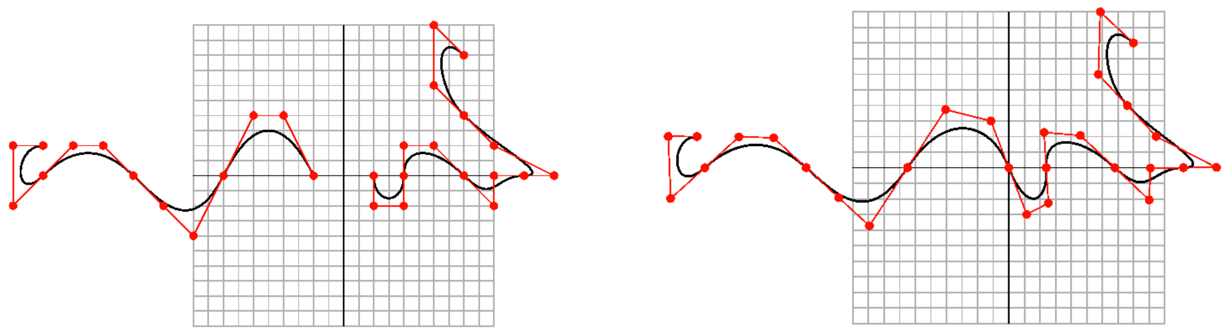
Vector Texture Synthesis

Papoj Thamjaroenporn pt2277
Fall Project Report, 2012
Prof. Eitan Grinspun, Prof. Changxi Zheng



Overview

In this project, we consider an alternative way to generate texture: using vector representation of the exemplar instead of pixel representation. In this report, we focus on the first step accomplished in the texture synthesis process: smooth curve connection based on minimization of curvature change constraint. My main contribution to this project includes: a solid OpenGL framework for Poly-Bezier curve display, a curve evaluation algorithm that calculates Bernstein coefficients efficiently, and an algorithm that robustly connects two generic Poly-Bezier curves together smoothly, while preserving the curvature of the original curves.

1. Introduction

In the texture synthesis literature, much work has been done on the synthesis using pixel-based exemplar. A few works incorporate the notion of splines into the texture synthesis. However, the works often consider the splines as separate elements instead of connectable curves, as seen in Discrete Element [Ma et al. 2011]. Approaching the texture synthesis in this manner is still similar to pixel-based texture synthesis, since a graph of separate elements can be generated, and traditional neighbor searching algorithm can be employed when generating new texture patch [Barla et al. 2006, Ma et al. 2011]. This project aims to explore the texture synthesis that consider the curves in the exemplar as connectable elements. This report specifically focuses on a main algorithm implemented for the project: smooth curve connection under curvature preservation constraint.

2. Project Timeline

Month	Progress
Sept 2012	• Project setup • Brainstorming • Reading and discussing related papers.
Oct 2012	• Derivation of a formula for curve connection constraints in a simple case. • Writing test scripts in Python
Nov 2012	• Derivation of a generalized formula for curve connection • Developing OpenGL framework in C++ • Implementing an efficient Bezier curve evaluation algorithm
Dec 2012	• Implementing the generalized optimization formula

3. Compiling and Program Control

The source code only depends on OpenGL framework and Eigen library, which is included in the package. To compile the code, simply `make`, then run `./vtsynth`. The user can change test cases, as instructed in `main.cpp`, to run different hard-coded test cases for different results.

The program control is the following:

Keyboard Control	Command
q	Quit
c , C	Recenter by covering all objects in the scene
[,]	Increase/decrease the size of the grid
G	Toggle the grid on/off
+ , -	Zoom in/out
b , B	Toggle curve render on/off
h , H	Toggle curve hull render on/off
A	Connect two curves presented in the scene

Mouse Control	Command
SHIFT + Right C	Zoom in/out
SHIFT + Left C	Panning

4. Concepts and Implementation

- OpenGL Framework

The program we have developed deals exclusively with 2D graphics. The program separated OpenGL code into three main sections: OpenGLScene, OpenGLDisplayController, and PolyBezierSceneRender, each focusing on specific tasks. OpenGLScene manages the entire OpenGL scene and call appropriate functions in the Controller when a task related to the user interface manipulation (zooming, scaling, etc) is requested, and the PolyBezierSceneRender when a task related to drawing Bezier curves is requested. All data in the OpenGL scene in OpenGLSceneData can be accessed by these three sections.

