IEICE TRANS. INF. & SYST., VOL.E92–D, NO.6 JUNE 2009

PAPER

Single-View Sketch Based Surface Modeling

Alexis ANDRE[†a)], *Nonmember*, **Suguru SAITO**[†], *and* **Masayuki NAKAJIMA**[†], *Members*

SUMMARY We propose a sketch-based modeling system where all user input is performed from a unique viewpoint. The strokes drawn by the user must not then be restricted to the drawing plane: their orientation in the 3D space is automatically determined by the system. The desired surface is reconstructed from a grid made of two groups of similar lines, that are considered co-planar. The orientation of the two sets of planes is determined by assuming that at the intersection of a representative line of each group, those two lines are perpendicular.

key words: *sketch-based modeling, single view, computer vision*

1. Introduction

The current design workflow often starts with the designer sketching a rough idea on a piece of paper, then modeling the shape in a 3D modeling software, using the previous sketch as reference. The ultimate goal of this research is to create the 3D shape directly from a unique design sketch. To help the user understand the shape of an object, traditional modeling tools rely on various projections of 3D objects on the screen, while the human brain is able to infer the shape of even complex objects by looking at a single view of the object, in our case, a drawing or a sketch. This work describes a framework to reconstruct shapes drawn from a unique viewpoint.

How our brain reconstructs the shape from a few visual clues is a challenging topic dealing with phenomena outside the scope of this paper. We refer the reader to [1] that presents the state of the art in the domain. While the process is complex, some rules that govern how we interpret shapes exist, and we will base this work on them.

The problem we are tackling is not obvious. One silhouette drawing is the projection of a infinity of various shapes, so the traditional approach is to produce natural shapes, where the contours determine the boundaries of a volume that is inflated like a balloon. The issue here is that the same outline gives always the same shape.

We think there is a need to give more control to the user on the inner shape in order to produce more complex surfaces, and for that purpose, we need to find other sources of information (that is, not only the contour information) to define the inside part of a surface.

We propose here a method able to construct one surface with a few strokes drawn from a single 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, with the ultimate goal of automatic single sketch-recognition in mind.

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 two-dimensional lines, where two of them are planar and perpendicular to each other, based on simple rules of the human visual intelligence. Second, we describe an interface that covers key points of sketch drawing, producing grids of lines with the properties needed for the previous algorithm. The system described in this paper allows then the sketching of free-form surfaces without changing the drawing viewpoint. It is then possible to create the 3D shape of a previously existing paper sketch, directly in the correct orientation and position, by tracing over the concept sketch.

The paper is organized as follows. After a review of prior work, we describe the rationale behind our reconstruction method (Sect. 3). We then describe how we process the user drawn strokes (Sect. 4 and Sect. 5) before reconstructing the surface in 3D (Sect. 6).

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 [2] is one of the first attempts to combine mouse gestures and non-photo realistic rendering to create and modify three dimensional models. It 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 [3], 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 [3]–[6] 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, those systems are not able to create sharp features without using a cut operation, and every

Manuscript received January 8, 2009.

[†]The authors are with Tokyo Institute of Technology, Tokyo, 169–8555 Japan.

a) E-mail: alexis@img.cs.titech.ac.jp

DOI: 10.1587/transinf.E92.D.1304

stroke is drawn on a different virtual canvas, seen from a different viewpoint, that is not the focus of this paper. The following systems allow some control on the inner shape.

"Harold, a world made of Drawings" [7], 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. [8] 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. [9] 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. In [10], 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. [11], with its extension [12]. 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.

Another approach is to construct the shape once the user has finished his drawing. 3D Sketch [13] 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, and present strong symmetry. 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. [14] 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 shape and of various genus. The approach we use is different, as our method does not rely on contour information. Zhang et al. [15] proposed a method to reconstruct 3D models using a sparse set of user-specified constraints on the local shape, such as normal vector constraints, depths discontinuities, and plane constraints. While the results are interesting, the user is deeply involved in the reconstruction process, specifying non-intuitive information.

We also refer the reader to a survey of sketch-based modeling [16] for a broader overview.

3. Perceptual Insights

The problem is the following: the user draws strokes in the $z = 0$ plane, under an orthographic projection. The challenge is then to find the desired z coordinate or height of every point in the sketch, not only for points on strokes but also for points in-between. A given line drawing is however the projection of an infinite number of objects, so how do we reconstruct the correct surface?

