Example-Based Curve Generation

Paul Merrell [*] Dinesh Manocha
University of North Carolina at Chapel Hill

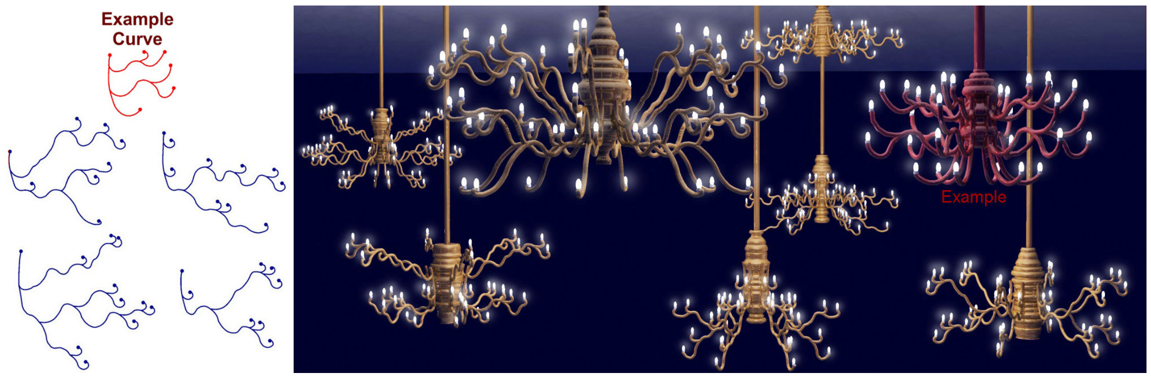

Figure 1: *Using the example curve sketched in red, several curves (blue) are generated automatically in the style of the example using our algorithm. The new curves have over 50 branches and all of them were created in less than three minutes. Four of the generated curves are shown on the left. The curves contain many branching points. The curves are directly used to create models of chandeliers by using them as generators for surfaces of revolution or lofting. The final surface is made of about 60,000 Bezier patches.*

Abstract

We present a novel synthesis method for procedural generation of complex curves. Our approach takes a simple example input curve specified by a user and automatically generates complex curves that resembles the input. The algorithm preserves many of the input shape features such as tangent directions, curvature and branch nodes. The overall system is simple and can be used to generate different kind of curved 3D models in a few minutes. We demonstrate its application to generating complex, curved models of man-made objects including furniture pieces, chandeliers, glasses, and natural patterns such as river networks and lightning bolts.

1 Introduction

One of the key problems in computer graphics is to generate geometric models of complex shapes and structures. The main goal is to generate three-dimensional content for different domains such as computer games, movies, architectural modeling, urban planning and virtual reality. In this paper, we address the problem of automatically or semi-automatically generating complex shapes with curved or non-linear boundaries. Curved artistic decorations are an important part of designing man-made objects. These include household items such as furniture, glasses, candlesticks, chandeliers, toys, etc.. Curved structures are also used in buildings and interior design. Moreover, many patterns in nature (e.g. terrain features or river network) or natural phenomena, such as lighting, also have a curved boundaries. As a result, we need simple and effective tools that can assist artists, designers, and modelers in designing elaborate curved objects and structures.

Most of the prior work in this area has been in procedural modeling, which generates 3D models from a set of rules. These include L-systems, fractals and generative modeling techniques which can generate high-quality 3D models of plants, architectural models and city scenes. However, each of these methods is mainly limited to a special class of models. Instead we use example-based techniques, which are more general and generate complex models from a simple example shape [Aliaga et al. 2008; Merrell 2007; Merrell and Manocha 2008]. Some of these methods have been inspired by texture synthesis and are currently limited to complex polyhedral models or layouts of city streets.

Main Result: In this paper, we present a new curve synthesis algorithm. Our algorithm accepts a simple 2D piecewise curve as an input and generates a complex curve which has similar features. Our approach is general and makes no assumptions about parametric continuity or branching nodes. We perform local shape analysis based on the tangent vectors and curvature of the input curve, and generate output curves that preserve these local features. Given a complex curve, we perform an extrusion operation or use them as generators for surface of revolution to generate 3D models of curved objects.