- Efficient Bezier Curve Evaluator

We decided to implemented the Bezier curve evaluation using Bernstein Basis instead of De Casteljau's Algorithm because of two reasons. First, Bernstein Basis has a closed form and thus is easy to compute. Although a very small numerical error may be introduced due to float number calculation, we have found that such errors does not interfere with the main features we are working with. Second, we can create robust *evaluators* that save computing time when a lot of curves with the same degrees are introduced in the scene, or even when a Poly-Bezier curve has several segments of the same degree. This can be very helpful since our program will very likely involve almost exclusively with Poly-Bezier curves that consist solely with only degree-3 segments.

The evaluators are basically a set of pre-computed Bernstein Bases, which are ready to be used for curve evaluation, simply by multiplying them with curve control points. Recall that Bernstein Basis is

$$B_i^n(t) = \binom{n}{i} t^n (1-t)^{n-i}, 0 \leq t \leq 1.$$

To evaluate a point on a Bezier curve, we need to find the summation of a product between all Bernstein Bases and their corresponding control points on that curve. To make the evaluation efficient, we implemented the evaluation process as follows:

Given a curve degree n, we end up having $n+1$ bases we will need for Bezier curve evaluation. These Bernstein Bases depends on not only the degree, but also the evaluating parameter t. We increase efficiency by pre-computing the terms independent of parameter t, which is $\binom{n}{i}$ where i is the index of a Bernstein Basis. Given a specific t, we can then use a provided function to generate complete set of Bernstein Bases. To increase efficiency further, we assume that the set of parameters t are often dependent of a level of details, or sample rate. Given such variable rate, we can precompute a full set of Bernstein Bases for all possible parameters t then stored them in a map. This map, mapping from parameter index to a set of proper Bernstein Bases, gives instant access an evaluation algorithm, which makes the calculation much faster during actual runtime.

- Derivation of the generalized smooth curve connection

Given two *Poly-Bezier curves*, or the curves that consist of solely degree-3 segments of Bezier curves, we would like to connect them while maintaining the shape of the curve and the smoothness at connecting point. We first assume that the curves have C^2 smoothness throughout. To connect two curves, we choose to use the criteria of the minimization of change in curvature, represented as the second-order derivative of a curve. Therefore, our simple *Energy Function* can be written as:

$$E = \sum_{i=0}^{x} F_i + \sum_{j=0}^{y} G_i,$$

$$F_i = \int_0^1 [\bar{C}_{0i}''(t) - C_{0i}''(t)]^2 \, dt,$$

$$G_j = \int_0^1 \left[\bar{C}_{1i}''(t) - C_{1j}''(t)\right]^2 dt$$

where C_0 is the first Poly-Bezier curve and C_1 is the second, and therefore $C_{0i}''(t)$ represents the curvature of the i^{th} Bezier segment of the first curve, evaluated at parameter t. The symbol x and y are the last index of the segment in each curve. The bar sign indicates the unknown, while the without indicates the known curve, where control points are given. The energy function E can be viewed as a sum of changes in curvature of all segments of the two curves being connected. We would like to compute $argmin_{\overline{a},\overline{b}}E$ to find an unknown set of control points $\overline{a}, \overline{b}$. Let us denote To compute for such control points, we consider the following desiderata:

1) Fixed non-connecting endpoints,
2) Joined connecting endpoints,
3) Smooth connection maintained. In other words, The new control points must still maintain the C^2 smoothness throughout the curve, both between the existing segments *and* the newly connected segment.

We solve for the unknown control points by letting the first derivative of E equal to zero. Most subsequent equations we obtain relate to the differentiation, with respect to each unknown control point, of an integral of the squared change in curvature. By simplifying each equation into differentiation under integral sign, we obtain a set of linear equations that we can program into our code and solve the unknown control points by a matrix solver.