3.1 Perception of Lines

Stevens [17], as well as Hoffman [1] provide a set of rules that the visual part of the brain follows, that will act as guidelines for the reconstruction process. The rules of interest for our setting are the following:

1. Similar lines in 2D are seen similar in 3D.
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 [18]: the drawing is stable under slight changes of the viewpoint. We suppose that the user draws from such a general viewpoint, a safe assumption as the user wants to model one surface without ambiguity.

The human perception of such grids of lines has also been investigated in order to improve the quality of line rendering of 3D shapes. The widely adopted principal directions of curvature are always perpendicular to each other and form grids of lines with similar orientations: an argument for their use can be found in [19], and an evaluation of various grids made of different types of lines showed that such grids do efficiently convey the underlying shape [20]. A study on the orientation of texture patterns also showed that grid-like patterns helped to understand the shape [21].

These rules allow us to reconstruct the shape from a grid of two sets of similar lines. Figure 1 explains of how our brain might reconstruct a 3D structure from such a grid. From a grid made of two sets of similar lines, lines in one set can be thought as coplanar. We now need to know the orientation of one plane per set to project the lines in 3D space. For this purpose, we extract one representative line of each set, the backbones, and as they appear perpendicular, we suppose that they are really perpendicular in 3D space. From these assumptions, we can compute the orientations of the planes, and project each line back to 3D space. From now on, we will refer to such strokes as "hatching" lines. Such strokes are not to be confused with cross-hatching lines used to depict shade or shadows, and whose orientation may not be related to the underlying object.

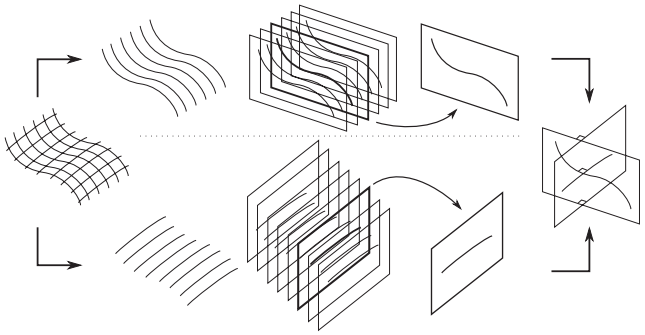


Fig. 1 From a grid of lines that can be split in two similar groups, the rules of our visual system make us see each group as lying on parallel planes, and the representatives of the groups are seen orthogonal to each other.

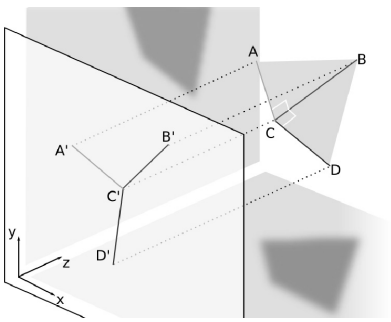


Fig. 2 A cubic corner: the angles made by the projection of the three edges of the corner of a cube are enough to know their orientation in 3D.

3.2 Cubic Corners

Assuming that the two planes are orthogonal is not enough to compute their orientation, as there is still one degree of freedom available. We will use here the geometric properties of the corner of a cube to estimate the normal vector (a third vector that is orthogonal to both backbones) to the surface at this particular point.

Cubic Corners, three edges orthogonal to each other, studied by Perkins [22] 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 [23] to reconstruct objects with orthogonal faces.

Using the notations of Fig. 2, we have:

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

$$\alpha = \cot \widehat{A'C'D'} \cot \widehat{B'C'D'} \quad (2)$$

where z_X represents the z-coordinate of the point X, $L_{C'D'}$ is the length of the segment $C'D'$. The two possible results come from Necker's reversion [24].

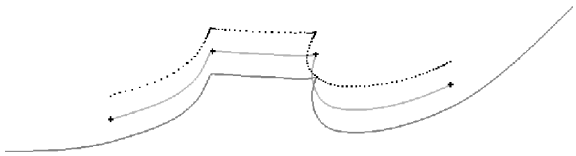


Fig. 3 Sampling process and extension: from the raw input data (top), the stroke's corners are detected then the segments in-between are approximated as Bezier curves (middle), finally the stroke is extended at both ends (bottom).

