EFFICIENT RENDERING OF NURBS AND T-SPLINE SURFACES

Adarsh Krishnamurthy, Yusuke Yasui

Figure 1: NURBS surfaces rendered using the graphics card

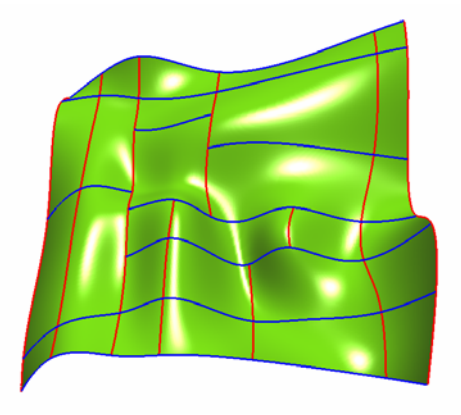

Figure 2: T-spline surface

PROJECT SUMMARY

In this project, we developed a method for evaluation and rendering of NURBS and T-splines surfaces using programmable graphics card. The main motivation is that GPU, being faster than the CPU can improve the performance of surface evaluation. The implementation consisted of two major parts. First part is the CPU implementation that includes setting up of knot vectors and control points in regular order so that they can be transferred to the GPU as textures. The second part is the GPU computation part that includes computation of basis functions and final surfaces from these textures. We achieved a speed increase of about 10 to 50 times by using the graphics card for evaluating NURBS surfaces.

INTRODUCTION

NURBS or Non-Uniform Rational B-Splines are the industry standard in commercial Computer Aided Design systems for the representation and design of geometry. The low-level primitives like triangles and lines can be used to define arbitrarily shaped objects, but at the cost of mathematical properties; for example, a circle that is approximated by a sequence of line segments will change its shape when rotated. One of the advantages of NURBS curves is that they offer a way to represent arbitrary shapes while maintaining mathematical exactness and resolution independence. Among their useful properties are the following

1. They can represent virtually any desired shape, which include points, straight lines, polylines, conic sections (circles, ellipses, parabolas and hyperbolas) and free-form curves with arbitrary shapes.
2. They give you great control over the shape of a curve. A set of control points and knots, which guide the curve's shape, can be directly manipulated to control its smoothness and curvature.
3. They can represent very complex shapes with remarkably little data.
4. They are invariant under affine as well as perspective transformations.
5. They can be evaluated reasonably fast by numerically stable and accurate algorithms.
6. They are generalizations of non-rational B-Splines and non-rational and rational Bezier curves and surfaces.

Even though NURBS are ubiquitous in the Design Industry, there is no built-in hardware support for displaying NURBS surfaces in current graphic cards. Therefore, all current CAD systems evaluate and tessellate the NURBS into triangles and then use the existing graphics pipeline to display them. This is not an ideal way to deal with rendering NURBS surfaces as the tessellation is done only for visualization purposes and it is usually a time consuming process. Thus, this project based makes use of programmable graphics hardware to display NURBS surfaces without actual tessellation.

T-splines are a much more general form of spline surfaces that allow T-junctions in the control mesh. Thus, not only T-splines have all the advantages of NURBS but also have some desirable features like local control and reduction in the total number of

control points. Figure 1 shows an example of the grid of NURBS surface (left) and the grid of T-spline surface (right). Figure 2 shows a corresponding T-spline surface of the grid in Figure 1b. The blue and red lines on the surface correspond to the lines of the grid. The implementation of T-spline is not very different from that of B-spline surfaces or NURBS surfaces. However, the grid of T-spline is not regular, so determining control points that affect the coordinates of the surface at a particular parametric position is complicated.

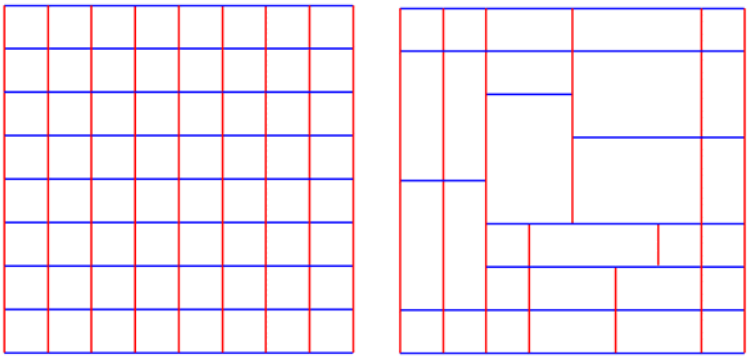

Figure 3 : NURBS and T-spline control Mesh

Figure 3a shows the area where one control point affects in a NURBS surface and the figure on the right shows the same in the case of T-spline surfaces. These examples are specific for bi-cubic surfaces and in the case of NURBS surfaces, it is the area enclosed within the 16 control points shown. In T-spline, the area is determined by considering rays going to left, right, top and bottom from the control point. The second intersection of the each ray defines the boundary of the area where the control point affects. Therefore, the area affected by each control point is complicated, and hence some preprocessing is required.

