of view, and that was successfully achieved, while being able to produce objects of various shapes. Building objects using multiple drawings from different viewpoints is different from the framework we used in this paper, but we hope to integrate our method to specify the inner details of shapes constructed with more powerful systems soon.

Another direction for future work is to use the important information the user gave in this paper to produce cross-hatched rendering images of the objects. As stated before, the user makes corrections where the shape does not follow the global hatching structure. Using the location of such small strokes in order to control the density of a cross-hatching generator may lead to interesting renderings. It may also helps the user to apprehend how the system interpreted his drawing.

We also hope in the near future to adapt the algorithm to spherical objects, using for example rotated planes along the other backbone to reconstruct the strokes. Lastly, we hope to extend the system to model more than one surface at the same time.

References

[CHZ00] COHEN J. M., HUGHES J. F., ZELEZNIK R. C.: Harold: a world made of drawings. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering* (New York, NY, USA, 2000), ACM Press, pp. 83–90.

[CPCN05] COMPANY P., PIQUER A., CONTERO M., NAYA F.: A survey on geometrical reconstruction as a core technology to sketch-based modeling. *Computers & Graphics 29*, 6 (2005), 892–904.

[CSSJ05] CHERLIN J. J., SAMAVATI F., SOUSA M. C., JORGE J. A.: Sketch-based modeling with few strokes. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics* (New York, NY, USA, 2005), ACM Press, pp. 137–145.

[Fre94] FREEMAN W. T.: The generic viewpoint assumption in a framework for visual perception. *Nature 368* (7 April 1994), 542–545.

[HKR93] HUTTENLOCHER D., KLANDERMAN D., RUCKLIGE A.: Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*, 9 (September 1993), 850–863.

[Hof00] HOFFMAN D. D.: *Visual Intelligence: How We Create What We See*. W. W. Norton & Company, February 2000.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 409–416.

[IOOI05] IJIRI T., OWADA S., OKABE M., IGARASHI T.: Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM Press, pp. 720–726.

[KH06] KARPENKO O. A., HUGHES J. F.: Smoothsketch: 3d free-form shapes from complex sketches. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), ACM Press, pp. 589–598.

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

[MSK00] MITANI J., SUZUKI H., KIMURA F.: 3d sketch: Sketch-based model reconstruction and rendering. In *IFIP Workshop Series on Geometric Modeling: Fundamentals and Applications, 7th Workshop GEO-7* (2000), pp. 85–112.

[MVS05] MARTIN R. R., VARLEY P., SUZUKI H.: Perpendicularity as a key to interpreting line drawings of engineering objects. *IJCC Workshop on Digital Engineering* (2005), 115–120.

[NSACO05] NEALEN A., SORKINE O., ALEXA M., COHEN-OR D.: A sketch-based interface for detail-preserving mesh editing. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM Press, pp. 1142–1147.

[Per68] PERKINS D.: Cubic corners. *Quarterly Progress Report*, 89 (1968), MIT Research Laboratory of Electronics, 207–214.

[PF06] PRASAD M., FITZGIBBON A.: Single view reconstruction of curved surfaces. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 1345–1354.

[RK00] RÖSSL C., KOBBELT L.: Line-art rendering of 3D models. In *Computer Graphics and Applications, 2000. Proceedings. The Eighth Pacific Conference on* (2000), pp. 87–96.

[SLKM04] SHIZUKA H., LIU W., KONDO K., MATSUDA K.: A sketch interpreter system with shading and cross section lines by freehand drawing. In *Proceedings of the 11th International Conference on Geometry and Graphics (ICGG2004)* (8 2004), pp. 357–362.

[SSD01] SEZGIN T. M., STAHOVICH T., DAVIS R.: Sketch based interfaces: Early processing for sketch understanding. *Workshop on Perceptive User Interfaces, Orlando FL* (2001).

[Ste82] STEVENS K.: Implementation of a theory for inferring surface shape from contours. *A.I. Memo*, 676 (1982), MIT Artificial Intelligence Laboratory.

[SWSJ05] SCHMIDT R., WYVILL B., SOUSA M. C., JORGE J. A.: Shapeshop: Sketch-based solid modeling with blobtrees. In *Proceedings of the 2nd Eurographics workshop on Sketch-Based Interfaces and Modeling,* (8 2005), pp. 53–62.

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: Sketch: an interface for sketching 3d scenes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM Press, pp. 163–170.