The overall algorithm is relatively simple, efficient and quite robust in practice. Our system also provides the capability to edit the input curve and to create closed loops. We have used it to generate complex 3D models of chandeliers, drinking glasses, candlesticks, river networks, lightning bolts, and cabinet handles with hundreds of curve segments or surface patches in a few minutes.

2 Related Work

A few techniques have been developed to transform curves sketched in one particular artistic style into a different artistic style [Hertzmann et al. 2002; Freeman et al. 2003]. The artistic styles are determined automatically from the example curves sketched by the user. Similar techniques have also been applied on meshes to transform the models [Bhat et al. 2004; Zelinka and Garland 2003] and also to tranform space-time curves for animation [Wu et al. 2008]. Simhon and Dudok [2004] use a hidden Markov model to add artistic details to sketches.

Example-based techniques are widely used to synthesize texture [Efros and Leung 1999; Kwatra et al. 2003] and similar techniques has been applied to vector data [Barla et al. 2006]. Three-dimensional closed polyhedral models can also be synthesized from

[*] http://gamma.cs.unc.edu/synthesis

example models [Merrell 2007; Merrell and Manocha 2008]. An example-based method has also been presented to synthesize the layout of city streets [Aliaga et al. 2008].

Procedural modeling techniques are widely used to generate different types of objects. Many of these methods are designed to model urban environments [Müller et al. 2006; Wonka et al. 2003]. Many elegant algorithms have also been developed for modeling plants using L-systems [Měch and Prusinkiewicz 1996; Ijiri et al. 2005; Power et al. 1999]. Wong et al. [1998] have developed a procedural techniques for designing decorative patterns, including floral patterns. Pottmann et al. [2007] have presented elegant algorithms to generate freeform shapes for architectural models.

Sketch-based interfaces have been developed as an intuitive way to model and deform meshes [Igarashi et al. 1999; Singh and Fiume 1998]. These methods complement our own work and can be used to transform 2D curves into full 3D models.

3 Curve Generation

3.1 Overview

We assume that the input example curve is represented as a piecewise parametric curve $\{\mathbf{c}_1(t), \mathbf{c}_2(t), \ldots\}$, where $\mathbf{c}_i(t)$ is a 2D Bezier curve. However, our algorithm can handle any curve representation as long as it can be subdivided and we can evaluate bounds on its tangents and curvatures. The curves may or may not contain closed loops, branches, and cusps. Our algorithm generates an output curve that resembles the input curve and is composed of segments from the input example curve $s_i(t)$. These segments are created in a process described in Section 3.2. The curves and the segments start at the value $t = 0$ and end at the value $t = 1$.

Our goal is not only to generate simple curves, but also to generate curves with multiple branches and loops. We use a graph data structure to represent how the parts of the curve connect. Each of the graph's edges is a sequence of the curve segments $s_i(t)$. Different versions of the new graph are constantly being generated and evaluated. Each version is evaluated and given a score based on how well it adjusts to changes the user makes and how well it forms closed loops if the example curve contains loops. The user can edit the graph by creating and moving adjustment points on the curves. The graph with the best current score is displayed even while the user is moving adjustment points. Section 3.4 describes how the sequence of segments $s_i(t)$ for the graph's edges are modified and evaluated. Even the best sequences of segments rarely produce loops that close exactly or curves that exactly touch the user's adjustment points. In order to accomplish this, the segments are bent and stretched as described in Section 3.3.

3.2 Creating and Connecting Segments

For the new curve to resemble the example curve, every local neighborhood of the new curve should closely resemble a neighborhood of the input example curve. We characterize a curve's local structure in terms of its curvature and tangent. The output curve is created by piecing together parts of the example curve which have roughly the same tangent angle $\theta = \tan^{-1} \frac{y'}{x'}$ and signed curvature $k = \frac{x'y''-y'x''}{(x'^2+y'^2)^{3/2}}$. Our algorithm two curves can be connected if their tangent vectors at the common end point are within an angle of about θ_b and their curvatures are within about k_b. As part of precomputation, we discretize the tangent angles and curvatures into uniform bins $\hat{\theta} = \lfloor \frac{\theta}{\theta_b} \rfloor$ and $\hat{k} = \lfloor \frac{k}{k_b} \rfloor$.