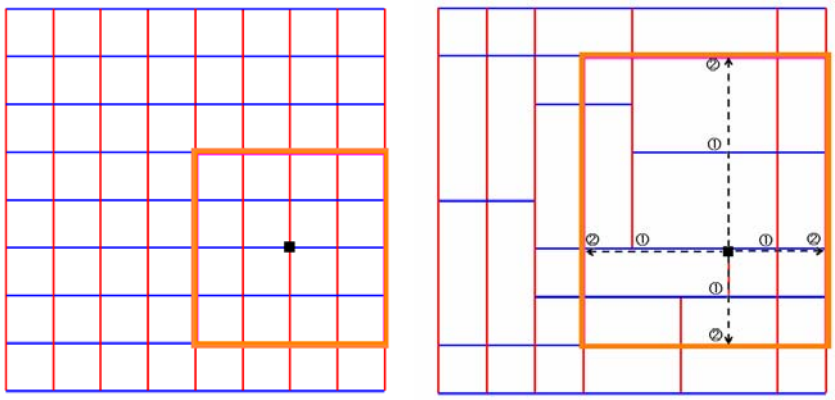

Figure 4: Area affected by a control point in the case of NURBS and T-splines

NURBS THEORY

Mathematically NURBS curves are defined by the set of points called the control points. The amount by which a control point influences the curve is defined by the BSpline basis function and the extent to which it influences the curve is defined by a vector called the Knot vector. Moreover, the control points can also have weights attached to them that make the BSpline rational.

Equation (1) gives the definition of a NURBS curve C as a function of the parameter u. The $N_{i,p}$ is the BSpline basis function given by Equation (2). The u_i in the definition of the basis functions represent the i^{th} component of the knot vector. This definition of the basis function is recursive and the basis function of order k depends on the basis functions of order k-1.

$$C(u) = \sum_{i=0}^{n} \left(\frac{N_{i,p}(u) w_i}{\sum_{j=0}^{n} N_{j,p}(u) w_j} \right) \mathrm{P}_i \tag{1}$$

$$\begin{aligned} N_{i,p}(u) &= \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \\ N_{i,0}(u) &= \begin{cases} 1 & \text{if } u_i \le u \le u_{i+1} \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

NURBS surfaces are a direct extension of the NURBS curves. They are tensor product surfaces of NURBS curves in two parametric directions. Equation (3) gives the mathematical definition of such a surface.

$$S(u,v) = \frac{\sum_{j=0}^{m} \sum_{i=0}^{n} N_{i,p}(u) N_{j,q}(v) w_{i,j} \mathrm{P}_{i,j}}{\sum_{j=0}^{m} \sum_{i=0}^{n} N_{i,p}(u) N_{j,q}(v) w_{i,j}} \tag{3}$$

T-SPLINES EVALUATION

Point-Based spline is a fundamental expression of a T-spline surface. The equation is the same as NURBS, that is, the linear combination of control points and basis function. However, the number of control points *n*, is different for each evaluation point. Moreover, locating the control points for surface coordinate computation at the parametric coordinate (*u*, *v*) is complicated. Once the control points are determined, the computation is just the linear combination of the control points and their corresponding basis function as shown in Equation (4).

$$P(u,v) = \frac{\sum_{i=1}^{n} B_i(u,v) P_i}{\sum_{i=1}^{n} B_i(u,v)} \tag{4}$$

$$B_i(u,v) = N_i^3(u) N_i^3(v) \tag{5}$$

Figure 5 shows control points needed to evaluate the surface on the green dot in both NURBS and T-spline case. In the NURBS case, it consists of the grid of 16 control points. On the other hand, in T-spline case, the location of control points is not orderly.

