# Approximating Subdivision Surfaces with Gregory Patches for Hardware Tessellation

Charles Loop
Microsoft Research

Scott Schaefer
Texas A&M University

Tianyun Ni
NVIDIA

Ignacio Castaño
NVIDIA

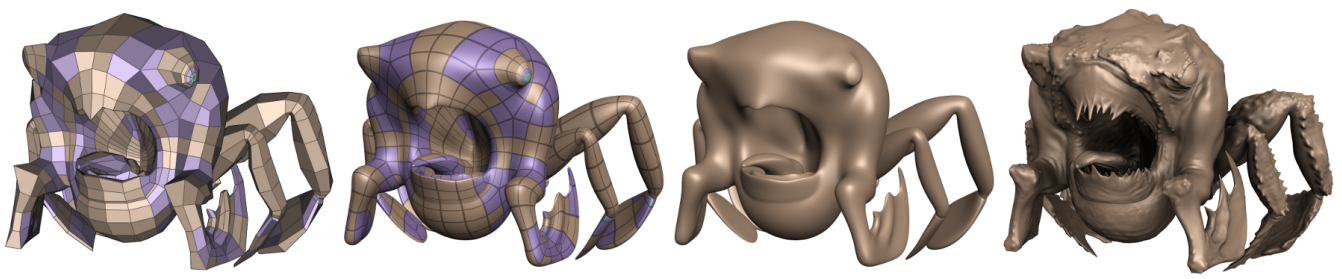

**Figure 1:** *The first image (far left) illustrates an input control mesh; regular (gold) faces do not have an incident extraordinary vertex, irregular quads (purple) have at least one extraordinary vertex, and triangular (green) faces are allowed. The second and third images show the parametric patches we generate. The final image is of the same surface with a displacement map applied.*

## Abstract

We present a new method for approximating subdivision surfaces with hardware accelerated parametric patches. Our method improves the memory bandwidth requirements for patch control points, translating into superior performance compared to existing methods. Our input is general, allowing for meshes that contain both quadrilateral and triangular faces in the input control mesh, as well as control meshes with boundary. We present two implementations of our scheme designed to run on Direct3D 11 class hardware equipped with a tessellator unit.

## 1 Introduction

Catmull-Clark subdivision surfaces [Catmull and Clark 1978] have become a standard for modeling free form shapes such as dynamic characters in movies and computer games. By adding displacement maps, we can create highly detailed shapes using a minimal amount of storage [Lee et al. 2000]. Tools such as ZBrush combine these two ideas to allow artists to edit models at multiple resolutions and automatically create low resolution control meshes and displacement maps.

Despite the prevalence of subdivision surfaces, realtime applications such as games predominately use polygon models to represent their geometry. The reason is understandable as GPU's are designed to accelerate polygon rendering and do so well. Yet, in many cases, subdivision surfaces are already part of the content creation pipeline for these applications. These surfaces are used in non-realtime parts of production such as cut-scenes, but their realtime counter parts are simplified polygon models of these high resolution characters. Ideally, we could use a high resolution polygon model extracted at a high level of subdivision to better approximate the character. However, this approach has a number of problems:

- Animation requires updating a large number of vertices each frame using bone weights or morph targets, consuming computational resources and harming performance.
- Faceting artifacts occur, due to the static nature of the polygon mesh connectivity.
- Large polygon meshes consume significant disk, bus, and network resources to store and transmit.

Given that these subdivision surfaces already exist, we could simplify the content creation pipeline by skipping the precomputed, fixed polygonalization step.

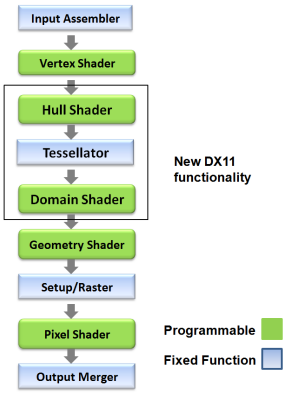


**Figure 2:** *The Direct3D 11 graphics pipeline.*

To address this issue, API designers and hardware vendors have added a new *tessellator unit* to the graphics pipeline in Direct3D 11 [Drone et al. 2008]. This change adds new programmable stages, the hull shader and the domain shader, to the graphics pipeline that

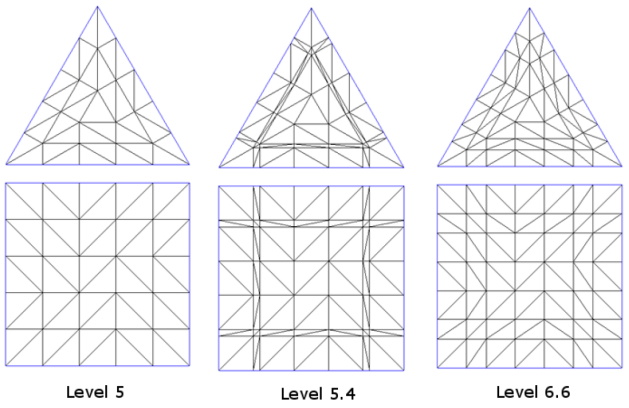


**Figure 3:** *Example tessellations produced by the tessellator unit for triangle and quad patches.*

enable the GPU to tessellate arbitrary parametric surfaces (see Figure 2). The *hull shader* takes a user defined patch primitive as input. Typically this input will consist of an irregular list of indices referencing the control mesh points needed to construct a patch. The job of the hull shader is to take this input, that may be irregular depending on the valence of the vertices, perform whatever geometric processing is required, and output a regular patch that is independent of valence and efficient to evaluate.

The tessellator unit takes this regular patch as input, as well as fractional tessellation factors along the edges of the patch, to automatically construct a triangulated sampling pattern for the patch. The fractional tessellation factors specify the number of vertices the tessellator should produce on each edge of the patch (fractional tessellation factors morph between integer levels to avoid popping artifacts when detail levels change). Figure 3 shows a set of example tessellations using the same tessellation factor per edge. These tessellation factors may be uniform, dependent on geometric complexity or even distance to the viewer and are the mechanism by which Direct3D 11 supports level of detail tessellations of these surfaces.

To evaluate a patch, the tessellator unit calls the *domain shader* with the patch data constructed in the hull shader along with the barycentric coordinates of each vertex in the tessellation. The domain shader uses this information to emit a vertex for the patch. The tessellator uses these vertices and the connectivity it generates to construct the patch. Using this paradigm, Direct3D 11 can support hardware tessellation of nearly any parametric surface.

The advantage of the tessellator unit is that we need only store and animate the control mesh of the subdivision surface, saving computational resources. Furthermore, only the small control mesh needs to be sent to the GPU to generate polygons saving bandwidth to the GPU. Moreover, we can tessellate these models dynamically on the GPU to avoid faceting artifacts common with polygonal models.