4. Stroke Processing Toolbox

Strokes are essential in this paper, as they contain all the information available. We therefore apply specific preprocessing to the strokes in order to get the most meaningful information. We describe the various processing methods we are using: stroke sampling, stroke elongation and stroke similarity.

4.1 Stroke Beautification and Elongation

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 the sampling method of [25], 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). The segments between corners are then approximated as Bezier curves. We modified [25] by enforcing C^2 continuity on the strokes (by moving the control points at the junctions so that they are aligned), to get smooth shading. Figure 3 shows one example of beautification.

One particular advantage of this method is that the stroke is sampled between corners: each segment (between two corners) can be resampled with any sampling interval. This also 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.

In order to elongate the strokes in a natural manner, we compute the curvature along one stroke every three sample points using the 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. In order to get smoother lines, we slightly decrement the desired curvature along the extension points, resulting in strokes that tend to straight lines. See Fig. 3 for examples of the extension process.

4.2 Similarity between Two Strokes

To measure the similarity of two strokes, we use a modified version of the Hausdorff distance, similar to the one used in [26] to register objects in photos. Adding a possible translation to the distance allows us to consider the result as a measure of the similarity in shape of the two strokes. We use the following definition to calculate the similarity S between two sets of points A and B:

$$S(A,B) = \min_{t} \max_{a\in A} \min_{b\in B} \|a - b + t\|, \tag{3}$$

where t is the translation vector, and $\|.\|$ is some norm in the considered space. We use the L_2 norm in the drawing plane. To apply this measure to strokes, 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.

5. Grid Generation from User Input

In this section, we describe the steps needed to create two sets of similar lines from user input. We require the user to draw only guidelines of the grid, specifying only a few strokes that will be interpolated to create a grid. In the examples, strokes drawn by the user are bold, and the other strokes comes from the grid generation process described in this section.

5.1 Classification of the Strokes

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, any human being 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.

We start by classifying the strokes into edges. We then trace connected strokes until we find a loop, and we consider the inside of the loop as a surface. Each stroke inside this surface is then classified as a hatching stroke.

Now we need to separate all the hatching strokes into two groups, as we made the assumption that a surface contains two sets of hatching lines. We will create the two groups incrementally, i.e. when a new hatching stroke is drawn, we classify it with the information available at 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, we look for the most similar (as in Sect. 4.2) strokes in the already classified ones. The new stroke is then classified as belonging to the same group as 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, as such cases are not included in the domain of shapes we aim to reconstruct.

5.2 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, while matching the length of the other strokes. We then need to approximate the hatching strokes where the user did not draw anything.

We interpolate here new strokes between the input strokes, in order to cover the whole surface with hatching information. Our approach is similar to the method described in [27], 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 user-given hatching strokes, each stroke is re-sampled 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 backbone, we interpolate strokes (ribs) between the reference strokes (the strokes the user drew) using a linear interpolation of the series of angles, on both sides of the backbone.

Figure 4 shows the process. The stroke on the left is sampled as the set $\{A_i | i = 0 \dots n\}$ while the stroke on the right comes as $\{B_i | i = 0 \dots n\}$. Each stroke in the middle of

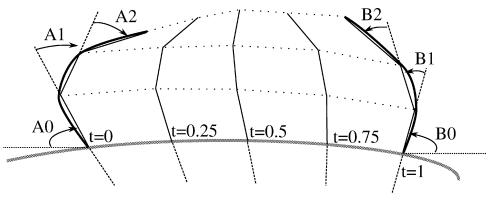


Fig. 4 The ribs generation. For a given backbone stroke (the light gray line), we sample the crossing lines as a series of segments of constant length, taking the relative angles for the calculation of the approximation along the sampled backbone. A similar interpolation is done on the other side of the backbone.