Given an input curve, we subdivide it into curve segments s_i. Each

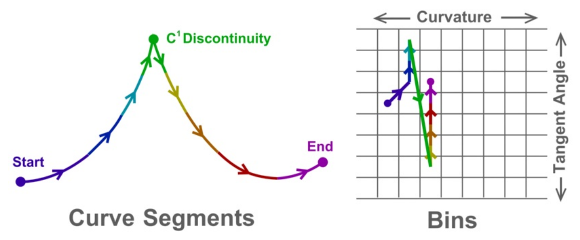

Figure 2: *The curve is decomposed into segments which have only small changes to their tangents and curvatures. The segments are placed into bins with bounds on tangent angles and curvatures.*

segment s_i has a starting and ending tangent angle, $\hat{\theta}_i^s, \hat{\theta}_i^e$, and a starting and ending curvature $\hat{k}_i^s, \hat{k}_i^e$. Smooth segments are subdivided until their start and end are only one bin apart $|\hat{k}_i^s - \hat{k}_i^e| \leq 1$ and $|\hat{\theta}_i^s - \hat{\theta}_i^e| \leq 1$ or $|\hat{\theta}_i^s - \hat{\theta}_i^e| = \lfloor \frac{\pi}{\theta_b} \rfloor - 1$. Moreover, we ensure that the tangent angles and curvatures of all points on each segment are contained with the starting and ending bin. However, since we do not assume C^1 or C^2 continuity, there may be points of discontinuity i.e. cusps. Every cusp is followed by and preceded by an open interval having C^2 continuity. We subdivide this open interval into curved segments until the start and end of each segment are only one bin apart. Wherever a cusp appears we combine it with the curve segment immediately before and immediately after it into a single unit. This means that even though the cusp itself does not have a well-defined tangent, it is integrated into a segment s_i whose beginning and end have well-defined tangents $\hat{\theta}_i^s, \hat{\theta}_i^e$. If a segment has a cusp, usually its start and end are more than one bin apart.

We place all the curve segments cut from the example curve into a 2D array of bins shown in Figure 2. The start of a segment s_j can be attached to the segment s_i if their endpoints are in the same bin $(\hat{\theta}_j^s, \hat{k}_j^s) = (\hat{\theta}_i^e, \hat{k}_i^e)$. Let a_i be the set of segments that may follow after s_i, $a_i = \{s_j | (\hat{\theta}_j^s, \hat{k}_j^s) = (\hat{\theta}_i^e, \hat{k}_i^e)\}$ and let b_i be the set of segments that may come before s_i. If for any i and j, $a_i = \{s_j\}$ and $b_j = \{s_i\}$, then the segment s_j is always attached to the end of s_i. These segments can be combined into a single segment s_m where $(\hat{\theta}_m^s, \hat{k}_m^s) = (\hat{\theta}_i^s, \hat{k}_i^s)$ and $(\hat{\theta}_m^e, \hat{k}_m^e) = (\hat{\theta}_j^e, \hat{k}_j^e)$. Combining segments that are always attached together improves the overall efficiency of the algorithm since there are fewer segments to manage.

There are special cases to consider within the example curve. The input curves could start and end at the same location $\mathbf{c}(0) = \mathbf{c}(1)$ to form a closed loop. In this case, there is nothing unusual about the segments at the start or end of the curve. Each segment has a segment before it and after it. But if $\mathbf{c}(0) \neq \mathbf{c}(1)$, the curve could start or end by branching off another curve or it could start or end in empty space. In order to handle these special cases, we include special symbols in the sets a_i and b_i to indicate that before s_i there is a branching point or empty space.