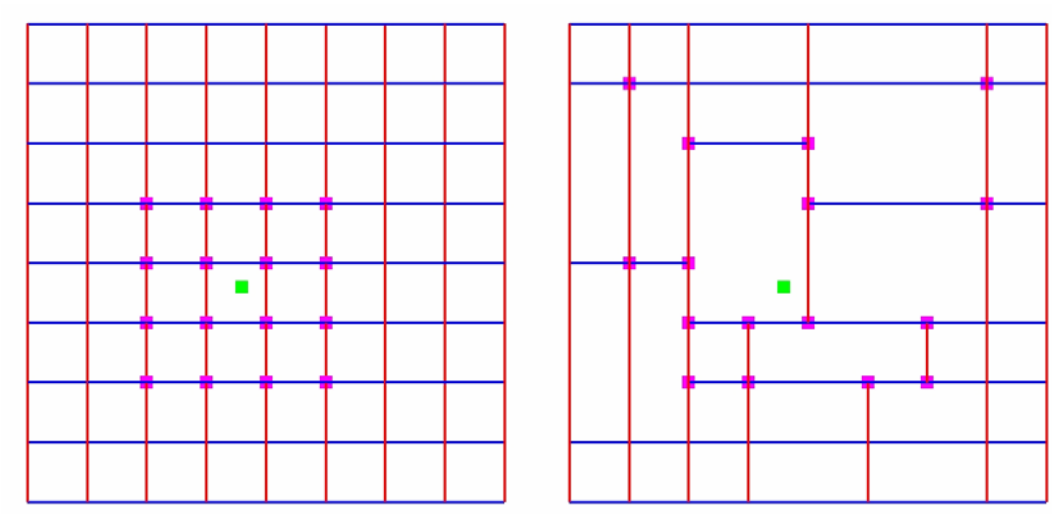

Figure 5: Control points required to evaluate the coordinates at a parametric coordinate

Moreover, the control points needed to evaluate the surface is different even in a same face as shown in Figure 6. This is because the boundary of the area where one control point affects sometimes cuts across a face as shown in Figure 4.

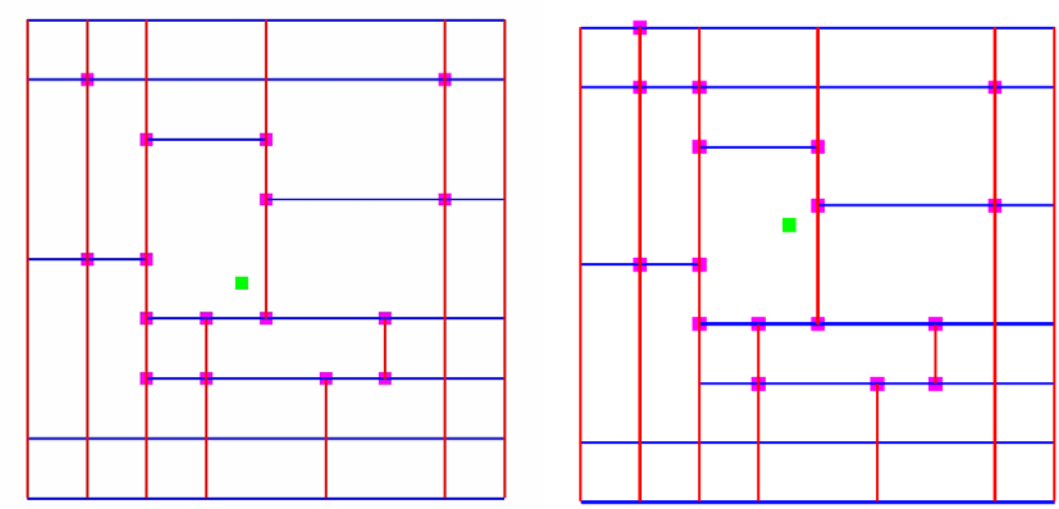

Figure 6: Control points required to evaluate the coordinates at a parametric coordinate

Since our goal is to render these surfaces as fast as possible using GPU, fast query of control points is essential for speeding up the computations. Therefore, for fast query, we divide each face of the T-mesh if other knot values are defined within the range of the face (Figure 7). Then we store the control points that are needed to evaluate the surface in each region in 2-dimensional array. The query time is in O(log *n*) for a particular parametric coordinate using binary search.