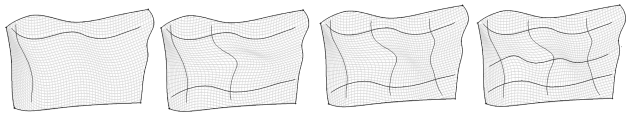


Fig. 5 Example of surface population. Strokes in bold denote user input. At first, with only two strokes, the hatching structure is constructed with extended and translated copies of the original strokes. As the number of strokes increases, the hatching lines between two strokes of the same group are interpolated.

those two strokes in created as $\{X_i = (1-t)A_i + tB_i | i = 0 \ldots n\}$, where t is the corresponding sampling parameter of the backbone stroke, and n is the length of the smallest stroke. Thanks to the previous extension, the length of the smallest stroke is enough to cover all the surface. For strokes on the 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 Fig. 5). This allows us to generate hatching structures with only two or three strokes. However, since we interpolate the strokes using the arc-length of the user-drawn stokes, 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 size. The bump will spread along the stroke direction instead of a more natural decrease in amplitude.

6. 3D Reconstruction of a Grid

We suppose that we have a grid, similar to the one on Fig. 1, with two groups of similar lines. We describe here how to infer the 3D information for all lines in the grid.

6.1 Backbone Selection and Normal Vector Estimation

The backbones are selected as follows: the first user drawn stroke of each group is the backbone for the other group. We will now estimate the normal vector at the intersection point of the two backbones to determine the orientation of the planes on which they lie, using the previously described Cubic Corner (Eq. (1)).

For a given pair of vectors, the third vector must lie in a specific zone for the triplet to be a possible projection (see Fig. 6), due to the square root in Eq. (1): In the case where only two vectors are available, Stevens [17] suggested the use of the bisector of the available domain, when strongly restricted, as an approximation of the third vector. This coincides with the bisector of the two other vectors (if the corresponding α is positive), or its perpendicular (otherwise). We apply the same approach on the point where the backbones cross. In most cases, taking the bisector of the available domain as an approximation of the normal vector to the surface at that particular point acceptable results.

While the method gives a plausible orientation (for possible pairs of romantic), the user might disagree with the estimation and in such cases is allowed to draw the projection of the normal vector at the intersection of two hatching strokes that become then the backbones. Now that we have the projection of the normal vector, we have enough information to calculate the orientation of the planes of the two sets of co-planar hatching lines.

6.2 Propagation along the Backbones

Once a normal vector on the surface is known, we can extend the 3D information to the whole surface. Figure 7 shows the whole process. The first step is to determine the two planes on which the backbones lie. Using the normal vector and Eq. (1), we compute the orientation of these two planes in 3D. We then project the drawn strokes on the 3D planes. The orientation of the strokes is now fixed, and the strokes are reconstructed (Fig. 7, second from the left). From these two backbones, we will reconstruct the other hatching strokes in a similar way. For simplicity, we will describe the process for one group of hatching line.

Now that the backbone for a particular group is known, (for example, the dark gray backbone on the Fig. 7, top line). All the light gray strokes (the opposite hatching group) are reconstructed in 3D by projecting them on planes orthogo-

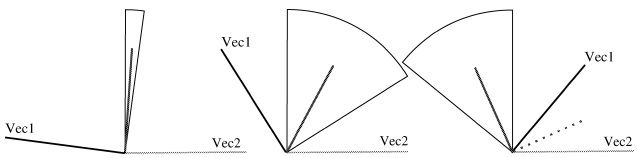


Fig. 6 For the two vectors vec1 (dark gray) and vec2 (light gray), the projected normal (middle gray) must be inside the black arc regions to form a cubic corner as shown in Fig. 2. We choose the bisector of that region to estimate the normal vector. The symmetric region with respect to the intersection point is also a possible choice, but this gives inverted orientations, similar to Necker's inversion.

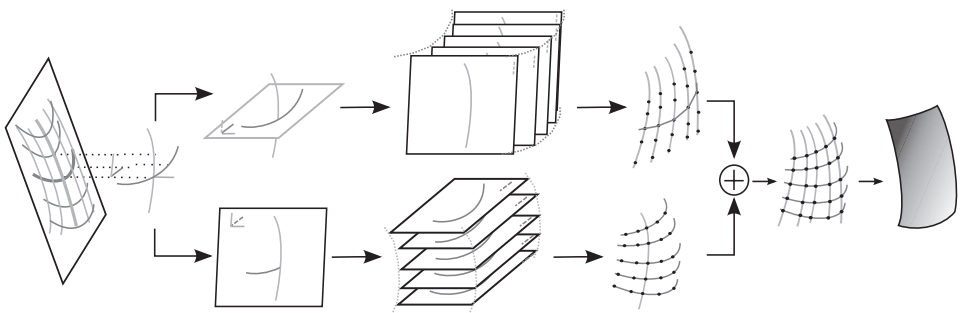


Fig. 7 For a given set of hatching lines, and from the normal vector, we reconstruct the two backbone strokes (second from the left). We then use these 3D strokes to create a series of projection planes for the other group of strokes, resulting in two groups of 3D hatching lines. Finally, we take the mean of the two groups to reconstruct the object.