3.3 Bending and Stretching the Curves

Our algorithm uses a graph to represent the topology of the output curve. The graph has vertices that denote the special cases of branching points, starting points, and ending points as well as any point the user selects as an adjustment point. The edges of the graph are sequences of curved segments. For each edge, let us combine all the segments s_i of that edge into a single piecewise parametric curve $\mathbf{s}(t)$ (i.e. the composite curve). Let $\mathbf{s}(0)$ and $\mathbf{s}(1)$ be the start point and end point of the composite curve. Moreover, let $\mathbf{s}(t)$ be an arbitrary point on the curve. We also estimate the arc length of $\mathbf{s}(t)$ using a piecewise linear approximation and compute an arc-

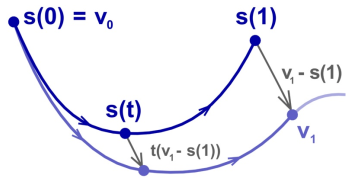

Figure 3: *The points along a parametric curve $\mathbf{s}(t)$ are bent and stretched so that the curve goes from the vertex $\mathbf{v}_0$ to $\mathbf{v}_1$.*

length parametrization of $\mathbf{s}(t)$. In the rest of the paper, we assume that all the computed curves are represented with an approximate arc-length parameterization.

Let $\mathbf{v}_0$ and $\mathbf{v}_1$ be two vertices of the graph that the curve segments are supposed to go between. We position the curve $\mathbf{s}(t)$ so that $\mathbf{s}(0) = \mathbf{v}_0$. Ideally, $\mathbf{s}(1) - \mathbf{s}(0) = \mathbf{v}_1 - \mathbf{v}_0$, but otherwise we must stretch the curve to get it to reach to $\mathbf{v}_1$ by performing a simple geometric deformation. We stretch the curve over its entire length as shown in Figure 3. We move the point $\mathbf{s}(1)$ by $\mathbf{v}_1 - \mathbf{s}(1)$ and every intermediate point $\mathbf{s}(t)$ by $t(\mathbf{v}_1 - \mathbf{s}(1))$. Long curves are distorted less than short curves when stretched a given distance since the stretching is distributed over a greater length. Our goal is to keep the extent that a curve stretches small especially for short curves.

We now have a simple formulation to stretch a curve given its two endpoints. The endpoints are the vertices of the graph. The locations of some of the vertices may be specified by the user, but any remaining vertices are treated as free variables in the plane. The goal is to determine the optimal vertex locations to minimize the extent of the stretching. The overall extent of the stretching or deformation should be minimized because it can distort the segments s_i found in the example curve. The extent of the stretching can be represented by a system of linear equations. Each edge provide two linear equations, one for the x and one for the y components. The edge connecting $\mathbf{v}_0$ to $\mathbf{v}_1$ provides the equation $\mathbf{s}(1) - \mathbf{s}(0) = \mathbf{v}_1 - \mathbf{v}_0$. Since the equations are all linear, we can easily minimize the amount of stretching for each edge in the least squares sense. It is best to use a weighted least squares optimization that weights each equation according to the reciprocal of the length of each curve and thereby, ensures that short curve segments stretch less than the long segments.

3.4 Choosing the Sequence of Curve Segments

Our method is designed to constantly refine and improve the current solution. Anytime the user is not physically moving an adjustment point, our algorithm performs random modifications to the structure of the graph and to the sequence of segments along the graph's edges. Most of these modifications do not improve the result, but by rapidly testing many of them, there are likely to be a few of them that do.

First, we select a portion of the graph to modify. We randomly select a point on the graph and construct a circle around it. Within the circle, we remove every curve segment as shown in Figure 4. The segments removed from the circle were connected to other segments from outside the circle. The segments outside the circle must be reconnected in one of three ways. First, they could be reconnected to another cut segment. This option is preferable since it reconnects two cut segments simultaneously. Second, they could become branches of an existing curve. Certain types of segments can have branches attached to them. If such a segment does not already have a branch attached to it, we can attach one. The third option is to connect the cut segment to the special type of segment that can start or end the curve as described in Section 3.2. This creates a leaf-vertex, a vertex incident to a single edge, in the graph. The first two options have a specific target location where the curve must attach to either a branch point or a cut segment. However, the third option does not have a target location because the curve could start or end anywhere. As a result, the third option is relatively easy to satisfy, but is not necessarily the best option.

