Direct3D11 Tessellation

Tianyun Ni
tni@nvidia.com

February 2009

Document Change History

Version	Date	Responsible	Reason for Change
1.0	03/20/10	tni	Initial release

1. Abstract

This sample implements a recent scheme on efficiently approximating Catmull-Clark subdivision surfaces[4]. The sample tessellates an animated character (a monster frog), a static environmental object (a tree), and the pebble road.

2. Introduction

Catmull-Clark subdivision has become a standard modeling tool for decades. A subdivision surface starts with a coarse polygon mesh, constructed by an artist. This mesh approximates the desired surface and consists of arbitrary combination of triangles and quads. The traditional recursive Catmull-Clark implementation requires at least 4 refinement steps, which indicates multiple passes on the modern GPU and large memory bandwidth to pass through the intermediate refined meshes. A direct patch representation is desired instead. Last year [1,2,3,4] provide such solutions. The general algorithm is that each triangle/quad (Figure 1.a) is converted to a polynomial patch representation.

There are three different facets in a coarse mesh:

- a regular quad: all 4 vertices have 4 neighbors
- a irregular quad: at least one vertex doesn't have 4 neighbors
- a triangle

A regular quad is converted into a bi-cubic patch by the standard B-spline to Bezier conversion rules (Figure 1 b). An irregular quad can be converted to

- a Bicubic patch (Figure 1 c.1) and a pair of tangent patches with 36 control points (Figure 1 c.2) [1], short as Bezier ACC.
- a Pm patch with 24 control points (Figure 1 d) [2, 3], short as Pm ACC.
- a Gregory patch with 20 control points (Figure 1 e) [4], short as Gregory ACC.

Both Gregory ACC and Pm ACC support triangular facets too. Allowing both triangles and quads reduces artists' workload and also avoids over-tessellated meshes. Among all ACC schemes, Gregory ACC [1] is the most efficient one for DX11 tessellation pipeline.

This sample shows you how to implement Gregory ACC on such an advanced tessellation pipeline for both animated characters and static environmental objects. In some cases, a simple linear tessellation scheme is good enough. This sample also demonstrates this scenario using an example of tessellating a pebbled stone pavement.

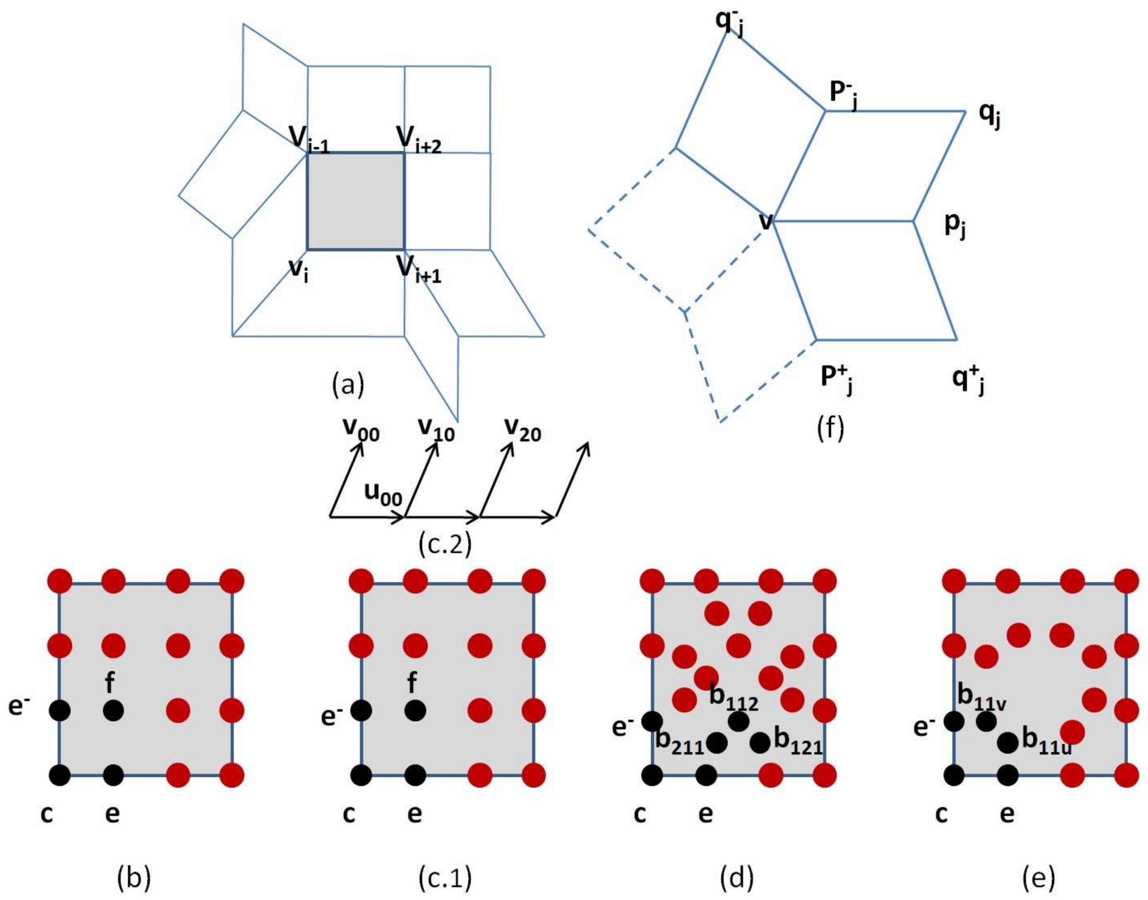

Figure 1

3. DirectX11 Implementation

DX11 pipeline is shown in Figure 2. The discussion will be emphasized on data flow of the first three shaders (Figure2 right).