However, directly tessellating subdivision surfaces can be slow [Loop and Schaefer 2008] since patches with one or more extraordinary vertex (valence not equal to 4) are composed of an infinite set of polynomials. Therefore, we present a method that approximates the subdivision surface by replacing these irregular patches with a single rational patch that joins with $G^1$ continuity to the surrounding patches. In the regular case, our construction reproduces the Catmull-Clark surface exactly and provides good approximations of the Catmull-Clark surface over patches containing extraordinary vertices. In some cases, artists do not create surfaces that are entirely quadrilateral and need to use a small number of triangles on the surface. Our technique is also general enough to work with mixed quad/triangle surfaces and surfaces with boundary. Finally, our method generates a very small number of control points and reduces memory bandwidth bottlenecks on the GPU, resulting in better performance than previous methods.

## 2 Previous Work

Because subdivision surfaces can be composed of an infinite set of polynomials patches, they can be difficult to evaluate on the GPU. Bolz and Schröder [2002] precompute samples of the subdivision basis functions for various valences and topological configurations to quickly evaluate these surfaces using a GPU. Shiue et al. [2005] take a different approach and do not directly evaluate the subdivision surface but provide a method for performing subdivision directly on the GPU. However, both of these methods restrict the tessellation patterns to binary subdivision, which is incompatible with the fractional tessellation patterns and evaluation algorithms necessary for GPU tessellation with Direct3D 11 [Drone et al. 2008].

Using the framework provided by Halstead et al. [1993], Stam [1998] presented a method to exactly evaluate Catmull-Clark surfaces at arbitrary parametric points. However, this method only operates on quadrilateral patches with at most one extraordinary vertex. To apply Stam's method to surfaces with triangles or patches with more than one extraordinary vertex, we must subdivide the surface up to two times to provide sufficient separation of extraordinary vertices. In principle, these subdivision steps could be carried out on the GPU using DirectX 11 Compute Shaders [Boyd 2008]. Doing so will increase the size of the patch queues fed to the hull shader by a factor of 4 or 16. If such refinement is not performed everywhere but only locally when needed, then 'T' junctions will be present that are incompatible with watertight tessellation. In addition to the overhead of performing the extra subdivisions and dealing with considerably larger patch queues, Stam's evaluation technique also requires an *eigen space transform*. This transform amounts to a $(2n + 8) \times (2n + 8)$ matrix/control point product in the hull shader where $n$ is the number of quads containing the extraordinary vertex. The domain shader must also contain some logic to determine which of the three sets of eigen basis functions are needed, requiring a total of $16(2n + 8)$ texture reads to fetch. Each of these bicubic functions must be evaluated and multiplied by a corresponding eigen value (raised to an appropriate power) and eigen space control point, then summed to generate the final surface value. While performing 'exact' subdivision surface evaluation using Stam's technique is feasible, the resource requirements are considerable.

For these reasons, many researchers have investigated methods for approximating Catmull-Clark surfaces with parametric patches that are easier and faster to evaluate. To this end, Loop and Schaefer [2008] developed approximate Catmull-Clark (ACC) patches, which were extended to creased subdivision surfaces by Kovacs et al. [Kovacs et al. 2009]. These patches approximate the geometry and normal field of the subdivision surface separately and require 25 control points to perform evaluation. Since the GPU will almost always be bounded by memory bandwidth (fetch limited) for these computations, reducing the number of control points necessary for evaluation can lead to a significant performance improvement.

Myles et al. [2008b] also approximate Catmull-Clark surfaces such that the ordinary patches are identical to the Catmull-Clark surface. For patches containing one or more extraordinary vertices, the authors create a bi-degree 5 patch encoded as a perturbation of a bi-degree 3 patch. This patch creates a smooth join to neighboring patches and can be represented using 32 control points. Later Ni et

al. [2008] created smooth surfaces by replacing patches containing extraordinary vertices with multiple polynomial patches encoded as a single patch for the purposes of GPU evaluation. This method creates smooth surfaces and uses 24 control points per patch.

While most Catmull-Clark surfaces are composed mainly of quadrilaterals, artists occasionally add triangles to the control mesh. These triangles allow the artist to add/delete parameter lines on the shape [Myles et al. 2008a] or provide a more natural parameterization of the surface [Stam and Loop 2003]. However, the techniques mentioned previously [Loop and Schaefer 2008; Myles et al. 2008b; Ni et al. 2008] operate on surfaces containing only quads. While we could perform one step of Catmull-Clark subdivision to create a control mesh with only quadrilateral faces, we increase the number of patches by a factor of 4 and reduce the parallel nature of the computation on the GPU. Since the tessellator in Direct3D 11 supports both triangle and quad parameterizations, we desire a method that can approximate Catmull-Clark surfaces composed of both quads and triangles to avoid this extra splitting operation.

Vlachos et al. [2001] and Boubekuer and Alexa [2008] both provide methods for constructing the appearance of a smooth surface from triangles. However, neither of these methods are specific to Catmull-Clark surfaces and cannot reproduce the ordinary patches of the Catmull-Clark surface. Myles et al. [2008a] extended the work of Ni et al. [2008] to create Pm-patches and include surfaces with both triangle and pentagonal regions of the shape. Their method requires 19 control points per triangle and 25 control points per quad. In contrast, we use quadrilateral [Gregory 1974] and triangular [Longhi 1987] Gregory patches to approximate the shape of Catmull-Clark surfaces. Our technique requires at most 20 control points to perform evaluation (triangle patches require 15) and we show that this technique leads to a significant improvement in speed over previous work.

Peters [2000] also provided a method for "patching" a Catmull-Clark surface using bicubic patches, which could be used to approximate the Catmull-Clark surface. However, this technique required large separation of extraordinary vertices, which necessitates a preprocessing step of one or two levels of subdivision before applying the algorithm. This preprocessing increases the number of patches by up to a factor of 16 and suffers from many of the same problems that exact evaluation using Stam's algorithm does.

Chiyokura and Kimura [1983] also describe a method for building a smooth surface composed of quadrilateral patches using Gregory patches from a network of curves that meet with tangent plane continuity. However, their work considered a more general setting than ours in that the tangents specified by the user could be arbitrary. This generality forced the authors to lower the degree of various cross-boundary derivatives in order to generate a $G^1$ surface. In contrast, our method is designed specifically for Catmull-Clark subdivision surfaces. The tangent vectors are not arbitrary and the surfaces we construct are more flexible.

# 3 Patch Construction

Over regular regions of the surface (quadrilateral patches with no extraordinary vertices), Catmull-Clark subdivision surfaces are equivalent to $C^2$ tensor product bicubic B-splines. These regions are easy to evaluate at any parameter value since the B-spline basis functions are polynomials. However, subdivision surface patches containing one or more extraordinary vertices or non-quadrilateral patches are composed of an infinite set of polynomials. These regions can be difficult and expensive to evaluate inside the domain shader.

In our method, we keep the ordinary regions of the surface identical to that of the Catmull-Clark surface. Over each irregular patch, we replace each patch by a single quadrilateral or triangular Gregory patch. We place no restriction on the number of extraordinary vertices per patch, hence our method generates only one patch per face in the control mesh of the subdivision surface.