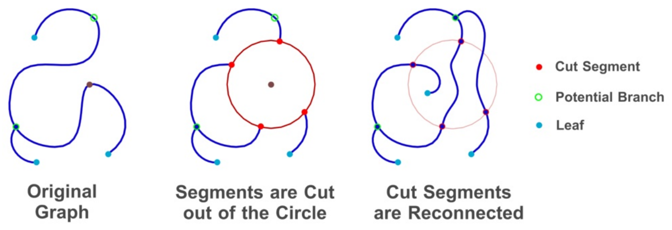
Cut Segment
Potential Branch
Leaf
Original Graph
Segments are Cut out of the Circle
Cut Segments are Reconnected

Figure 4: *The graph is modified by removing segments from part of the graph and then reconnecting them.*

There may be several target locations that the cut segments can connect to. We start from one cut segment and connect it to a sequence of segments. The segment s_j can only follow s_i in the sequence if $s_j \in a_i$ as described in Section 3.2. When a_i contains several options, we randomly choose between them. This is a type of random walk. The segments are eventually connected to whichever target location, a branching point or a cut segment, they pass the closest to. However, not only does the end of the segments need to be close to the target, but it also needs to have the appropriate tangent angle and curvature when it reaches the target. Let $(\hat{\theta}^s_t, \hat{k}^s_t)$ be the target segment's tangent angle and curvature. As the sequence is being generated, we need to ensure that it ends at the target's tangent angle and curvature. Let $(\hat{\theta}^e_u, \hat{k}^e_u)$ be the end of the unfinished sequence that the algorithm is currently generating. We can compute the shortest sequence of segments that will take us between these given two tangent angles and curvatures using Dijkstra's algorithm. This gives us the shortest curve that we could attach to our unfinished curve to finish it with the correct tangent angle and curvature. We attach it to the end of our unfinished curve and compute the distance between the end of the attached curve and the target. We perform this distance computation everytime a segment is added to our random walk and for every target. Whichever target the random walk gets the closest to is the target the cut segment connects to. If we do not get close to any of the targets, then the cut segment connects to a leaf-vertex.

We connect all of the cut segments to either another cut segment, a branch, or a leaf. Next, the curves are stretched as needed as described in Section 3.3. Once all these changes have been made to the graph, they must be evaluated. The new modified graph is compared to the graph before the modifications were made based upon several criteria. Curves that are stretched more, get more distorted and so a stretch cost is computed as the sum of the distances that each curve is stretched $||\mathbf{c}(1) - \mathbf{v}_1||$. This stretch cost has some unintended consequences. It penalizes graphs with cycles and promotes graphs with leaf-vertices. When a cut segment is attached to a leaf-vertex, the curve does not need to be stretched since the leaf-vertex may be placed anywhere. But when the cut segment is attached to a loop the curve must stretch. The cycles of the graph may all be removed unless we compensate for them by adding a cost for each leaf-vertex and subtracting a cost for each cycle. We also may want to encourage branching by removing a cost for each branch. It is possible that parts of the graph may become disjoint or that parts may self-intersect. In order to discourage this, heavy penalties are added for each disjoint part and for each self-intersection.

