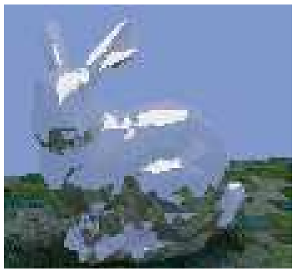
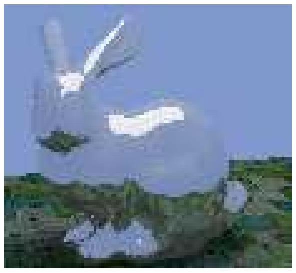
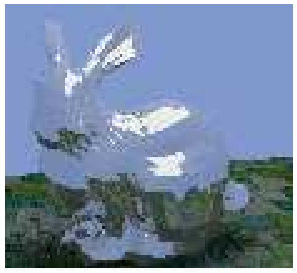
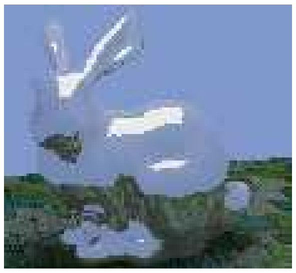
Automatic Smoothing of Quad & Triangle Meshes

Patrick Donnelly


Coarse Bunny
(1k triangles)


Smooth Bunny
(1k triangles)


Coarse Bunny
(1quads)


Smooth Bunny
(1k quads)

Abstract

I present a methodology to create smooth meshes using spline patches generated automatically from the geometry of the mesh. The intent is to create a smooth surface across the entire mesh, which creates smooth silhouette edges and is very useful for high quality ray tracing. An additional goal was that this methodology could be used on meshes made out of triangle faces, quadrilateral faces or a combination of both.

1. Introduction

1.1 Motivation

Ray tracing is a popular field in computer graphics due to its ability to create stunningly photorealistic renderings. However, in order to create high quality renderings, high quality meshes are a requirement, such as meshes with a high polygon count or implicit surfaces, otherwise visible artifacts will be present due to the lack of information or discontinuities. Additionally, high quality meshes increase rendering time as more complexity is introduced. Furthermore, one alternative of using meshes made out of splines can be both tedious and difficult to setup without introducing discontinuities at the seams. The goal of the smoothing method in this paper is to remedy these issues using pre-existing polygon mesh and guaranteeing continuous normals (G1 continuity) across the entire surface. Outside of ray tracing, this method also has applications to Level of Detail and mesh subdivision.

1.2 Related Work

The creation of smooth meshes from polygonal meshes is not a new concept. The problem has been solved for triangle meshes in multiple ways, including using PN Triangles [1] and bi-cubic splines [2]. Furthermore, a method of ray tracing cubic Bezier patches interactively has been developed by Benthin, Wald and Slusallek [3].

2. Data Structures

2.1 Axis-Aligned Bounding Box Array

In order to efficiently raytrace the scene, bounding boxes are used to drastically reduce the number of costly face intersections. This is even more important when dealing with splines, which are particularly costly to compute. A slightly modified version of the algorithm created by Kay-Kajiya [4] is used. Only axis-aligned slabs are chosen and as much pre-computation as possible is done, so the end result is that minimal amounts of math are needed to detect intersections between the collision boxes and the ray. Furthermore, while the bounding boxes are hierarchical, they are stored in memory as a depth-first array [5]. This speeds up the collision detection by eliminating the need for recursion and practically assuring good memory cohesion during traversal.

2.2 "Smart" Edges

In addition to the typical data stored by the half-edge data structure, edges keep useful information such as the normal at the vertex, the tangent to the plane defined by the normal in the direction of the edge, and the length of the edge. This information is very useful in simplifying and optimizing the creation of the control mesh for the spline patches.

In addition, for quad faces two "pseudo-edges" are added across the diagonals to keep the code consistent for all spline calculations.

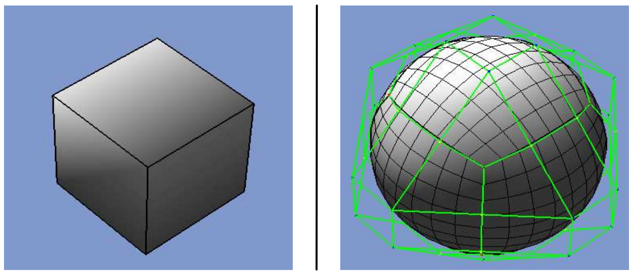
3. Algorithms

3.1 Quad Face Control Mesh Generation

The most important aspect of this methodology is how it generates the control mesh that defines the smooth spline for each face. Cubic Bezier patches were chosen as they have generally well understood behavior, they give a good degree of control over the surface and are obviously well suited for a quad face. Here is the general flow of the algorithm:

1. Calculate the blended normal for each vertex. This is calculated by averaging all the normals at the vertex for each face surrounding it.
2. For each edge, find the tangents to the planes defined by the blended normal at that edge's two vertices and in the direction to the opposite vertex.
3. Extend these tangents out by the length of the edge multiplied by the *tangent_scale*, which for the best results is found to be *0.3505466*. These points define the outer control points for the control mesh of the face.
4. For each face diagonal, repeat steps 2 and 3 except with the *tangent_scale* in step 3 multiplied by $\sqrt{2}$. These points define the interior control points of the control mesh of the face.
5. Using the vertices for the corners of the control mesh, the edges for the outer control points and the diagonals for the internal control points, a Bezier patch can be generated.

The *tangent_scale* values were chosen as they will produce a nearly perfectly continuous smooth mesh out of a perfectly regular quad mesh (i.e. a sphere is made out of a cube).


Source mesh and smooth mesh showing control mesh.

One important note is that while the patches generated are almost always geometrically continuous across seams, the normals defined by the patches are not continuous across seams. In order to produce continuous normals, which is necessary for ray tracing, linear interpolation of normals is used as decent approximation.

3.2 Solving for the Surface

Solving for a given parametric location $P(s,t)$ on the Bezier patch given its control points was handled through matrix math as follows [6]:

$$C = \begin{bmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{bmatrix}$$

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

$$M = B \cdot C \cdot B^T$$

$$P(s,t) = \left[s^3, s^2, s, 1\right] \cdot M \cdot \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

In the equations above, *C* represents the control points as setup in the previous section, *B* is the Bezier basis matrix and *M* is a matrix that gets computed once and stored for optimization. This calculation can be used to discretize the mesh and to determine the point of intersection with a ray.

3.3 PN Triangles

In order to handle triangles, the curved PN triangles algorithm described by Alex Vlachos was used [1]. A PN triangle will generate a cubic Bezier surface automatically from the vertex positions and normals, much like my algorithm for quad faces. That paper should be refered to for the implementation of the algorithm.

3.4 Ray tracing

Due to time constraints, ray tracing was performed by subdividing the mesh into discrete faces and using Moller-Trumbore ray-triangle intersection [7]. While this is extremely inefficient it works well enough to demonstrate the improvements of this methodology. The algorithm proposed by Benthin [3] could be exchanged and produce much faster results.

4. Results

The results of the mesh smoothing methodology are marvelous. Low quality quad meshes with extremely rough silhouette edges are made smooth with no rough silhouette edges. All creases are removed and the mesh as a whole looked much more visibly pleasing.

The increase in quality of the mesh also directly correlates to an increase in quality of the ray tracing. Reflections and lighting were the most obvious areas of improvement as the smoothed positions and normals produced much higher quality results.

Triangle meshes and hybrid meshes were not extensively explored as the utility of being able to smooth such meshes was realized to be minimal as few hybrid meshes are in existence (and none could be found!). A simple mesh was created and seemed to work out well enough, though the result was not as expected.

5. Future Work

One obvious area for improvement involves speeding up the ray tracing. That would greatly improve the benefits of this methodology, as there easily could be an increase in speed over other similar methods such as subdivision.

References

[1] Alex Vlachos , Jörg Peters , Chas Boyd , Jason L. Mitchell, Curved PN triangles, *Proceedings of the 2001 symposium on Interactive 3D graphics*, p.159-166, March 2001
[2] Toshio Ueshiba, Gerhard Roth, "Generating Smooth Surfaces with Bicubic Splines over Triangular Meshes: Toward Automatic Model Building from Unorganized 3D Points," 3dim, pp.0302, *Second International Conference on 3-D Imaging and Modeling* (3DIM '99), 1999
[3] Carsten Benthin, Ingo Wald, and Philipp Slusallek. Interactive Ray Tracing of Free-Form Surfaces. *ACM Afrigraph*, 2004.
[4] Timothy L. Kay and James T. Kajiya., Ray tracing complex scenes. In *Computer Graphics*, pages 269-278, ACM SIGGRAPH, 1986.
[5] Brian Smits, Efficiency issues for ray tracing, *Journal of Graphics Tools*, v.3 n.2, p.1-14, 1 February 1998
[6] Bézier Surfaces. 24 Apr. 2009 <http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0405/DONAVANIK/Bezier.html>.
[7] Tomas Moller & Ben Trumbore, "Fast, Minimum Storage Ray-Triangle Intersection", *J. Graphics Tools 2(1)*, 21-28, 1997