ABSTRACT

Real Time Animation Using Spline Surfaces Within the GPU

Zachary Lapointe

In computer graphics, frequently buffering high-resolution models (models defined with a large number of vertices) to the GPU is a significant performance bottleneck, as sending data between the CPU and GPU is relatively slow compared to the processing speed of either device. For this reason, a complex model is usually sent once to the GPU, where simple and limited transformations are applied to it. Alternatively, by drawing high-resolution models using a relatively small number of vertices, the buffering between devices becomes significantly less costly, and complex animation operations can be performed in the CPU. Using spline surface functions, models of any resolution can be represented with relatively few vertices. The models can be rendered by generating the vertices at run time, in the GPU, from the spline surface’s control points and function. This enables the implementation of unusual real time animation which act non-linearly on vertices, such as evolving force fields and inter-vertex forces.

Real Time Animation Using Spline Surfaces Within the GPU

A Report Presented to
Dr Sudhir P. Mudur
Concordia University

In Fulfillment
of the Requirements
of COMP 490

by
Zachary Lapointe
ID: 27518536
Concordia University
April 2020

Table of Contents

1. INTRODUCTION

1.1 Problem Statement

3D animation allows changing models in any imaginable way. Modern real time animation runs into certain bottlenecks, however. In particular, throughput between the CPU and graphics card can be quite limited compared to the computation capabilities of either device. Because model data needs to be sent do the GPU to be rendered, this creates significant limitations for real time animation. Because of this, trade-offs must be made, usually in one of two directions. One option is to reduce the size of the data transmitted, which can be achieved with lower resolution models. Alternatively, a high-resolution model can be sent to the GPU once, during program initialization. However, this limits the types of animation which can be done, as high-level geometry created during modelling is lost. Though techniques, such as hierarchical modelling, have been developed to counter these limitations, vertices of pre-tessellated models are effectively bound together in animation. There is thus a need for an alternative which can allow delaying model tessellation until rendering.

1.2 Background

Splines are computationally simple mathematical functions which can be used to interpolate arbitrarily complex shapes. As they are continuous functions, they are completely precise within their domain. Spline surfaces have been used extensively in modelling software for representing models. In real time animation (primarily video games) these models are used in a vertex format, where the spline surfaces are tessellated into simple polygons, often triangles. The higher the desired resolution of a model, the more polygons the spline surfaces are tessellated into. However, the tessellation removes the precise property of the model, and its spline geometry. This is necessary however, as graphical renderers rely on these simple polygons for rendering models.

The OpenGL pipeline is no exception, the only polygon it can render are triangles. Recently, in 2010, version 4.0 of the OpenGL standard allowed for doing tessellation in real time, on the GPU through two programmable shader stages: the *tessellation control shader* (TCS) and *tessellation evaluation shader* (TES), which are capable of generating new vertices from some source input, when programmed correctly.

1.3 Solution

By using splines to define models, similar to how it is done with modelling software, and by deferring tessellation until rendering time, one can create high resolution models which can be precisely animated, by sending the control points defining the surface to the GPU, and tessellating it using the new OpenGL shader stages.

2. RESULTS

2.1 OpenGL Pipeline Implementation

This project makes use of all the modifiable shader stages of the pipeline.

Vertex Shader

The vertex shader is used only for passing control vertex data along to the next stage. No transformations are applied, though it could be used to apply global model transformations (scaling, rotation and translation) to the spline surface's control points.

Tessellation Control Shader

The level of tessellation is controlled through the TCS, with a uniform variable. All inner and outer tessellation levels are equal across all patches. For a tessellation level of 64, each patch (which is of type quad) of 16 points creates a uniform grid of 65*65 points, which create a tessellation of 64*64*2 triangles.

Tessellation Evaluation Shader

This is where the spline surfaces are primarily implemented. The tessellated points are evaluated with the spline surface vector equation

$$R(u,v)=U(u)^T M^T P M V(v)$$

where ***U*** and ***V*** are cubic parameter vectors, **P** is the control point matrix and **M** is the characteristic matrix corresponding to the spline type.

Furthermore, normals at the tessellated points are determined with the normal equation

$$N(u,v)=\frac{R_u \times R_v}{\left\| R_u \times R_v \right\|},$$

$$where\ R_u \wedge R_v\ are\ the\ \partial\ derivatives\ of\ the\ surface,\ evaluated\ at\ (u,v)$$

Geometry Shader

The geometry shader takes vertex data output from the TES and arranges their positions and normals into triangles, as the usage of patch primitives required for tessellation requires explicitly defining the geometry of the tessellation output.

Fragment Shader

Finally, the fragment shader generates the fragments to be displayed. It does nothing pertaining to the tessellation. It implements the Phong lighting algorithm using the normals computed in the TES.

2.2 Patch Calculations

The program allows patch vertices and normals to be calculated for Bézier spline surfaces and Catmull-Rom spline surfaces, using the above equations. Their characteristic bicubic matrices were derived, and are listed in appendix A.

2.3 Spline Cylinder

Two or more bicubic patches can be joined to form a smooth, cylinder-like shell, by joining them in a *circular array*. Control points for the first patch are reused for the final patch to ensure smoothness across the surface along its width.