	Num. Output	Input Seg.	Output Seg.	Input Branch	Output Branch
Chandelier	8	105	5,686	4	56
Cabinet	9	38	9,787	0	0
Streams	5	203	9,900	5	82
Lightning	4	633	12,509	3	149
Glasses	13	42	2,749	0	0
Candlestick	8	53	1,080	0	0

Table 1: *Table of the number of output curves generated, input segments, output segments, input branches, and output branches.*

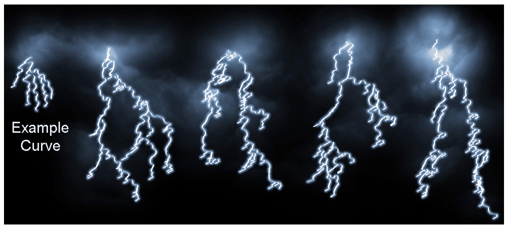

Figure 5: *From a sketch of a few bolts of lightning, several more complex lightning patterns are generated in under two minutes. The output curves contain about 150 branches and over 12,000 segments.*

The process of modifying and evaluating the graph can be accomplished quickly. This means that the curves will respond immediately to the user's changes. The shortest path between any $(\hat{\theta}_t^s, \hat{k}_t^s)$ and $(\hat{\theta}_u^s, \hat{k}_u^s)$ can be precomputed. The stretching cost is computed by summing up one vector for each segment. The number of cycles, branches, leaves, and disjoint subsets can be easily counted. Counting the number of self-intersection among the segments is the most expensive computation in this algorithm. As a result, we only perform that test after we have confirmed that the new graph is a good candidate based on other criteria.

4 Results and Analysis

Our method was tested on sketches of several different kinds of natural and man-made objects. It was used to design the branches of many chandeliers in Figure 1. Automatically generated curves were used to design candlesticks (Figure 8) and drinking glasses (Figure 9) by revolving the curves around an axis. It was also used to model fancy cabinet handles in Figure 7, bolts of lightning Figure 5, and river systems Figure 6. The output curve computed by our system is a piecewise Bezier curve. Typically, a single cubic Bezier curves is used to represent each curve segment.

Analysis: The final shapes produced by our algorithm depend on the example curve and the adjustment points added by the user as

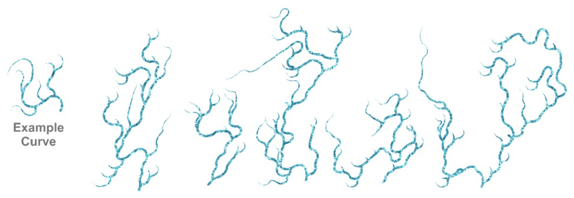

Figure 6: *From a sketch of a river system, several larger and more complex river systems are automatically generated in under two minutes. The output curves each have an average of about 80 branches and 10,000 segments.*

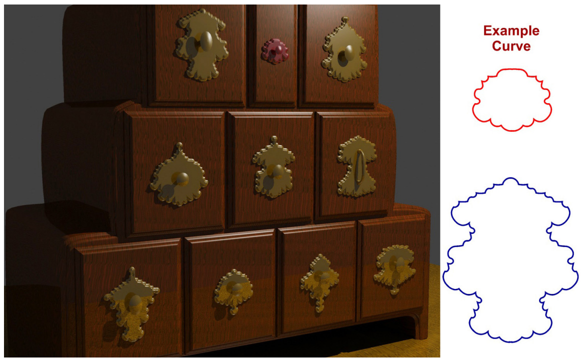

Figure 7: *From a simple sketch of the handle of a cabinet, several complex cabinet handles are designed automatically in the same style. The example handle is the top center red-tinted handle.*

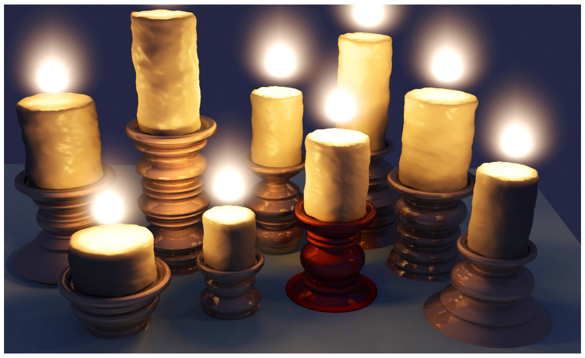

Figure 8: *The profile of a candlestick is sketched and used to design many similar candlesticks in under one minute. The curves are revolved around an axis to produce these candlesticks. The example candlestick is tinted red.*

part of the editing. Our underlying algorithm ensures that the local features of the final curve, in terms of tangent vectors and curvature, are similar to that of the input example curve. However, the basic stretching deformation can change those. The running time of the algorithm varies as a function of number input segments, curve degree and the number of branch points in the final curve. In practice, our algorithm is very fast and can generate new curves at interactive rates on a desktop PC for curves with a lot of segments.