## 3.1 Gregory Patches

Gregory patches are a modified form of traditional tensor product and triangular polynomial patches. This modification was introduced by Gregory [1974] to overcome the difficult problem of ensuring that a collection of patches, sharing a vertex and pairwise sharing edges, meet with tangent plane continuity. Unlike polynomial patches, Gregory patches have the property that mixed partial derivatives need not match; that is, $\frac{\partial^2}{\partial u \partial v} \neq \frac{\partial^2}{\partial v \partial u}$ at patch corners. This property manifests itself in Bézier form as special control points that are functions of the bivariate parameter, taking on a (possibly) different position for every domain point; in particular along different patch boundaries. This makes the construction of piecewise tangent plane smooth surfaces over arbitrary 2-manifold tessellations possible with (effectively) lower order patches than can be achieved with polynomials. There are caveats of course; Gregory patches possess singularities (zero denominator) at patch corners, and the surface itself is rational with complicated tangent functions. From a practical standpoint however, these difficulties are not limitations and Gregory patches are well suited to solve the problem at hand.

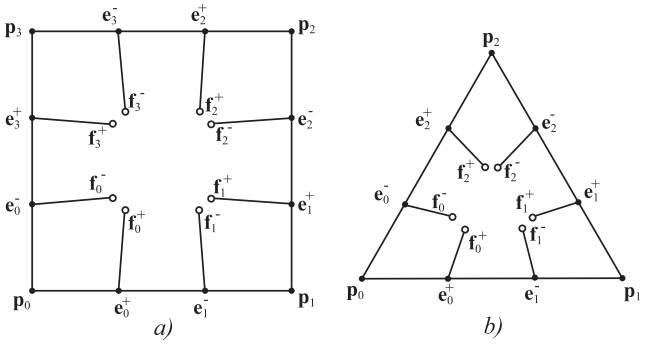


**Figure 4:** *The control point label scheme for a) a tensor product, and b) a triangular Gregory patch.*

The bicubic Bézier form of a quadrilateral Gregory patch has 20 control points labeled according to Figure 4a. We write the patch

$$Q(u,v) \;=\; \mathbf{B}^3(u) \cdot \mathbf{G} \cdot \mathbf{B}^3(v),$$

where $\mathbf{B}^d(\cdot)$ are degree $d$ Bézier basis functions and

$$\mathbf{G} = \begin{bmatrix} \mathbf{p}_0 & \mathbf{e}_0^- & \mathbf{e}_3^+ & \mathbf{p}_3 \\ \mathbf{e}_0^+ & \mathbf{F}_0 & \mathbf{F}_3 & \mathbf{e}_3^- \\ \mathbf{e}_1^- & \mathbf{F}_1 & \mathbf{F}_2 & \mathbf{e}_2^+ \\ \mathbf{p}_1 & \mathbf{e}_1^+ & \mathbf{e}_2^- & \mathbf{p}_2 \end{bmatrix},$$

where

$$\mathbf{F}_0 = \tfrac{u\mathbf{f}_0^+ + v\mathbf{f}_0^-}{u+v}, \qquad \mathbf{F}_1 = \tfrac{(1-u)\mathbf{f}_1^- + v\mathbf{f}_1^+}{1-u+v},$$
$$\mathbf{F}_2 = \tfrac{(1-u)\mathbf{f}_2^+ + (1-v)\mathbf{f}_2^-}{2-u-v}, \qquad \mathbf{F}_3 = \tfrac{u\mathbf{f}_3^- + (1-v)\mathbf{f}_3^+}{1+u-v},$$

are the special control points whose values vary with $u$ and $v$.

The triangular Gregory patch we use is a cubic-quartic hybrid; it is a quartic Gregory triangle with cubic boundary curves. The labeling

of our triangular patch control points is shown in Figure 4b. We write this triangular Gregory patch as

$$\begin{aligned} T(u,v,w) = {} & u^3\,\mathbf{p}_0 + v^3\,\mathbf{p}_1 + w^3\,\mathbf{p}_2 \\ & + 3uv(u+v)\left(u\,\mathbf{e}_0^+ + v\,\mathbf{e}_1^-\right) \\ & + 3vw(v+w)\left(v\,\mathbf{e}_1^+ + w\,\mathbf{e}_2^-\right) \\ & + 3wu(w+u)\left(w\,\mathbf{e}_2^+ + u\,\mathbf{e}_0^-\right) \\ & + 12uvw\left(u\,\mathbf{F}_0 + v\,\mathbf{F}_1 + w\,\mathbf{F}_2\right), \end{aligned}$$

where

$$\mathbf{F}_0 = \tfrac{w\mathbf{f}_0^- + v\mathbf{f}_0^+}{v+w}, \quad \mathbf{F}_1 = \tfrac{u\mathbf{f}_1^- + w\mathbf{f}_1^+}{w+u}, \quad \mathbf{F}_2 = \tfrac{v\mathbf{f}_2^- + u\mathbf{f}_2^+}{u+v},$$

are the varying control points. Note that for triangles, we use a barycentric parameterization where $u + v + w = 1$, so that the formulation is symmetric with respect to each edge.

To build the control points for our Gregory patches, we identify three distinct control point types corresponding to corners (e.g. $\mathbf{p}_0$), edges (e.g. $\mathbf{e}_0^+$), and faces (e.g. $\mathbf{f}_0^+$). All of the patch control points can be constructed from these cases.

### 3.2 Corner Points

Since Gregory patches are a simple, rational extension of Bézier patches, they share many of the same properties. In particular, the boundaries of the Gregory patches are cubic Bézier curves. Bézier curves interpolate their end-points so we place these control points (corresponding to the corner case $\mathbf{p}_0$) directly on the Catmull-Clark limit surface.

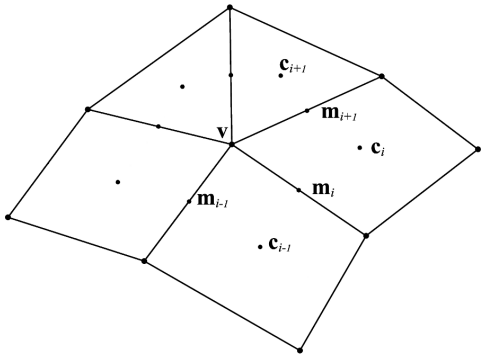


**Figure 5:** *The one-ring neighborhood of vertex* $\mathbf{v}$*, together with its* split *one-ring neighborhood. The points* $\mathbf{m}_i$ *are edge midpoints, and* $\mathbf{c}_i$ *are the average of the vertices of the* $i^{th}$ *face.*

The limit stencil for a subdivision surface corresponds to the dominant left eigenvector of its subdivision matrix [Halstead et al. 1993]. However, this stencil is only valid if the Catmull-Clark surface is composed solely of quads. We can generalize this stencil easily though. Let $\mathbf{m}_i$ be the midpoint of the $i^{th}$ edge surrounding the vertex $\mathbf{v}$ on the control mesh and $\mathbf{c}_i$ be the average of the vertices of the $i^{th}$ face according to Figure 5. The limit position is then

$$\mathbf{p}_0 = \frac{n-3}{n+5}\mathbf{v} + \frac{4}{n(n+5)}\sum_{i=0}^{n-1}\left(\mathbf{m}_i + \mathbf{c}_i\right).$$