5. Results and Discussion

Here we show several results from the implemented code mentioned above. Figure 1 illustrates three generic Poly-Bezier curves. The segments of each curve have various degrees, ranging from degree 1 to 4.

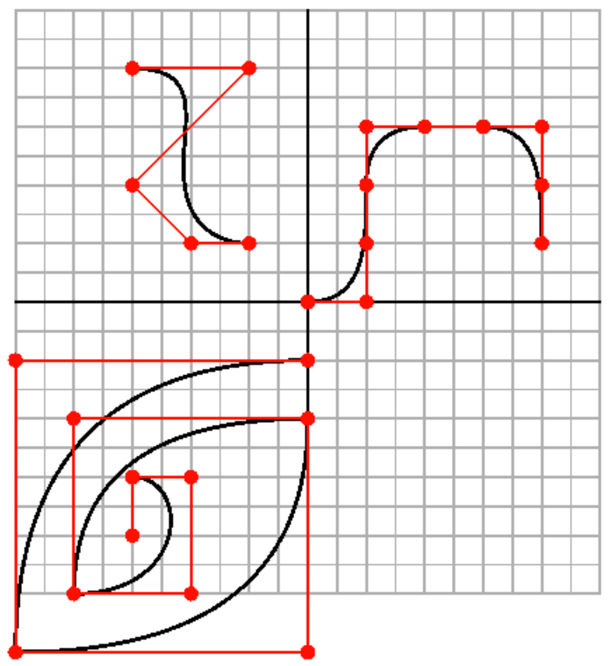


Figure 1: Generic Illustration of Poly-Bezier Curve Display

Figure 2 shows how curve connection works in a simple and ideal scenario, where we have only two segments and the endpoints are lined up closely with the same tangent direction. Notice that, after the connection, the shape of the new curve remains close to the previous ones.

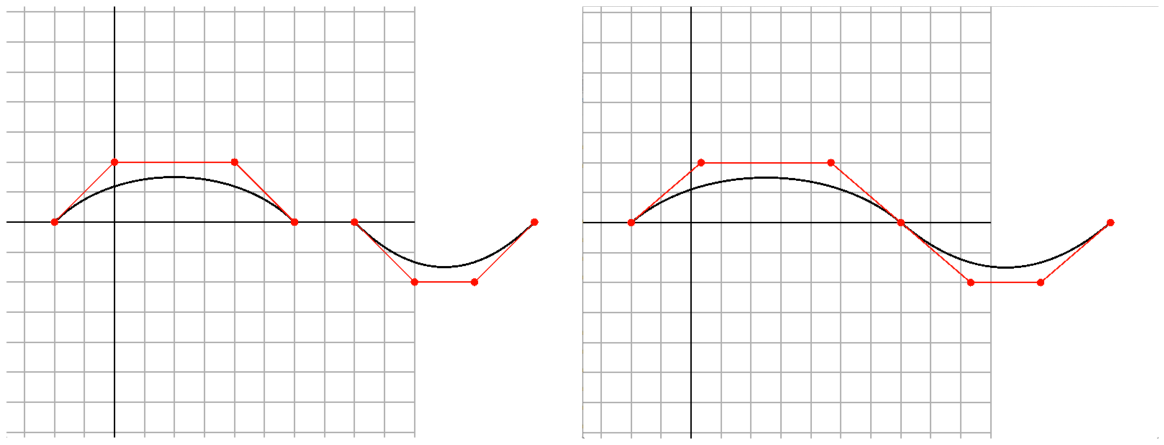


Figure 2: Curve connection of two simple segments

It takes special care to get the generic curve connection to work as expected, since as we have more segments in between the curve, we have to apply different differential equation constraints to them as opposed to the connecting segments. Figure 3 and 4 illustrate how the curve connection works robustly in the scene of two curves with arbitrary amount of segments:

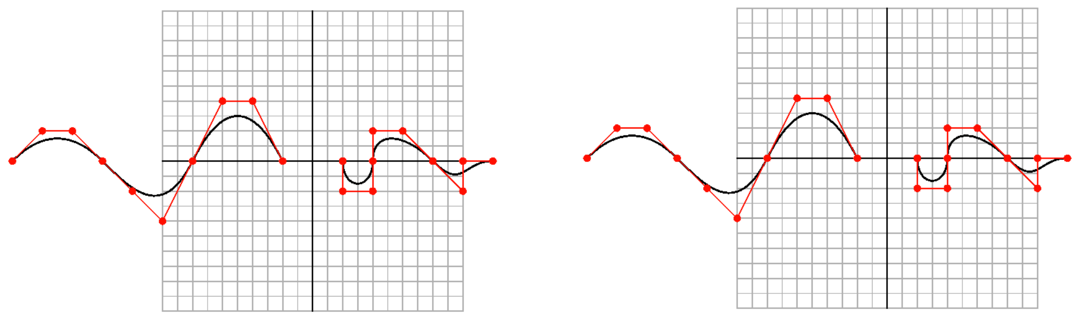


Figure 3: Curve connection of two three-segments Poly-Bezier curves
Notice that the control points change less and less as they can farther
From the connecting location.

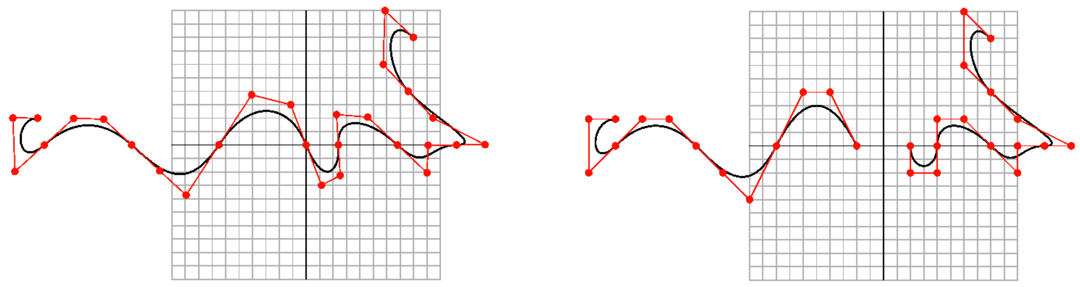


Figure 4: Curve connection between two curves, both with
arbitrary amount of degree-3 segments

In some cases, there may be the scenario where the curves are not meant to connect. For example, they might be far apart, or their tangents are not well matched, as shown in Figure 5. In this case, our curve connection may have to change the curvature drastically, which is to be expected.

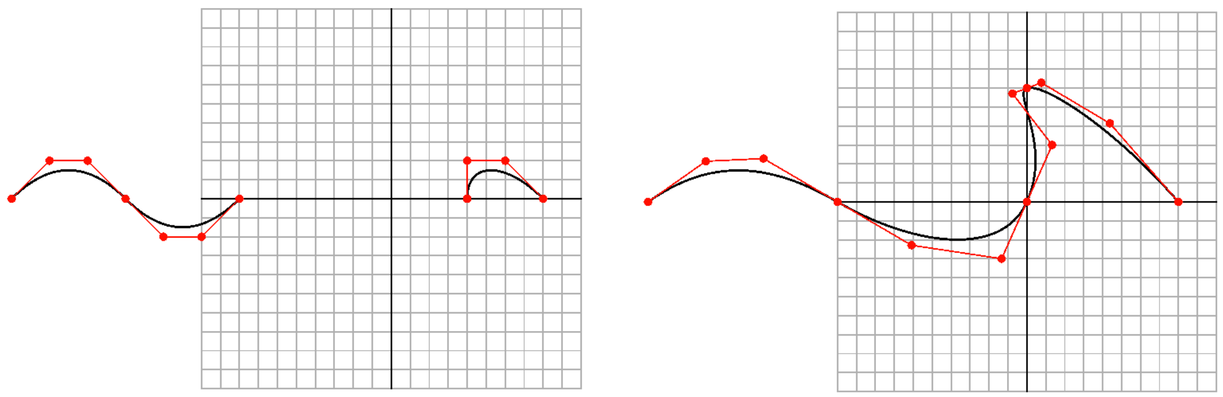


Figure 5: Curve connection that changes the curvature drastically

6. Lessons Learned

During this project I have learned tremendously both in theory, coding practices, and research mindset. Theory-wise, I have been given a chance to read many, many papers in texture synthesis during the early state of research, allowing me to immerse myself into the field. The amount of reading I have been given, combined with regular reading workload I had in the Seminar, teaches me a couple of things. I have learned how to read faster by constantly asking myself whether I understand the high-level algorithm paper or not, then as time allows I can then go a little deeper to explore more details. I have also learned not to read papers just for the sake of reading, but for building up ideas and understanding the context of works being done in the field, so I am context-aware and prepared for my own research as I collect background knowledge and new ideas.

