Computers & Graphics 34 (2010) 304–311

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Example-based curve synthesis

Paul Merrell *, Dinesh Manocha

University of North Carolina at Chapel Hill, USA

ARTICLE INFO

Keywords:
Procedural modeling
Model synthesis
Curve synthesis

ABSTRACT

We present a novel synthesis algorithm for procedurally generating complex curves. Our approach takes a simple example input curve specified by a user and automatically generates complex sets of curves that resemble the input. The algorithm preserves many of the local shape features of the input curves such as tangent directions, curvature, branch nodes, and closed loops. The overall approach is simple and can be used to generate varied curved 3D models in a few minutes. We demonstrate its application by 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 3D geometric content for many domains such as computer games, movies, architectural models, urban planning and virtual reality. In this paper, we address the problem of automatically or semi-automatically generating complex shapes with curved boundaries. Curved artistic decorations with repeated patterns are an important part of the design of 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 natural patterns (e.g. terrain features or river network) and natural phenomena, such as lighting, also have curved boundaries with random variation. 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 with repeated patterns 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, our goal is to use example-based techniques which are more general and can generate complex models from a simple example shape [1–4]. Some of these model synthesis methods have been inspired by texture synthesis, but the current methods are limited primarily to complex polyhedral models with planar boundaries or layouts of city streets.

Main result: In this paper, we present a new method for rapidly generating many sets of curves based on an example. Our algorithm accepts a set of 2D curves as an input and rapidly generates many more complex curves in a similar style. Our method is primarily designed to capture the local structure of the example curve based on local shape characteristics such as tangent and curvature, and not the global structure. As a result, it is better suited for generating complex curves, which have a random layout.

Our approach is general and makes no assumptions about the shape or smoothness of the input curves. We perform local shape analysis based on tangent vectors and curvature of the input curve, and generate output curves that tend to preserve these local features including cusps, branches, and closed loops. Furthermore, we explicitly check for self-intersections between the curve segments. We use a graph to maintain the topology of various curve segments and present automatic methods to incrementally refine the structure of the graph to generate the final curves. We also present an intuitive metric to evaluate the quality of the results. Given a set of generated curves, we perform an extrusion operation or use the final curves as generators for surfaces of revolution to generate 3D models of curved objects.

The overall algorithm is relatively simple, efficient and quite robust in practice. Our system allows users to edit the generated curves interactively. We use our algorithm to generate complex 3D models of chandeliers, drinking glasses, candlesticks, river networks, lightning bolts, and cabinet handles with hundreds or thousands of curve segments or surface patches in a few minutes.

The rest of the paper is organized as follows. In Section 2, we discuss related work on procedural modeling and curve generation. In Section 3, we explain our curve generation algorithm. We show results and analyze our method in Section 4. We compare

* Corresponding author.
E-mail address: pmerrell@cs.unc.edu (P. Merrell).
URL: http://gamma.cs.unc.edu/synthesis (P. Merrell).

0097-8493/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2010.05.006

our approach to other methods in Section 5 and discuss ideas for future research in Section 6.

2. Related work

A few techniques have been developed to transform curves sketched in one particular artistic style into a different artistic style [5,6]. The artistic styles are determined automatically from the example curves sketched by the user. Similar techniques have also been applied to meshes to transform the models [7,8] and also to transform space–time curves for animation [9]. Simhon and Dudok [10] use a hidden Markov model to add artistic details to sketches. These techniques rely on the user to specify the large-scale structure of the curve or shape to be generated, but do not address the problem of generating many curves of a particular style with a varied large-scale structure.

Layouts of city streets have been created using an interactive tool which uses tensor fields [11]. Kalnins et al. [12] use a non-photorealistic rendering technique to draws 3D models in different artistic styles. It generates curves representing the strokes an artist might draw or paint in a particular style.

Example-based techniques are widely used to synthesize texture [13,14]. Similar techniques have been applied to vector data to generate strokes in a particular style [15,16]. Two-dimensional arrangements of elements can be created from an example [17]. Three-dimensional closed polyhedral models can also be synthesized from example models [2–4]. However, example-based techniques have only been applied to curved models in specific cases such as the layout of city streets [1].