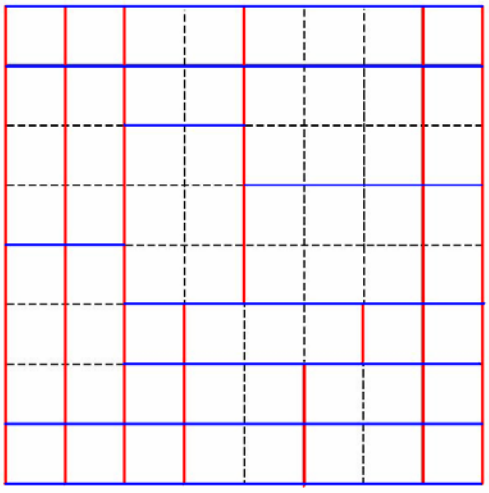

Figure 7: Dividing the T-mesh into regions

NURBS GPU EVALUATION

Given all the data for a NURBS surface, the surface point coordinates at parametric coordinates (*u*, *v*) is computed in the following manner.

1. Locate the lower left corner of the sub-mesh of control points that influence the evaluation point coordinates.
2. Compute the non-zero basis functions along the parameter directions.
    a. Compute the non-zero u basis functions using the u-direction knot vector.
    b. Compute the non-zero v basis functions using the v-direction knot vector.
3. Multiply the non-zero basis functions with their corresponding control points from the sub-mesh and sum the results.

The first step of computing the lower left corner control point that influences the current surface point coordinate is equivalent to the first step in the curve evaluation; it is done on the CPU and transferred as 1D texture to the graphics card. The two substeps of the second step are each performed in the GPU using a fragment program. Finally, the evaluated basis functions are multiplied with the corresponding control points and added together. Figure 8 represents the surface evaluation process pictorially. We specify the parametric u and v coordinates of the points required to be evaluated in the CPU. We then calculate the basis functions corresponding to these coordinates on the GPU using the basis-function evaluation program and generate the two textures for u and v having the basis function values at the required parameter coordinates.

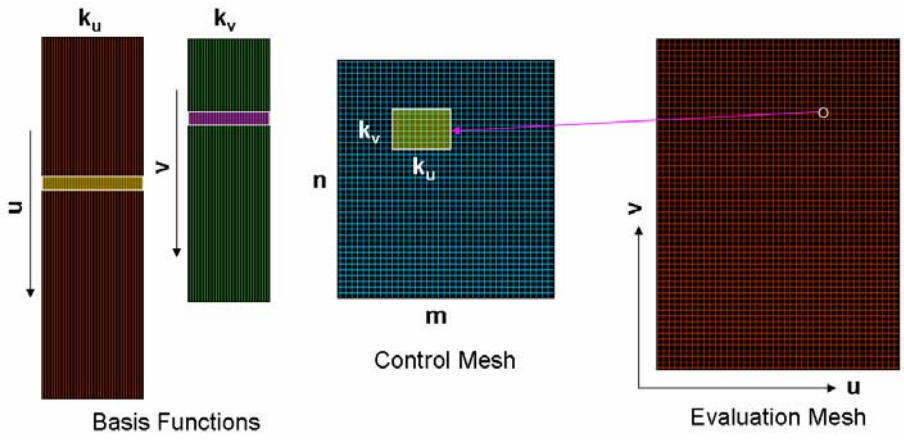

Figure 8: Surface evaluation method

Once the basis functions are evaluated, we again alternate between output textures to evaluate the final surface coordinates. We store the control point data in a texture of size $n \times m$ in the GPU memory. We also have a texture of size equal to the evaluation mesh (tex1), which is initialized to zero. Given a particular u and v coordinate, we look up the coordinates of the control point that influences the current evaluation point using the index values stored in the 1D textures calculated in step 1. We then multiply this control point with its corresponding u and v basis function values and add it to tex1 using a fragment program. This hardware fragment program directly renders the multiplied result to another texture, tex2. In the next pass, the newly multiplied values of this pass are added to tex2 and rendered directly to tex1. Thus, the final curve point is evaluated in $k_u \times k_v$ passes; for example, a bi-cubic NURBS surface point is evaluated in 16 passes.

T-SPLINES GPU EVALUATION

The GPU T-spline evaluation is also very similar to the NURBS evaluation and consists of three similar steps.

1. Identify the indices of all the control points that influence the evaluation point coordinates and store them in textures. This is the step done on the CPU and it has some required preprocessing as mentioned before.
2. Compute the non-zero basis functions along the parameter directions for each control point.
    a. Compute the non-zero u basis functions using the u-direction knot vector.
    b. Compute the non-zero v basis functions using the v-direction knot vector.
3. Multiply the non-zero basis functions with their corresponding control points from the sub-mesh and sum the results.