### 3.3 Edge Points

Not only do we place the corner points of the Gregory patch $\mathbf{p}_0$ on the limit surface, but we also interpolate the tangent plane of the limit surface at that point. Similar to limit positions, the limit tangent stencil for a subdivision surface is given by the subdominant left eigenvector of the subdivision matrix. Again, we generalize this stencil for the limit tangent $\mathbf{q}$ to Catmull-Clark surfaces composed of both quads and triangles as

$$\mathbf{q} = \frac{2}{n}\sum_{i=0}^{n-1}\left(\left(1-\sigma\cos\left(\tfrac{\pi}{n}\right)\right)\cos\left(\tfrac{2\pi i}{n}\right)\mathbf{m}_i + 2\sigma\cos\left(\tfrac{2\pi i+\pi}{n}\right)\mathbf{c}_i\right),$$

where $\sigma = (4+\cos^2\left(\tfrac{\pi}{n}\right))^{-1/2}$. Since the derivative of the Gregory patch at its end-point is $3(\mathbf{e}_0^+ - \mathbf{p}_0)$, we can solve for $\mathbf{e}_0^+$

$$\mathbf{e}_0^+ = \mathbf{p}_0 + \tfrac{2}{3}\lambda\mathbf{q}.$$

The value for the scale factor $\lambda$ is free in our construction, but setting $\lambda$ to the subdominant eigenvalue of the Catmull-Clark surface gives good results and has the well known form

$$\lambda = \frac{1}{16}\left(5 + \cos\left(\frac{2\pi}{n}\right) + \cos\left(\frac{\pi}{n}\right)\sqrt{18 + 2\cos\left(\frac{2\pi}{n}\right)}\right).$$

### 3.4 Face Points

Our choice of $\mathbf{p}_0$ and $\mathbf{e}_0^+$ yields a surface that has a common tangent plane at each of the patch corners $\mathbf{p}_0$. To make the surface smooth everywhere, we must choose the face points $\mathbf{f}_0^+$ carefully. Our construction generalizes the tangent patch conditions described by Loop and Schaefer [2008]. For tangent plane continuity, we need that the versal (along the edge) and two transversal (pointing toward the patch interiors) derivatives are linearly dependent along the shared edge. We equate these derivatives to the vector fields

$$\begin{aligned} \mathbf{u}(t) &= \mathbf{B}^2(t)\cdot[\mathbf{u}_0,\mathbf{u}_1,\mathbf{u}_2], \\ \mathbf{v}(t) &= \mathbf{B}^3(t)\cdot[\mathbf{v}_0,\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3], \\ \hat{\mathbf{v}}(t) &= \mathbf{B}^3(t)\cdot[\hat{\mathbf{v}}_0,\hat{\mathbf{v}}_1,\hat{\mathbf{v}}_2,\hat{\mathbf{v}}_3], \end{aligned}$$

and require they be linearly dependent for $t \in [0,1]$, see Figure 6.

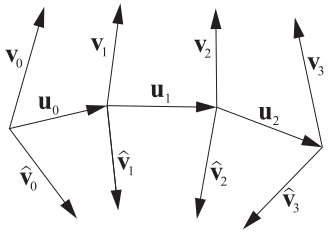


**Figure 6:** *Control vector label for the three tangent fields used in our tangent plane smoothness constraints .*

In our notation, $\mathbf{u}(t)$ is identical for both quad and triangle patches.

$$\begin{aligned} \mathbf{u}_0 &= 3(\mathbf{e}_0^+ - \mathbf{p}_0), \\ \mathbf{u}_1 &= 3(\mathbf{e}_1^- - \mathbf{e}_0^+), \\ \mathbf{u}_2 &= 3(\mathbf{p}_1 - \mathbf{e}_1^-). \end{aligned}$$

However, the definition of $\mathbf{v}(t)$ and $\hat{\mathbf{v}}(t)$'s control vectors vary depending on whether the Gregory patch is a quad or a triangle patch. In the following, setting $d = 4$ for triangles and $d = 3$ for quads,

we have

$$\mathbf{v}_0 = 3(\mathbf{e}_0^- - \mathbf{p}_0),$$
$$\mathbf{v}_1 = d(\mathbf{f}_0^+ - \mathbf{e}_0^+),$$
$$\mathbf{v}_2 = d(\mathbf{f}_1^- - \mathbf{e}_1^-),$$
$$\mathbf{v}_3 = 3(\mathbf{e}_1^+ - \mathbf{p}_1).$$

The control vectors of $\hat{\mathbf{v}}(t)$ are defined similarly to $\mathbf{v}(t)$.

The condition we satisfy for tangent plane continuity is

$$((1-t)c_0 - tc_1)\,\mathbf{u}(t) \;=\; \tfrac{1}{2}\left(\mathbf{v}(t) + \hat{\mathbf{v}}(t)\right), \qquad (1)$$

where $c_i = \cos\left(\frac{2\pi}{n_i}\right), i = 0, 1$; $n_0$ and $n_1$ are the number of patches shared at the corners corresponding to $t = 0$ and $t = 1$. Note that condition (1) is specialized to the case where the tangent vectors surrounding a vertex form an affine $n$-gon; we inherit this property by interpolating Catmull-Clark subdivision surface structure at patch corners.

Both sides of condition (1) are cubic polynomials; so this polynomial condition is equivalent to 4 scalar conditions found by equating the Bézier coefficients of each side, resulting in

$$c_0\mathbf{u}_0 = \tfrac{1}{2}\left(\mathbf{v}_0 + \hat{\mathbf{v}}_0\right) \qquad (2)$$
$$\tfrac{2}{3}c_0\mathbf{u}_1 - \tfrac{1}{3}c_1\mathbf{u}_0 = \tfrac{1}{2}\left(\mathbf{v}_1 + \hat{\mathbf{v}}_1\right) \qquad (3)$$
$$\tfrac{1}{3}c_0\mathbf{u}_2 - \tfrac{2}{3}c_1\mathbf{u}_1 = \tfrac{1}{2}\left(\mathbf{v}_2 + \hat{\mathbf{v}}_2\right) \qquad (4)$$
$$-c_1\mathbf{u}_2 = \tfrac{1}{2}\left(\mathbf{v}_3 + \hat{\mathbf{v}}_3\right) \qquad (5)$$

Conditions (2) and (5) will be satisfied by interpolating the Catmull-Clark limit tangents at endpoints. Condition (3) (and similarly (4)) will be satisfied by constructing a transversal vector $\mathbf{r}$ such that

$$\mathbf{v}_1 \;=\; \tfrac{2}{3}c_0\mathbf{u}_1 - \tfrac{1}{3}c_1\mathbf{u}_0 + \mathbf{r}$$
$$\hat{\mathbf{v}}_1 \;=\; \tfrac{2}{3}c_0\mathbf{u}_1 - \tfrac{1}{3}c_1\mathbf{u}_0 - \mathbf{r}$$