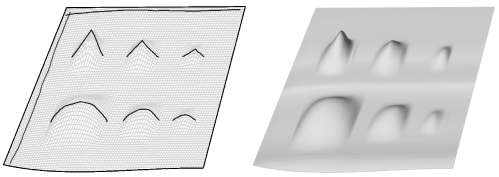


Fig. 8 Details added using the oversketching feature: from sharp to round shapes. The range of the modification depends on the length of the stroke. Parts in bold correspond to user input.

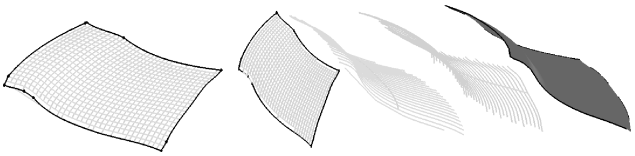


Fig. 9 Automatic generation of the hatching strokes from the edges, and its reconstruction, with the two sets of co-planar lines shown.

nal to the backbone, that is, all the strokes lie on parallel planes. Once the hatching strokes of both groups have been reconstructed, the height z of the intersections points (on the drawing plane) of the two hatching groups are averaged to produce the final shape.

7. Sketch Metaphors for Grid Edition

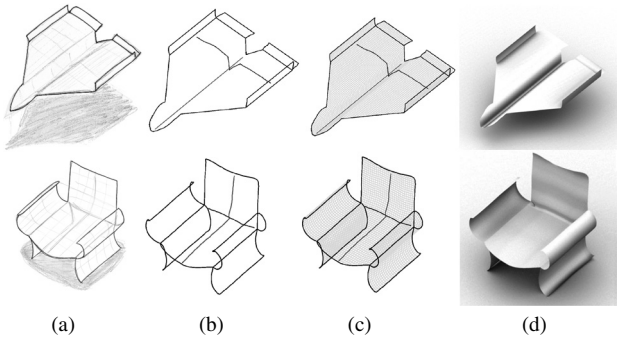
7.1 Detail Addition with Small Strokes

The previous interpolation is done as soon as there is enough information to create hatching strokes on the whole surface. The user might then want to add detail to the surface without having to draw a whole new complete profile curve. We allow then the user to draw small strokes on the surface, in a *oversketching* way. The surface then is only locally modified to match the new input. As the user 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 surface locally rather than globally. We then localize the stroke in the previously created hatching structure, and we smoothly interpolate the over-sketched stroke with the existing hatching structure: we first compute the difference between the existing hatching stroke and the newly drawn one, we then consider the neighbor strokes of the corresponding hatching group and we shift the points by the original difference weighted by the distance to the original stroke.

This allows oversketching in a natural way: the user can redraw on the strokes, and the corresponding stroke will be corrected locally, removing the need for an undo function. The resulting modeling process can then be thought as a two step operation: the first long strokes decide the global shape of the surface, then the following smaller strokes add details to the shape.

For example, Fig. 8 (left) shows the result of adding small strokes on a previously established grid, while Fig. 8 (right) shows the resulting shapes that can be added using this feature.



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

Fig. 10 A chair and a plane, from the design sketches (a) to the final models (d). (b) shows the corresponding user inputs and (c) the grids automatically generated by the system. The shading is performed using ambient occlusion.

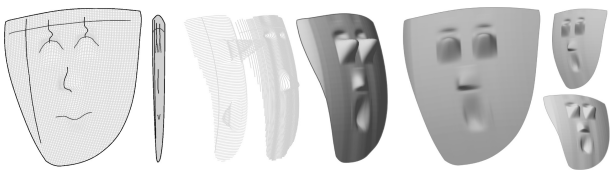


Fig. 11 An mask modeled by our system. The middle figure corresponds to the two sets of coplanar lines. User input is shown in bold.