Comparison: Much of the prior work in procedural modeling has focused on modeling plants using L-systems. Our method has rules similar to L-systems. For example, the acceptable sequences of segments described in Section 3.2 could be generated using a complex L-system. Overall, we expect that prior algorithms based on L-systems would generate higher quality models of plants and trees and our algorithm is slightly faster. The main benefit of our approach is in generating models of curved man-made objects (e.g. decorative objects) such as furniture and household items. We are not aware of any prior procedural methods for such curved objects. As a result, tools that are general and that can model a wide variety of objects are highly valuable. Our method can model a wide variety of objects because it is example-based. The user can easily switch between creating two very disimiliar types of objects by sketching a new example curve.

Limitations: As explained in Section 3.4, the curves are generated by making small incremental changes and testing if the changes improve a cost function. In some ways, our algorithm is performing an optimization in the design space and there is a risk that the cur-

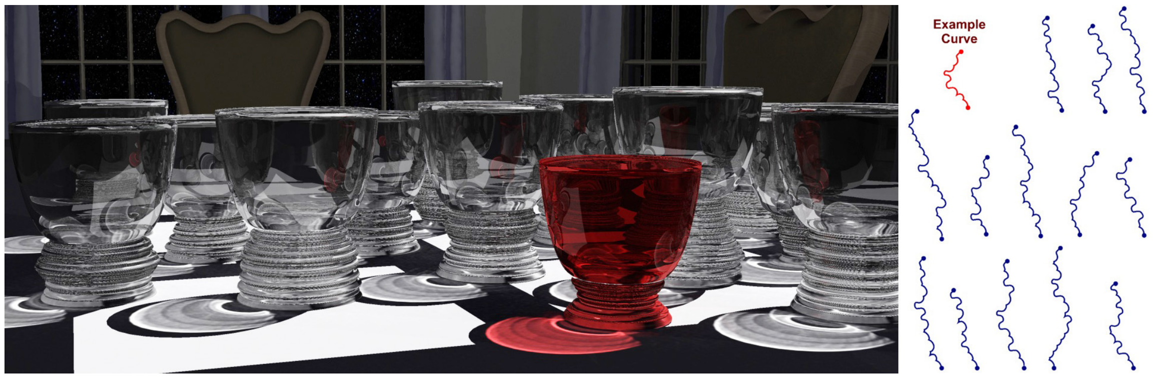

Figure 9: *The profile of a drinking glass is sketched (red curve) and used to design many glasses in a similar style (blue curves). The glasses are modeled by revolving the curves around an axis. The surface of revolution obtained from the example input curve is shown as red tinted glass.*

rent solution may fall within a local minima of the cost function meaning that small incremental changes do not improve the solution, only large changes do. The shape of the final object depends on the input curve and where the user moves the adjustment points. In some cases, our approach can result in unnatural shapes. Our algorithm can produce self-intersections, and the algorithm needs to explicitly check for that.

5 Conclusion and Future Work

We have presented a method for taking curves sketched by a user and automatically generating new curves that resemble the input shape. Parts of the new curve closely resemble parts of the example curve because they have the same geometric characterization including tangent vectors, curvature and branches. The example curve is divided into segments. These segments are bent and stretched and rearranged to fit a cost function. The cost function gives low scores to curve that follow the user's control points, that do not self-intersect, and that have many branches. We have applied the algorithm to different input curves and used to generate curved models of different man-made objects and some natural patterns.