While $\mathbf{r}$ is arbitrary, we choose this vector to reproduce the ordinary case when $n = 4$. The vector $\mathbf{r}$ will be different for each face point and the corresponding vector for $\mathbf{f}_0^+$ is given by

$$\mathbf{r}_0^+ = \frac{1}{3}\left(\mathbf{m}_{i+1} - \mathbf{m}_{i-1}\right) + \frac{2}{3}\left(\mathbf{c}_i - \mathbf{c}_{i-1}\right).$$

If we now solve for the face point $\mathbf{f}_0^+$, we obtain

$$\mathbf{f}_0^+ \;=\; \frac{1}{d}\left(c_1\,\mathbf{p}_0 + (d - 2c_0 - c_1)\,\mathbf{e}_0^+ + 2c_0\mathbf{e}_1^- + \mathbf{r}_0^+\right). \qquad (6)$$

For irregular regions of the surface, this construction leads to tangent smooth surfaces. Over regular regions, we reproduce bicubic b-splines and the surface is $C^2$.

# 4 Implementation

We explored various implementation approaches for the best fit to the Direct3D 11 pipeline and present two methods that perform well. The choice of which method to use will be application and hardware vendor specific, as each has its own strengths and weaknesses. One of our implementations maps the patch construction to the vertex and hull shaders, while the other performs all computations in the hull shader freeing the vertex shader for other purposes, such as animation and deformations of the input mesh.

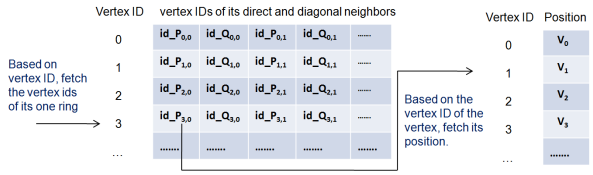


**Figure 7:** *The left texture contains the connectivity, where $P_{i,j}$ (or $Q_{i,j}$) is the vertex id of $j^{th}$ direct (or diagonal) neighbor for $i^{th}$ vertex. If the adjacent face is a triangle, no diagonal neighbor exists and we store the id: $-1$. The right texture stores the actual vertex positions, which will be dynamically updated during animation.*

## 4.1 Vertex Shader

Section 3 describes how we compute the control points of the Gregory patch using the one-ring of the face corresponding to the patch (the set of vertices from faces adjacent to the current face). Among the three control point types, most control points are exclusively influenced by a small subset of these vertices. Specifically, $\mathbf{p}$, $\mathbf{e}_i^+$, and $\mathbf{e}_i^-$ are only influenced by the vertex $\mathbf{v}$ and its one-ring neighborhood (the set of vertices whose faces contain $\mathbf{v}$). These vertex-based computations map well to the vertex shader. We pass the vertex position $\mathbf{v}$ to the vertex shader and fetch its one-ring from two textures (see Figure 7). The first texture contains the one-ring indices for each control mesh vertex; this texture is invariant under animation. The second texture stores the dynamic vertex positions. If the vertex has $n$ adjacent faces, we compute $n$ edge midpoints $\mathbf{m}_i$ and $n$ face centroids $\mathbf{c}_i$ where $i = 0 \ldots n-1$ and indexes the faces around the vertex. Each $\mathbf{p}, \mathbf{e}_i^+, \mathbf{e}_i^-, \mathbf{r}_i^+, \mathbf{r}_i^-$ is a function of $\mathbf{v}, \mathbf{m}_i, \mathbf{c}_i$. These control points (along with $\mathbf{r}_i^+, \mathbf{r}_i^-$) form the output from the vertex shader. The corner points and edge points are only computed once if the vertex post transform cache is unlimited. In practice, these points will be re-computed on a vertex cache miss. By carefully arranging vertices in the cache, we can reduce these redundant computations. Many optimizations, such as [Lin and Yu 2006; Forsyth 2006; Sander et al. 2007], exist for minimizing vertex cache misses. One simple solution is to use separate index buffers for regular and Gregory patches. In this way, the vertices processing the same shader are grouped more tightly.

## 4.2 Hull Shader

The input to the hull shader is a patch primitive that consists of 3 and 4 points for the triangle and tensor product cases respectively. Three/four threads are invoked to compute one Gregory patch sector that includes one corner point, two edge points and two face points. For each vertex of the patch, we find the index of the edge $j$ from the texture shown in Figure 8, which provides an offset into the vertex data to find the control points $\mathbf{e}_0^+$, and $\mathbf{e}_0^-$ as well as the vectors $\mathbf{r}_0^+$ and $\mathbf{r}_0^-$. Using the edge-points from adjacent vertices as well as the vectors $\mathbf{r}_0^+$ and $\mathbf{r}_0^-$, we also construct the face points by evaluating Equation 6. Because the patch primitive contains at most 4 vertices, each vertex in the primitive is transferred much less frequently compared to the patch primitive that consists of 32 points.

## 4.3 Alternative Hull Shader Stencil Approach

While the patch construction maps naturally to the vertex and hull shaders, it is also possible to perform all the computations in the hull shader so that the vertex shader can be freed for other purposes. The key observations for this approach are

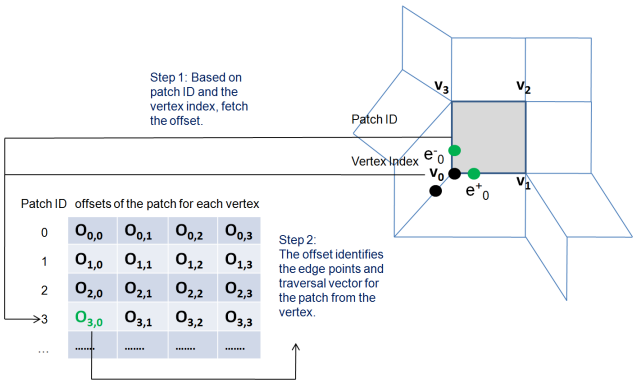


**Figure 8:** *2D texture that stores the offsets of the current edge point for each vertex in the patch primitive.* $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ *are the vertices of a quad patch primitive (only three vertices for a triangle).* $O_{i,j}$ *has the rotation offset of* $i^{th}$ *patch from the* $j^{th}$ *vertex.*

- each control point is a linear combination of the vertices in the one-ring of the face.
- for all the patches sharing the same connectivity in their one-ring, the weights contributed from the vertices have a unique set of values (or stencil).

We first preprocess the input meshes to sort all patches based on their connectivity type (i.e. valence of each of the 3 (triangles) or 4 (quads) vertices of the patch). Given a connectivity type, we precompute the weight of each vertex that contributes to a Gregory control point using the formulas for $\mathbf{p}$, $\mathbf{e}$, or $\mathbf{f}$. The results are stored in an array where each element associates with a vertex-weight pair. If there are $m$ connectivity types, $k$ control points per patch, and $\ell$ vertices in the one-ring of the patch, the total number of weights computed would be $mk\ell$. We pack this data into a stencil texture.