7.2 Automatic Hatching from Edges and Stroke Copy

When a surface with one pair of two similar edges is drawn, the system suggests the use of such edges as strokes of one group of hatching lines. Moreover, for four-sided surfaces with two pairs of similar edges, the system interprets such input as distorted planes, where the hatching strokes are the same as the edges. This allows the quick modeling of various surfaces. See Fig. 9 for an example of recognition of similar strokes and the automatic generation of the hatching strokes from the edges.

Since two strokes similar on the drawing are interpreted as similar strokes in the 3D space, we also implemented a copy tool, that performs a translation of the stroke in space along the translation on the drawing plane. This allows the creation of perfectly aligned surfaces, by using the same stroke at both ends of the surface.

8. Results and Discussion

We present here some results obtained by our system. Figure 10 gives examples of modeling from existing sketches: by allowing every operation to be performed from a unique viewpoint, we were able to create the corresponding surfaces by tracing over the sketches. Figure 11 shows a mask modeled in less than two minutes (drawing and processing

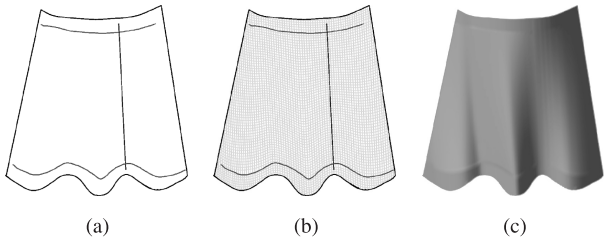


Fig. 12 A skirt, where the wavy pattern is determined by the stroke on the bottom. (a) shows the input strokes, (b) the grid automatically generated and (c) the resulting shaded (Gouraud) model.

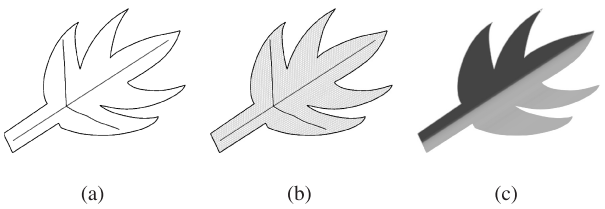


Fig. 13 A leaf modeled with three strokes. (a) Input strokes. (b) Grid. (c) Shaded model. The angle is close to 90°.

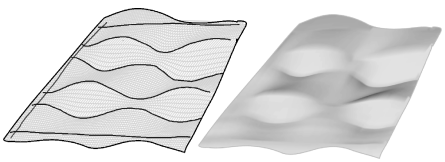


Fig. 14 A shape difficult to draw using traditional modeling software. Strokes in bold are the user input.

time included) by a trained user, using the oversketching feature to add detail. Figure 12 illustrates how three strokes are enough to model a skirt with a wavy line: this could be used to change what a model is wearing in a picture. Figure 13 shows a single leaf modeled with just two strokes to specify the inner shape, resulting in a simple shape, while the contour of the surface is complex: a silhouette-based system can not generate such surface. 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 [8], our method was meant to allow the modeling of objects in the correct orientation. More-over, the stroke needed by [8] to specify the cross-section seems to be drawn from a different viewpoint, while for our system, everything is done from the same viewpoint.

Figure 14 shows how the grid generation algorithm is able to create coherent grids from a complex input, with an interesting resulting shape. The stroke copy feature combined with the previous automatic hatching feature allows the modeling of CAD-oriented objects really fast. Two straight lines are enough to model a plane, cylinders are done with one arc and an edge (experienced users can anticipate how the normal vector will be estimated). It also allows us to model structures whose interpretation is not clear on the paper alone by specifying which strokes are coplanar during the drawing process, for example the famous Yonas's

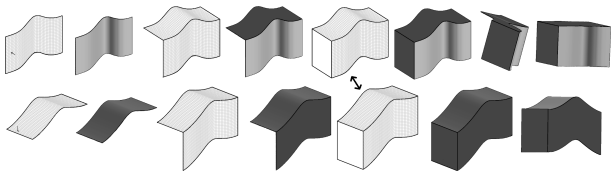


Fig. 15 Yonas's Curves: for the same input curves (in the middle, shown by the arrow), two different objects can be generated, according to the order of drawing.

curves (see Fig. 15).