2.4 Toroidal Spline Surfaces and Animation

Joining many spline cylinders structures into a ring structure allows the construction of a toroidal surface. This surface is equivalent to a grid of

patches, whose *width* is the number of patches across a single cylinder, and whose *height* is the number of patches along one revolution of the toroid. Thus, such a surface is composed of $width \times height$ patches, laid out in a *spherical grid* (the topology of the grid is as follows: the top of the grid is connected to the bottom, and the left side to the right side, similar to a circular array, but along two dimensions). The resulting spline surface is closed, and smooth everywhere. This structure is animated in real time with various forces. There are force fields which apply forces to the control points of the structure depending on their position, and tension forces which attempt to maintain the structures integrity, acting on neighbouring control points. Controls for moving the camera and modifying the tessellation are listed in appendix B.

3. DIFFICULTIES & LIMITATIONS

3.1 Continuity Across Patches

Many challenges were encountered with regards to surfaces having the property of C1 continuity (all points being continuously differentiable once, in all directions). This property is quite useful to have for surfaces in graphics, as the normal of a spline surface depends directly on its partial derivatives: continuously differentiable surfaces give rise to continuous normals along said surfaces, which allows for smooth lighting. Thus, for a surface which is meant to be smooth, being able to guarantee C1 continuity is necessary. However, guaranteeing C1 continuity along a spline surface is somewhat tricky. Though individual patches are easily sewn together continuously, it often must be done under specific conditions. For example, to guarantee C1 continuity, adjacent Bézier patches must have colinear points along their edges, in outgoing directions. This means that such patches must be arranged in a grid structure. Creating rounded closed surfaces from grids is quite difficult; the simplest such shape is a 1-holed torus. Thus, 0-holed, C1 smooth, closed surfaces, such as sphere or rounded cube, cannot be constructed with Bézier patches. A similar argument holds true for Catmull-Rom patches. Though not the focus of this project, this can be shown in two parts. First, that Bézier patches cannot be adjacently collinear unless in a grid structure, and second that grids cannot form smooth, 0-holed, closed structures.

4. AVENUES FOR FURTHER DEVELOPMENT

4.1 Dynamic insertion and removal of patches

The toroidal structures, though somewhat limited by their topology, can be animated in many ways while retaining C1 smoothness. The number of possible bends in the structure is limited by the number of cylindrical sections along the toroid, with each bicubic section allowing two curves and an inflection point. To increase these, one can add more cylindrical sections into the structure. However, more sections make animation more complex, as there are more points to control. Thus, the ability to dynamically control the number of sections in given structure can make said structures more flexible. Thin cylindrical sections of zero width can be inserted between others, and then expanded over time, creating a seamless transition. The reverse can be done to remove sections as well.

4.2 Bézier Triangles

The above-mentioned issue regarding the impossibility of smooth no-holed surfaces is a predicament. However, it may be addressable with a more unusual type of spline surface, the Bézier Triangle [CITATION], which is composed with barycentric coordinates. Depending on their continuity conditions, it may be possible to create a closed surface with either an octahedral or icosahedral lattice of Bézier triangles, which could lead to interesting animation opportunities.

5. CONCLUSION

In modern graphics pipelines, one of the most significant bottlenecks is the buffering of data to GPU, forcing a trade-off between the resolution and animation complexity of models. However, using OpenGL’s recent shader advances, models can be represented as spline surfaces, and this trade-off can be avoided. This allows complex animation to be performed on the CPU, where recursion and non-linear data structures can be used for unusual animations, such as simulating inter-vertex forces. In this project, a toroidal surface was generated and deformed in real time, in a manner which would be otherwise difficult to replicate. However, there is much to improved upon in the method, in particular when it come to ensuring continuity between patches in non-grid-like topology.

APPENDIX A: CHARACTERISTIC MATRICES

Bézier spline matrix

$$M_B = \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Catmull-Rom spline matrix

$$M_{CR} = \frac{1}{2} \cdot \begin{bmatrix} 0 & -1 & 2 & -1 \\ 2 & 0 & -5 & 3 \\ 0 & 1 & 4 & -3 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

APPENDIX B: PROGRAM CONTROLS

W, A, S, D: Moving the camera, in a 1st person style

Control: Lowers the camera

Shift: Raises the camera

Q: Halves the tessellation level, down to a minimum of 1

E: Doubles the tessellation level, up to the maximum defined by GL_MAX_TESS_GEN_LEVEL

Escape: Exits the program

APPENDIX C: PROGRAM REQUIREMENTS

Java 1.8 runtime or compatible

Support for OpenGL 4.0 or higher

APPENDIX D: USE OF EXTERNAL LIBRARIES

LWJGL: A set of Java bindings for many hardware standards, here used for OpenGL and GLFW

JOML: A mathematics library providing functions and classes relevant to 3D graphics (vectors, matrices, etc.)

APPENDIX E: OPENGL CAPABILITIES OF TEST COMPUTER

Name	**Minimum**	**Test Computer**	**Enum**
OpenGL Version	4.0	4.6	GL_MAJOR_VERSION, GL_MINOR_VERSION
Maximum vertices per patch	32	32	GL_MAX_PATCH_VERTICES
Maximum tessellation level	64	64	GL_MAX_TESS_GEN_LEVEL