Given a list of the $\ell$ vertices in the one-ring of the patch (the hull shader limits this number to 32), we fetch the necessary set of weights from the stencil texture using the connectivity id and control point id. Using these weights, we compute the 20 (or 15 for triangle patches) control points for the Gregory patch as a weighted sum of all the vertices in the input patch primitive. Although this procedure maps very well to the SIMD nature of the hardware, it has the following inefficiencies that our first method avoids.

- Since each control point is only influenced by a subset of the vertices of the input patch, many of the weights in the stencil will be zero. The weighted sum we compute includes all input vertices, resulting in many unnecessary products and texture samples. This issue can be diminished by precomputing a boolean mask that indicates what products can be skipped and avoid the corresponding texture samples.
- Corner and edge control points shared by adjacent patches are evaluated independently. Unlike in Section 4.1, there are no opportunities for sharing intermediate results.
- Finally, in order to guarantee consistent evaluation, a global ordering of the vertices is required and an additional offset needs to be sampled in order to compute the sum in an order that is consistent for all patches.

On the other hand, the first method requires the vertex shader for patch construction and we must add an additional rendering pass for animation/deformation. Since each method has its own advantages/disadvantages, we present both approaches for completeness.

### 4.4 Domain Shader

Once the hull shader generates the control points, the tessellator calls the domain shader to evaluate the patch at various parametric coordinates. We customize two domain shaders for evaluating the irregular tensor product patches and triangular patches. However, directly evaluating these patches in their Gregory form could be expensive due to the rational nature of these patches. Therefore, we transform the patch into its polynomial Bézier form by evaluating the formula for the $\mathbf{F}_i$ (see Section 3.1). The caveat that this evaluation results in a divide by zero at patch corners is easily handled using a conditional assignment; the SIMD branch divergence this creates is negligible.

The transformation to polynomial form allows us to treat the Gregory patch as a standard Bézier patch to evaluate the position and normal of the surface simultaneously using the DeCasteljau algorithm. Notice that while the position is correct when evaluated in this fashion, the tangent vectors, and hence the normal, are not. The reason is that Gregory patches are rational functions with complicated derivatives. However, our approximate normals will be exact along the boundaries of the patch and continuous over the interior. In general these normals are much faster to compute than the true rational derivatives and we have not observed any artifacts from this approximation. From this surface, we can also create highly-detailed models by offsetting these vertices using a displacement map.

### 4.5 Water-tight Evaluation

Water-tight evaluation of the surface is important both for the position (to avoid pixel dropouts in the surface) and for the normal when using displacement mapping (to avoid cracks/holes). Unfortunately, due to the nature of floating point computations, addition and multiplication are not associative or distributive. Furthermore, additions and products are usually performed using multiply-adds, which increase the accuracy of the result but make the resulting expression non-commutative. Therefore, we must take great care in order to guarantee that operations are performed in an order that produces a consistent result. Such consistency arises in

- control point evaluation. We derive the shared control as the weighted sum of a patch's one-ring in a predefined order. As long as the labeling/ordering of one-ring neighborhood is always consistent, the computations of the same control point leads to a single value.
- patch evaluation. We perform patch evaluation consistently by either consistent parametric orientation or symmetric evaluation along the shared edge.

Water-tight evaluation of the normals is more complex as we compute the normal by taking the cross product of a pair of tangent vectors. If the normal at a point shared by multiple patches is not the same, we might see holes with displacement mapping. However, we can easily achieve water-tight normals using edge/corner ownership to ensure there is only one normal at the shared point.

### 4.6 Boundary Cases

We can also modify our construction to handle meshes with boundaries as well. For corner points $\mathbf{p}$ contained by more than one patch, we place $\mathbf{p}$ using the limit stencil for cubic B-spline curves at a ratio of $1:4:1$ of the adjacent boundary vertices. If $\mathbf{p}$ is contained by exactly one patch, we interpolate the vertex of the control mesh and set $\mathbf{p} = \mathbf{v}$. Edge points $\mathbf{e}_0^+$, $\mathbf{e}_{k-1}^-$ (where $k$ is the number of edges adjacent to $\mathbf{v}$) along the boundary are likewise modified to use a ratio of $2:1$ of the control mesh vertices at the end-points of

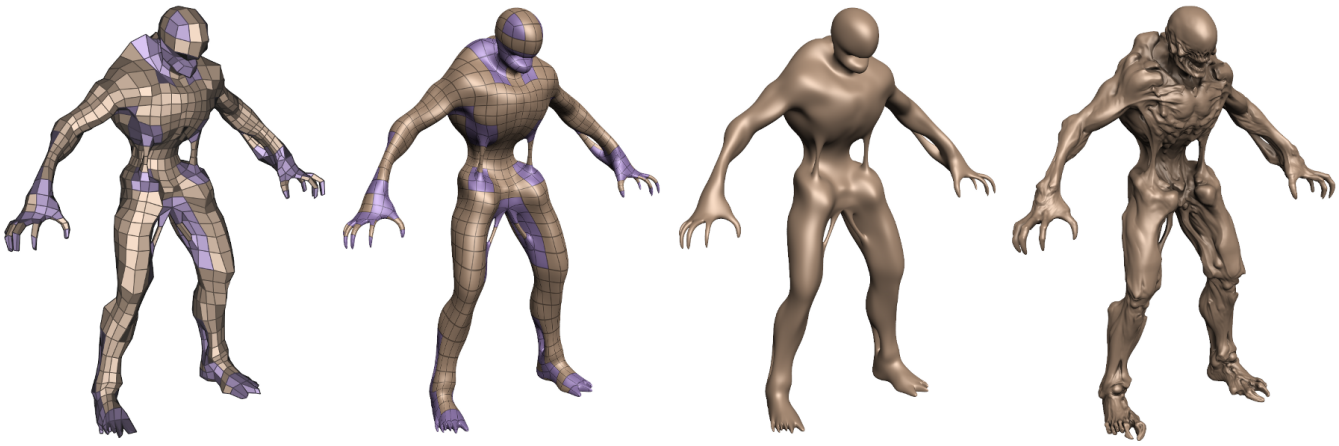

**Figure 9:** *From the left: the control mesh, the patch structure, the surface produced by our method, and the surface with a displacement map.*

the edge. We create interior edge points using the tangent stencils for Catmull-Clark surfaces with boundaries described by Biermann et al. [2000]. For face points $\mathbf{f}_0^+$ and $\mathbf{f}_{k-1}^-$ we reflect interior control points across the boundary to create $\mathbf{r}_0^+$ and $\mathbf{r}_{k-1}^-$.