The fact that we are performing all the operations from a unique viewpoint requires less operations than existing multi-view systems. For example, in order to control the shape of a surface in Fibermesh [12], one of the latest Teddy [3] improvements, the user needs to draw the outline of the object, draw another stroke on the surface, change the viewpoint, and finally drag the stroke in a local way (so two strokes, one change in viewpoint, and one or more dragging operations). In our system, we generate the surface according to the user drawn strokes, without any other operation needed (three strokes). Our system produces the desired surface with less operations.

The rules about the human visual system are merely guidelines, they do not hold under any circumstances. For surfaces close to a plane, or a cylinder, the proposed guidelines apply, and we can reconstruct the surface with confidence. In the case of a spherical surface, however, the grid that complies with the chosen rules is not natural to draw: the system is not suited for such surfaces.

Another limitation of the system is the fact that only frontal geometry is reconstructed, and self-occluding parts are impossible. As a result, models designed with this system are meant to be viewed from a viewpoint close to the one used during modeling. A system able to produce hierarchical objects with frontal and back geometry might be more suited. Our main focus was however 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.

9. Conclusion

We proposed a system able to model free-form surfaces from sketching operations performed from a unique viewpoint. The variety of the results make us believe that this system could help accelerate the current design workflow by offering to designers a quick preview of the object they are trying to put on paper, for example by showing them various shadings as well as different materials. 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. In the future, we hope to extend the system to more complex objects by allowing the modeling of the back-facing parts of the object, as well as allowing the modeling

of complex scenes that contain various objects, still from a unique viewpoint.

References

[1] D.D. Hoffman, Visual Intelligence: How We Create What We See, W.W. Norton & Company, 2000.

[2] R.C. Zeleznik, K.P. Herndon, and J.F. Hughes, "Sketch: An interface for sketching 3d scenes," SIGGRAPH '96: Proc. 23rd Annual Conference on Computer Graphics and Interactive Techniques, pp.163–170, ACM Press, New York, NY, USA, 1996.

[3] T. Igarashi, S. Matsuoka, and H. Tanaka, "Teddy: A sketching interface for 3d freeform design," SIGGRAPH '99: Proc. 26th Annual Conference on Computer Graphics and Interactive Techniques, pp.409–416, ACM Press/Addison-Wesley Publishing, New York, NY, USA, 1999.

[4] O.A. Karpenko and J.F. Hughes, "Smoothsketch: 3d free-form shapes from complex sketches," SIGGRAPH '06: ACM SIGGRAPH 2006 Papers, pp.589–598, ACM Press, New York, NY, USA, 2006.

[5] O. Karpenko, J.F. Hughes, and R. Raskar, "Free-form sketching with variational implicit surfaces," Comput. Graph. Forum, vol.21, no.3, pp.585–594, 2002.

[6] R. Schmidt, B. Wyvill, M.C. Sousa, and J.A. Jorge, "Shapeshop: Sketch-based solid modeling with blobtrees," Proc. 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling, pp.53–62, 2005.

[7] J.M. Cohen, J.F. Hughes, and R.C. Zeleznik, "Harold: A world made of drawings," NPAR '00: Proc. 1st International Symposium on Non-Photorealistic Animation and Rendering, pp.83–90, ACM Press, New York, NY, USA, 2000.

[8] J.J. Cherlin, F. Samavati, M.C. Sousa, and J.A. Jorge, "Sketch-based modeling with few strokes," SCCG '05: Proc. 21st Spring Conference on Computer Graphics, pp.137–145, ACM Press, New York, NY, USA, 2005.

[9] T. Ijiri, S. Owada, M. Okabe, and T. Igarashi, "Floral diagrams and inflorescences: Interactive flower modeling using botanical structural constraints," SIGGRAPH '05: ACM SIGGRAPH 2005 Papers, pp.720–726, ACM Press, New York, NY, USA, 2005.

[10] H. Shizuka, W. Liu, K. Kondo, and K. Matsuda, "A sketch interpreter system with shading and cross section lines by freehand drawing," Proc. 11th International Conference on Geometry and Graphics (ICGG2004), pp.357–362, 2004.

[11] A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or, "A sketch-based interface for detail-preserving mesh editing," SIGGRAPH '05: ACM SIGGRAPH 2005 Papers, pp.1142–1147, ACM Press, New York, NY, USA, 2005.