There are many avenues for future work. We can improve the user interface and use more sophisticated physically-based deformation algorithms for stretching the curves. We would use our algorithm to generate more kind of models include architectural structures and outdoor scenes. The set of input curves can also include subdivision curves, and our approach can be extended to generate non-planar 3D curves as well as 4D space-time curves for animation. Finally, we would like to extend our approach to procedurally generate generate 3D surface models composed of freeform surfaces.

References

ALIAGA, D. G., VANEGAS, C. A., AND BENEŠ, B. 2008. Interactive example-based urban layout synthesis. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, ACM, New York, NY, USA, 1–10.

BARLA, P., BRESLAV, S., THOLLOT, J., SILLION, F. X., AND MARKOSIAN, L. 2006. Stroke Pattern Analysis and Synthesis. 663–671.

BHAT, P., INGRAM, S., AND TURK, G. 2004. Geometric texture synthesis by example. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, ACM Press, New York, NY, USA, 41–44.

CHEN, X., NEUBERT, B., XU, Y.-Q., DEUSSEN, O., AND KANG, S. B. 2008. Sketch-based tree modeling using markov random field. *ACM Trans. Graph. 27*, 5, 1–9.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, 1033–1038.

FREEMAN, W. T., TENENBAUM, J. B., AND PASZTOR, E. C. 2003. Learning style translation for the lines of a drawing. *ACM Trans. Graph. 22*, 1, 33–46.

HERTZMANN, A., OLIVER, N., CURLESS, B., AND SEITZ, S. M. 2002. Curve analogies. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 233–246.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: a sketching interface for 3d freeform design. In *Proc. Of ACM SIGGRAPH '99*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 409–416.

IJIRI, T., OWADA, S., OKABE, M., AND IGARASHI, T. 2005. Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. In *Proc. Of ACM SIGGRAPH '05*, ACM Press, New York, NY, USA, 720–726.

KWATRA, V., SCHDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *Proc. Of ACM SIGGRAPH '03*, 277–286.

MERRELL, P., AND MANOCHA, D. 2008. Continuous model synthesis. *ACM Trans. Graph. 27*, 5, 1–7.

MERRELL, P. 2007. Example-based model synthesis. In *I3D '07: Symposium on Interactive 3D graphics and games*, ACM Press, 105–112.

MĚCH, R., AND PRUSINKIEWICZ, P. 1996. Visual models of plants interacting with their environment. In *Proc. Of ACM SIGGRAPH '96*, 397–410.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND GOOL, L. V. 2006. Procedural modeling of buildings. *ACM Trans. Graph. 25*, 3, 614–623.

POTTMANN, H., LIU, Y., WALLNER, J., BOBENKO, A., AND WANG, W. 2007. Geometry of multi-layer freeform structures for architecture. In *Proc. of ACM SIGGRAPH '07*.

POWER, J. L., BRUSH, A. J. B., PRUSINKIEWICZ, P., AND SALESIN, D. H. 1999. Interactive arrangement of botanical l-system models. In *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 175–182.

SIMHON, S., AND DUDEK, G. 2004. Sketch Interpretation and Refinement Using Statistical Models. In *Proceedings of Eurographics Symposium on Rendering 2004 (Norrköping, Sweden, June 21–23)*, EUROGRAPHICS Association, A. Keller and H. W. Jensen, Eds., 23–32, 406.

SINGH, K., AND FIUME, E. 1998. Wires: a geometric deformation technique. In *Proc. Of ACM SIGGRAPH '98*, ACM, New York, NY, USA, 405–414.

WONG, M. T., ZONGKER, D. E., AND SALESIN, D. H. 1998. Computer-generated floral ornament. In *Proc. Of ACM SIGGRAPH '98*, ACM, New York, NY, USA, 423–434.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. In *Proc. Of ACM SIGGRAPH '03*, 669–677.

WU, Y., ZHANG, H., SONG, C., AND BAO, H. 2008. Space-time curve analogies for motion editing. In *GMP*, 437–449.

ZELINKA, S., AND GARLAND, M. 2003. Mesh modelling with curve analogies. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Sketches & Applications*, ACM, New York, NY, USA, 1–1.