Procedural modeling techniques are widely used to generate different types of objects including urban environments [18,19] and plants using L-systems [20–22]. Pottmann et al. [23] have presented elegant algorithms to generate freeform shapes for architectural models. Wong et al. [24] have developed a procedural technique for designing floral ornamentation. These techniques produce compelling models, but requires significant effort from the user to control and cannot be modified interactively.

Sketch-based interfaces have been developed as an intuitive way to model and deform meshes [25,26]. These methods complement our approach and can be used to transform 2D curves into full 3D models.

3. Curve generation

In this section, we present our curve synthesis algorithm. We first give an overview of our method, then we describe each stage of the algorithm in more detail. We describe how to decompose the input into a set of small segments which are connected and stretched into larger curves. These curves are modified stochastically and evaluated based on how closely they resemble the original curves and if they avoid self-intersections.

3.1. Overview

The user inputs an example curve as a set of parametric curves. The curves may or may not contain closed loops, branches, and cusps. Our algorithm generates a set of output curves that resemble the input curves.

We first subdivide the input curve into smaller parts called *curve segments* as described in Section 3.2. The output curves are formed as an ordered sequence of segments. Two curve segments can connect together if their tangents and curvatures are close to each other. We allow the user to control the shape of the curves interactively by selecting points on the curves and repositioning them. In order for those points to reach their specified positions, the curves are stretched as described in Section 3.3. But the curves should not be stretched very far, since this causes them to resemble the original curve less. To prevent the curves from stretching too far, it is necessary to modify the sequence of segments. We replace segments in the sequence with new segments to decrease the amount of stretching.

Minimizing the amount of stretching is one of several goals we have for the set of curves. It is also important that the curves do not intersect themselves and do not separate into disjoint parts. We evaluate the set of curves by determining how far the curves are stretched and how many branches, self-intersections, and disjoint parts they contain. The curves are modified stochastically as described in Section 3.4 and are evaluated as described in Section 3.5. Each curve modification is only kept if it decreases the amount of stretching. Changes to the curves can occur rapidly and they are displayed to the user in real-time as part of our interactive interface.

Since our goal is to generate sets of curves which may contain multiple branches and loops, we use a graph data structure to represent the set of curves (see Section 3.4). The vertices of the graph are the curves' endpoints and branching points.

3.2. Creating and connecting segments

This section discusses how we subdivide the input curves into curve segments and then combine them together. The curve segments are the basic building blocks of the output curves. The input is a set of parametric curves $\{\mathbf{c}_1(t), \mathbf{c}_2(t), \ldots\}$, where each curve $\mathbf{c}_i(t)$ starts at $t=0$ and ends at $t=1$. We use 2D Bezier curves in practice, but our algorithm can handle any curve representation as long as we can subdivide the curves and evaluate bounds on their tangents and curvatures.

Our method ensures that the new curves resemble the example curves in a similar way to many texture synthesis algorithms, which is to force every local neighborhood of the new curve to resemble a neighborhood of the example curve. A curve's local neighborhood is characterized by its tangent angle $\theta = \text{atan2}(y', x')$ and signed curvature $k = (x'y'' - y'x'')/(x'^2 + y'^2)^3/2$. The tangent angles and curvatures are discretized into uniform bins $\hat{\theta} = \lfloor \theta/\theta_b \rfloor$ and $\hat{k} = \lfloor k/k_b \rfloor$ where θ_b and k_b are the size of the bins. In our algorithm, two curve segments can be connected if they have tangent angles and curvatures inside the same bin.

Each 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 they start and end only one bin apart. The tangents and curvature are not defined at cusps. All this means is that we never divide the curve at a cusp. The cusp is always contained inside one of the segments. It is always combined with the part of the curve immediately before and the part immediately after it into a single segment. The cusp is integrated into a segment s_i with well-defined beginning and ending tangents $\hat{\theta}_i^s, \hat{\theta}_i^e$. Algorithm 1 gives a pseudo-code description of how the curves are subdivided.

Algorithm 1. Method for subdividing a piecewise smooth curve into segments.

subdividePiecewise($\{\mathbf{c}_1(t), \mathbf{c}_2(t), \ldots, \mathbf{c}_n(t)\}$)
c is a piecewise smooth curve composed of the curves $\mathbf{c}_1(t), \mathbf{c}_2(t), \ldots$ which may intersect at cusps.

```
1: for i = 1 to n do
2:     subdivide(c_i,0,1)
3: end for
```