[12] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, "Fibermesh: Designing freeform surfaces with 3d curves," ACM Trans. Graph., vol.26, no.3, p.41, 2007.

[13] J. Mitani, H. Suzuki, and F. Kimura, "3d sketch: Sketch-based model reconstruction and rendering," IFIP Workshop Series on Geometric Modeling: Fundamentals and Applications, 7th Workshop GEO-7, pp.85–112, 2000.

[14] M. Prasad and A. Fitzgibbon, "Single view reconstruction of curved surfaces," CVPR '06: Proc. 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp.1345–1354, IEEE Computer Society, Washington, DC, USA, 2006.

[15] L. Zhang, G. Dugas-Phocion, J.S. Samson, and S.M. Seitz, "Single-view modelling of free-form scenes.," J. Visualization and Computer Animation, vol.13, no.4, pp.225–235, 2002.

[16] P. Company, A. Piquer, M. Contero, and F. Naya, "A survey on geometrical reconstruction as a core technology to sketch-based modeling," Comput. Graph., vol.29, no.6, pp.892–904, 2005.

[17] K. Stevens, "Implementation of a theory for inferring surface shape from contours," A.I. Memo, MIT Artificial Intelligence Laboratory, no.676, 1982.

[18] W.T. Freeman, "The generic viewpoint assumption in a framework for visual perception," Nature, vol.368, pp.542–545, April 1994.

[19] A. Girshick, V. Interrante, S. Haker, and T. Lemoine, "Line direction matters: An argument for the use of principal directions in 3d line drawings," NPAR '00: Proc. 1st International Symposium on Non-Photorealistic Animation and Rendering, pp.43–52, ACM, New York, NY, USA, 2000.

[20] A. Andre, S. Saito, and M. Nakajima, "A study on the efficiency of line renderings of 3d shapes," 6th International Conference on Cyberworlds (CW 2007), 2007, pp.257–264, IEEE Computer Society, Hannover, Oct. 2007.

[21] S. Kim, H. Hagh-Shenas, and V. Interrante, "Conveying three-dimensional shape with texture," APGV '04: Proc. 1st Symposium on Applied Perception in Graphics and Visualization, pp.119–122, ACM Press, New York, NY, USA, 2004.

[22] D. Perkins, "Cubic corners," Quarterly Progress Report, MIT Research Laboratory of Electronics, no.89, pp.207–214, 1968.

[23] R.R. Martin, P. Varley, and H. Suzuki, "Perpendicularity as a key to interpreting line drawings of engineering objects," IJCC Workshop on Digital Engineering, pp.115–120, 2005.

[24] D. Marr, Vision, Freeman, 1982.

[25] T.M. Sezgin, T. Stahovich, and R. Davis, "Sketch based interfaces: Early processing for sketch understanding," Workshop on Perceptive User Interfaces, Orlando, FL, 2001.

[26] D. Huttenlocher, D. Klanderman, and A. Rucklige, "Comparing images using the Hausdorff distance," IEEE Trans. Pattern Anal. Mach. Intell., vol.15, no.9, pp.850–863, Sept. 1993.

[27] C. Rössl and L. Kobbelt, "Line-art rendering of 3d-models," PG '00: Proc. 8th Pacific Conference on Computer Graphics and Applications, p.87, IEEE Computer Society, Washington, DC, USA, 2000.

Alexis Andre graduated from the French engineering school Supélec in 2005 while getting a M.Eng. in 2004 from the department of computer science, graduate school of information science and engineering, Tokyo Institute of Technology, Japan. He is now a Ph.D. candidate in the same department. His research interests include sketch-based modeling, line rendering and human perception.

Suguru Saito received a B.Eng. degree in 1994 and a D.Eng. degree in 1999 from Tokyo Institute of Technology, Japan. Since 2004, he has been an associate professor at the department of computer science and engineering, Tokyo Institute of Technology, Japan. His fields of interest are image processing, image reconstruction, computer graphics, color science and human perception.

Masayuki Nakajima received the B.E.E. degree and Dr.Eng. degree in 1969 and 1975, respectively from the Tokyo Institute of Technology, Japan, where he is now a professor at the department of computer science, the faculty of graduate school of information science and engineering. His fields of interest are computer graphics, pattern recognition, image processing and virtual reality.