The main difference between the NURBS and T-splines is in step 1 and 2. In step 1 all the different control points that influence a particular point coordinate needs to be computed and stored. Also in step 2, the basis functions have to be calculated for each control point as each control point has its own knot vector. However, due to limitations of the hardware we were not able to implement T-spline evaluation on the GPU.

TIMING RESULTS

Figure 9 compares the evaluation timings of a NURBS patch made of 144 control points on increasing the size of evaluation grid. It can be clearly seen that the GPU based evaluation is better than the CPU by at least by a factor of 10 for smaller patches to around a factor of 50 for big patches. In addition, the model can be interactively displayed with varying levels of detail without sending the data to the GPU repeatedly. Also any changes to the model will necessitate the sending of only the control points that are much fewer in number than the evaluated points.

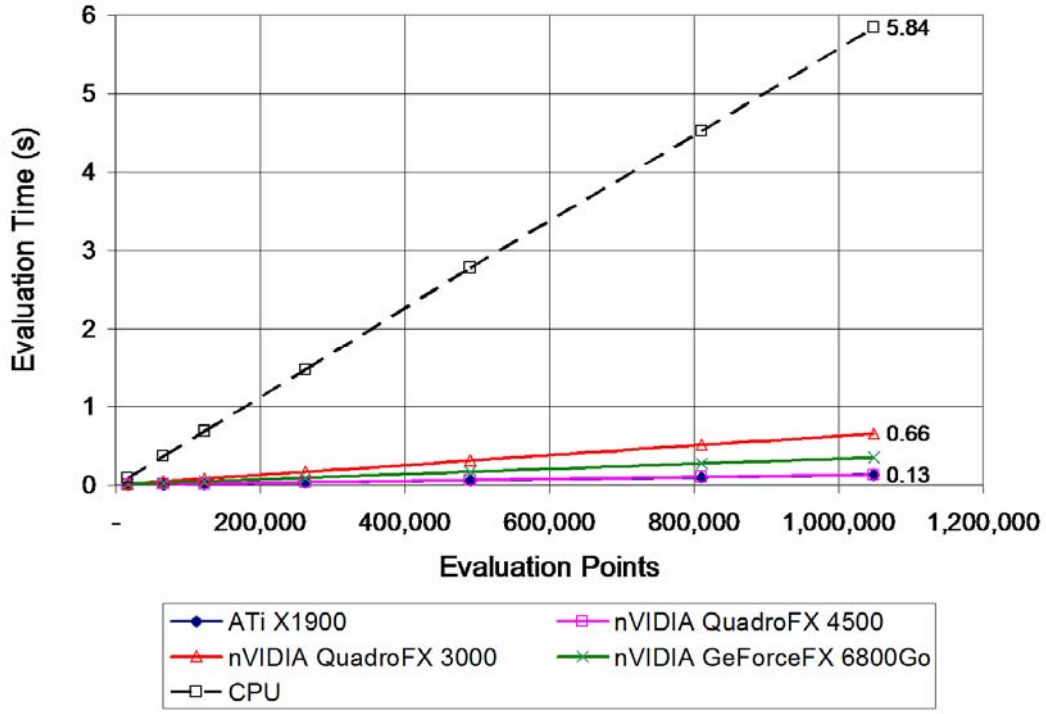

Figure 9: NURBS surface evaluation timings

We were not able to get timing results for evaluating T-spline surfaces as the graphic card had limitations on the number of instructions it can support. However, we can overcome this problem by splitting the fragment program into smaller pieces and then execute each individual part sequentially.

CONCLUSIONS

The evaluation of NURBS surfaces on the GPU shows tremendous promise for rendering large surfaces interactively. The rendered surfaces can also be manipulated in real time. T-splines on the other hand requires much more processing than NURBS. The limitations of graphics hardware prevent the interactive rendering of T-splines on graphics cards. Future work will focus on other methods for evaluating T-splines on the GPU.

REFERENCES

SEDERBERG, T. W., ZHENG, J., BAKENOV, A. AND NASRI, A. 2003, *T-Splines and T-NURCCs*, ACM Trans. Graph. 22, 3, 477–484.

PIEGL, L. A., AND TILLER, W. 1997. *The NURBS Book*, Version 1.2, second ed. Springer.