## 5 Results

Since at the time of this writing Direct3D 11 class hardware is still under development, we implemented the three most competitive schemes on a hardware emulator that supports all Direct3D 11 functionality. Our preliminary results indicate that Gregory patch evaluation is up to 20% faster than other methods including [Loop and Schaefer 2008], [Ni et al. 2008] and [Myles et al. 2008a]. This difference is mainly due to the fact that our method requires the domain shader to fetch fewer control points. We require only 20 control points for evaluation while both Pm-patches and ACC patches require 25 control points. Table 1 shows timing data for our DX11 implementation on Figure 1, which has 508 irregular patches. This performance data does not include CPU overhad (specifically PC memory and CPU PCI express overhead) or driver-side overhead. We made two observations from the results of the hardware emulator. First, our scheme outperforms ACC and Pm-patch in all cases. Second, the two-stage approach (described in Section 4.1 + 4.2) works well when the tessellation level is low while the stencil approach (described in Section 4.3) is well suited to higher tessellation levels. Due to lack of cache optimizations in our current two-stage approach, we expect better performance after optimizations.

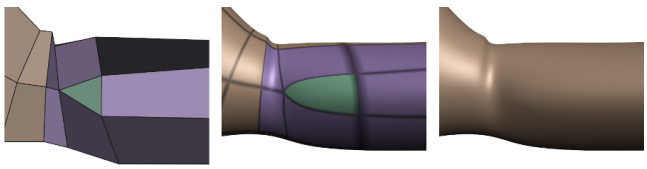

**Figure 10:** *A close-up of a portion of the model from Figure 1 showing a tri/quad region and our surface on the right.*

We also show several examples of our surfaces in Figures 1, 9, and 12. Our construction can handle surfaces composed entirely of quads like the examples in Figure 9 and the first 4 columns of Figure 12. Figure 1 and columns $5-7$ of Figure 12 also demonstrate surfaces composed of both quads and triangles. Figure 10 shows a zoomed in portion of Figure 1 containing a triangle/quad region of the surface and the smooth transition produced by our method. Furthermore, our surfaces may also contain boundaries as demonstrated in column 4 of Figure 12. Finally, since our evaluation is water-tight, both in position and normal, these surfaces are suitable for displacement mapping as shown in Figures 1 and 9.

| | Stencil | | Two-stage | |
|---|---|---|---|---|
| | ACC | Our Scheme | Pm-patch | Our Scheme |
| N=10 | 0.104 | 0.0913 | 0.1327 | 0.1047 |
| N=6 | | 0.053 | | 0.0485 |

**Table 1:** *Using the shape from Figure 1, we construct each irregular quad on the emulator and then evaluate on a grid of size N $\times$ N. Performance measured in ms. Refer to stencil and two-stage approaches in section 4.3, and 4.1 + 4.2 respectively.*

In addition, we developed our shader code to run on Direct3D 10 hardware by substituting the hull shader with the geometry shader. We emulate the tessellator unit using a generic GPU refinement kernel [Boubekeur and Schlick 2007] and perform Instanced Tessellation using one extra rendering pass. In the first pass, the geometry shader streams out the control points for the patches. In the following pass, we instance the parametric domains and send each barycentric coordinate to the vertex shader as an input vertex. The vertex shader then performs the job of the domain shader in Direct3D 11 and evaluates the tessellated patch. Instancing is the most efficient method for tessellation on current hardware and reduces CPU overhead in terms of the number of draw calls, state changes, and buffer updates. Table 2 shows the performance of our Direct3D 10 implementation in terms of frames per second.

| | ACC | | Pm-patch | | Our Scheme | |
|---|---|---|---|---|---|---|
| N = | 17 | 25 | 17 | 25 | 17 | 25 |
| monster frog | 365 | 186 | 377 | 208 | 398 | 213 |
| bigguy | 348 | 176 | 359 | 186 | 370 | 194 |

**Table 2:** *Performance comparison. Instanced Tessellation with GPU patch construction measured in frames per second on NVIDIA GTX260, and Intel Core 2 Duo CPU, 3GHz. Monster frog contains* 1292 *patches and bigguy* 1754 *patches. We construct each patch on the GPU every frame and then evaluate on a grid of size N $\times$ N.*

In our Direct3D 10 implementation, the domain sample points are known a priori, so our performance advantage over Pm-patches is not as pronounced as in Direct3D 11. With a tessellator unit in Direct3D 11, these sample points are not known ahead of time and will

| | ACC | Pm-patch | Our Scheme |
|---|---|---|---|
| monster frog | .146/.0148 | .079/.0111 | .090/.0108 |
| big guy | .096/.0109 | .072/.00783 | .082/.00710 |

**Table 3:** *Error as measured by Metro between different approximations and the Catmull-Clark surface. Error reported is RMS and of the form position error as a percentage of the bounding box diagonal of the model * $10^{-3}$ / error in radians of normal.*

be generated by the tessellator at runtime. Gregory patches fit better than Pm-patches in Direct3D 11 because Gregory patches do not require the domain transformations that Pm-patches do. Furthermore, our tests in Direct3D 11 indicate that the methods are bound by fetching control points and that fewer control points translates into larger gains on Direct3D 11 hardware.

Figure 11 compares the smooth quad mesh surfaces generated by various schemes with the Catmull-Clark limit surface using parametric correspondence between the surfaces. The triangle areas are not rendered since [Loop and Schaefer 2008] only works for quadrilateral meshes. At each parametric domain, we measure both the geometric distance and normal angle difference between the two surfaces. While parametric correspondence is important for texturing and especially for displacement mapping, we also measured the error between the surfaces using Metro [Cignoni et al. 1998], which approximates the Hausdorff distance between the surfaces. This distance is closer to perceptual distance between the surfaces and we measure the RMS error reported by Metro for both the position and normals (using the parameterization provided by the distance calculation) of the shapes, which is summarized in Table 3.

Since Pm-patches only interpolate the Catmull-Clark limit positions at the vertices of the control mesh, the normals vary more from the Catmull-Clark surface, which can be seen in Table 3. ACC patches use geometry patches to encode the location of the vertices and tangent patches to create a continuous normal field. At the vertices of the control mesh, ACC patches interpolate both the limit position and normals of the Catmull-Clark surface. However, this construction does not extend to triangle patches. In contrast Gregory patches handle both triangle and quad surfaces and interpolate both the limit positions and normals of the Catmull-Clark surface at the control vertices. In general, the approximation quality of our method exceeds ACC patches and is similar quality to that of Pm-patches but faster to evaluate.

## 6 Conclusions

We have presented a new method for approximating Catmull-Clark subdivision surfaces with Gregory patches that is suitable for programmable hardware implementation supporting automatic surface tessellation. As memory access is the bottleneck in dynamic patching schemes such as ours, we optimize the critical memory fetch requirements compared to previous work. Our experiments demonstrate comparable visual fidelity, and superior performance on both available hardware and a Direct3D 11 class hardware emulator.

The surfaces we generate show very little difference to their Catmull-Clark subdivision surface counterparts. With additional effort, we could improve this error. The degrees of freedom available in our construction, the scale factor $\lambda$ and transversal vector $\mathbf{r}$ could be optimized to minimize (some meaningful measure of) the difference with the Catmull-Clark surface. Our choices were made to keep the construction simple, while producing reasonable results.