A single rendering pass flows through the graphics pipeline in the following steps:

1. The input to the pipeline is a coarse input mesh, which consists of quads and possibly triangles. The vertices and their connections are defined in vertex and

index buffers. The patch primitive specifies a face and its one-ring neighborhood (Figure 1, top left).

2. The vertex shader transforms the vertices if needed
3. The hull shader is used for computing control points.
4. The tessellator generates domain points and triangles that are based on the partitioning style and tessellation factor.
5. The domain shader is used to compute surface position and normal. Applying Displacement mapping (DM) here too.
6. Finally, the pixel shader fetches the normal after DM and then produces a lit pixel.

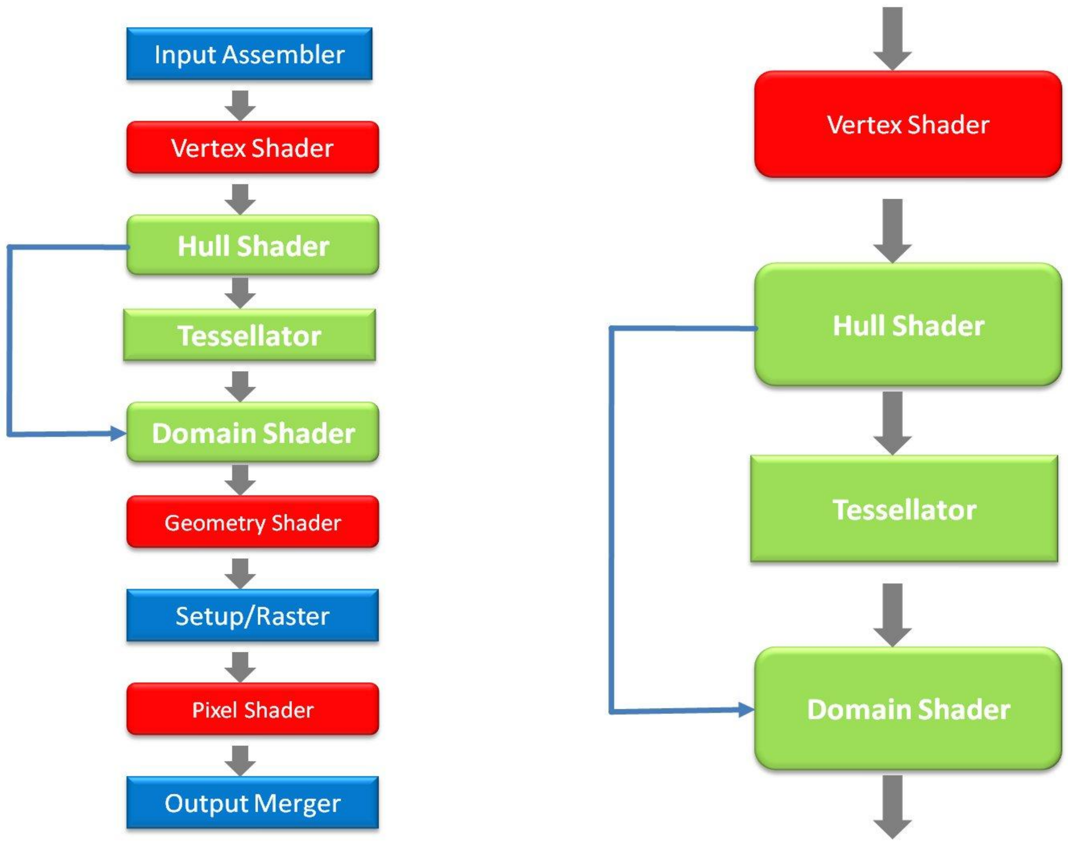

Figure 2. Tessellation Pipeline

4. Running the Sample

4.1 Adjust Tessellation Factors

- Static TFs: manually set a tessellation factor for all edges. To see the input mesh by setting the value to one.
- Turn adaptive TFs on: a tessellation factor per edge is chosen at run time in the hull shader based on a combined metric. This metric includes the distance factor from a camera to each edge, as well as the gradient of height values in the displacement map.

4.2 Toggle wireframe

All objects in the scene including a frog, a tree and a stone pavement are all tessellated through hardware tessellation. The tree and frog are tessellated using Gregory ACC scheme. The stone pavement is tessellated using the simplest tessellation scheme. Each new vertex is a linear interpolation of input vertices. Turn wireframe option on to see better how each object is tessellated on the fly and what the underlying tessellation pattern looks like for each patch.

4.3 Show Gregory patches

Highlight the areas where Gregory patches are used.

References

[1] "**Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches**", Charles Loop, Scott Schaefer, ACM Transactions on Graphics, Volume 27, issue 1, 2008.

[2] "**GPU Smoothing of Quad Meshes**", T. Ni, Y. Yeo, A. Myles, V. Goel, J. Peters, In IEEE International Conference on Shape Modeling and Applications, 2008.

[3] "**Fast Parallel Construction of Smooth Surfaces from Meshes with Tri/Quad/Pent Facets** ", A. Myles, T. Ni, J. Peters, Symposium on Geometry Processing, 2008.

[4] "**Approximating Subdivision Surfaces with Gregory Patches for Hardware Tessellation**", Charles Loop, Scott Schaefer, Tianyun Ni, Ignacio Castano, ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia), Vol. 28 No. 5, pages 151:1 - 151:9, 2009.