I have also learned to work with math in a sense that deals with arbitrary size of variables—a system that I have only been exposed in Computer Animation class as a previous background. However, this time I learn to derive the formula from scratch, and implement the linear equations in the arbitrary sized matrix form all on my own, which allows me to understand significantly how we can employ computers and matrix libraries like Eigen to help us solve time-consuming problems. Earlier in the process I would spend a lot of time trying to derive the matrix for just two one-segment curves until I obtained a final coefficient matrix that I could use to hardcode into Python script to see some results. Although the results were correct, I was then stuck with the idea of how to generalize the matrix. Through the help of Prof. Changxi Zheng, I have learned substantially when I learned to derive the equations *only as needed*; in other words, I do not have to simplify the equation any further if such simplification rule can be implemented in the matrix itself, such as the rules of the control points as described in the desiderata. Personally, I am very glad that the matrix has worked successfully after correcting countless mistakes I have made during the derivation process.

Coding-wise, I have also learned to code in a larger scale. Knowing that this project will be carried as a year-long project, I planned out my code structure very early on and made sure that my code remained generic and expandable as much as possible when I added more features after. In addition, I have learned several techniques from coding OpenGL codes from scratch, but based heavily on FOSSSim, which amazes me greatly in terms of how well-designed it is when I look closely. I have also learned a couple of little things including the use of template classes, and the implementation of user-defined STL-style iterator.

Lastly, I have learned to be more research oriented as I am preparing myself toward the graduate study. Advised by Prof. Eitan Grinspun, I have learned a different perspective: instead of trying to code my program as structurally beautiful as possible, I should aim to implement the main algorithm as fast as possible since I have to see if the algorithm should be included in the main framework or not. Furthermore, as there is no hard deadline like assignments in other classes, I have learned to find my balances in doing research and maintaining my performance in other classes. I have come to understand that doing research can have slow progress, since no obvious instructions or solutions are given to me since the beginning. However, the exploration process has been really rewarding as I learned several things as mentioned above along the way. In sum, I am really glad that I have been given an opportunity to work on this project, and I am truly looking forward to implement more features to make this project mature.

7. Conclusion and Further Plans

To conclude, there are several contributions to the project Vector Texture Synthesis. We have presented an OpenGL 2D framework, an efficient Poly-Bezier curve evaluation algorithm, and the curve connection method based on the curvature change minimization. There are many things that can be added to grow the project further. On a smaller scale, we can automate the curve loading step by parsing input form SVG file, present a *history* of curves to compare before- and after- curve connection process in one run, or use sparse matrix solver to make the coefficients solving process faster. On a larger scale, we are looking forward to generate simple textures by connecting a network of curves, incorporating Diffusion Curves [Orzan et al. 2008] into the synthesis process, or generating texture in a hierarchical fashion.

8. Reference

BARLA, P., BRESLAV, S., THOLLOT, J., SILLION, F., AND MARKOSIAN, L. 2006. Stroke pattern analysis and synthesis. In *EUROGRAPH '06*, vol. 25, 663–671.

IJIRI, T., MECH, R., IGARASHI, T., AND MILLER, G. 2008. An example-based procedural system for element arrangement. In *EUROGRAPH '08*, vol. 27, 429–436.

MA, C., WEI, L. AND TONG, X. Discrete Element Textures. *In Proc. ACM SIGGRAPH* 30, 4 (2011), Article 62

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: a vector representation for smooth-shaded images. In *ACM SIG- GRAPH 2008 papers*, ACM, New York, NY, USA, SIGGRAPH '08, 92:1–92:8.