One drawback that we can foresee for any *approximate subdivision surface* is the slight disparity artists may encounter between the surface displayed by their content creation tools and the one that is rendered in realtime. The solution is to use the approximate surfaces as part of content creation and we intend to make our source code public to make such a transition easier for tool vendors.

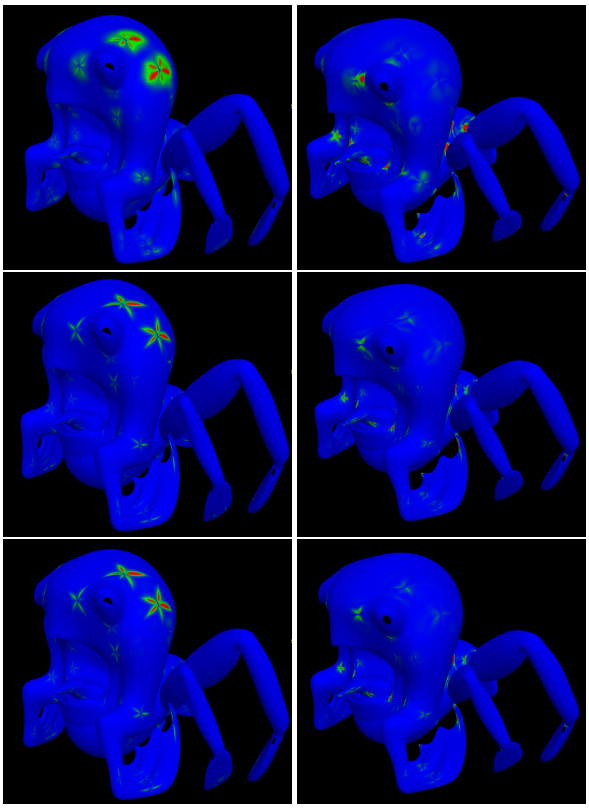

**Figure 11:** *From top to bottom, parametric error of ACC, Pm-patches and our method with respect to the Catmull-Clark limit surface. The position (column 1) and normal angle difference (column 2) to the limit surface of Catmull-Clark subdivision are shown with blue representing no error and red representing high error.*

**Acknowledgements**
We thank Bay Raitt and Mike Asquith from Valve Corporation and Kenneth Scott from id Software for providing some of the models depicted in this work. We also acknowledge Denis Kovacs for his early contributions in extending this work to triangular patches.

## References

BIERMANN, H., LEVIN, A., AND ZORIN, D. 2000. Piecewise smooth subdivision surfaces with normal control. In *proceedings of SIGGRAPH*, 113–120.

BOLZ, J., AND SCHRÖDER, P. 2002. Rapid evaluation of catmull-clark subdivision surfaces. In *Proceeding of the International Conference on 3D Web Technology*, 11–17.

BOUBEKEUR, T., AND ALEXA, M. 2008. Phong tessellation. *ACM Trans. Graph. 27*, 5, 1–5.

BOUBEKEUR, T., AND SCHLICK, C. 2007. Generic adaptive mesh refinement. In *GPU Gems 3*. Addison-Wesley, 93–104.

BOYD, C., 2008. Direct3D 11 Compute Shader. http://www.microsoft.com/downloads/details.aspx?FamilyId=9F943B2B-53EA-4F80-84B2-F05A360BFC6A.

**Figure 12:** *From top to bottom: the input mesh, the patch structure (regular patch in gold, irregular quad patch in purple, triangular patch in green), and our smooth surface. Columns 1-3 contain closed quadrilateral meshes. Column 4 is a quadrilateral mesh with boundary, and columns 5-7 contain closed tri/quad meshes.*


CATMULL, E., AND CLARK, J. 1978. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6, 350–355.

CHIYOKURA, H., AND KIMURA, F. 1983. Design of solids with free-form surfaces. *Computer Graphics 17*, 3, 289–298.

CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. 1998. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2, 167–174.

DRONE, S., LEE, M., AND ONEPPO, M., 2008. Direct3D 11 Tessellation. http://www.microsoft.com/downloads/details.aspx?FamilyId=2D5BC492-0E5C-4317-8170-E952DCA10D46.

FORSYTH, T., 2006. Linear-speed vertex cache optimisation. http://home.comcast.net/ tom_forsyth/papers/fast_vert_cache_opt.html.

GREGORY, J. 1974. Smooth interpolation without twist constraints. In *Computer Aided Geometric Design*, R. E. Barnhill and R. F. Riesenfeld, Eds. Acadmic Press, New York, 71–87.

HALSTEAD, M., KASS, M., AND DEROSE, T. 1993. Efficient, fair interpolation using catmull-clark surfaces. In *Proceedings of SIGGRAPH*, 35–44.

KOVACS, D., MITCHELL, J., DRONE, S., AND ZORIN, D. 2009. Real-time creased approximate subdivision surfaces. In *Proceedings of the symposium on Interactive 3D graphics*, 155–160.

LEE, A., MORETON, H., AND HOPPE, H. 2000. Displaced subdivision surfaces. In *Proceedings of SIGGRAPH*, 85–94.

LIN, G., AND YU, T. P. Y. 2006. An improved vertex caching scheme for 3d mesh rendering. *IEEE Transactions on Visualization and Computer Graphics 12*, 4, 640–648.

LONGHI, L. 1987. Interpolating patches between cubic boundaries. Tech. Rep. CSD-87-313, University of California.

LOOP, C., AND SCHAEFER, S. 2008. Approximating catmull-clark subdivision surfaces with bicubic patches. *ACM Trans. Graph. 27*, 1, 8:1–8:11.

MYLES, A., NI, T., AND PETERS, J. 2008. Fast parallel construction of smooth surfaces from meshes with tri/quad/pent facets. *Computer Graphics Forum 27*, 5, 1365–1372.

MYLES, A., YEO, Y. I., AND PETERS, J. 2008. Gpu conversion of quad meshes to smooth surfaces. In *SPM '08: ACM symposium on Solid and physical modeling*, 321–326.

NI, T., YEO, Y. I., MYLES, A., GOEL, V., AND PETERS, J. 2008. Gpu smoothing of quad meshes. In *SMI '08: IEEE International Conference on Shape Modeling and Applications*, 3–9.

PETERS, J. 2000. Patching catmull-clark meshes. In *Proceedings of SIGGRAPH*, 255–258.

SANDER, P. V., NEHAB, D., AND BARCZAK, J. 2007. Fast triangle reordering for vertex locality and reduced overdraw. *ACM Trans. Graph. 26*, 3, 89.

SHIUE, L.-J., JONES, I., AND PETERS, J. 2005. A realtime gpu subdivision kernel. *ACM Trans. Graph. 24*, 3, 1010–1015.

STAM, J., AND LOOP, C. 2003. Quad/triangle subdivision. *Computer Graphics Forum 22*, 1, 1–7.

STAM, J. 1998. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. *Computer Graphics 32*, Annual Conference Series, 395–404.

VLACHOS, A., PETERS, J., BOYD, C., AND MITCHELL, J. L. 2001. Curved pn triangles. In *Proceedings of the Symposium on Interactive 3D Graphics*, 159–166.