

Smooth Surfaces from 4-sided Facets

T. L. Ni, Y. Yeo, A. Myles, V. Goel and J. Peters

Abstract

We present a fast algorithm for converting quad meshes on the GPU to smooth surfaces. Meshes with 12,000 input quads, of which 60% have one or more non-4-valent vertices, are converted, evaluated and rendered with 9×9 resolution per quad at 50 frames per second. The conversion reproduces bi-cubic splines wherever possible and closely mimics the shape of the Catmull-Clark subdivision surface by c-patches where a vertex has a valence different from 4. The smooth surface is piecewise polynomial and has well-defined normals everywhere. The evaluation avoids pixel dropout.

1 Introduction and Contribution

Due to the popularity of Catmull-Clark subdivision [Catmull and Clark 1978], quad-meshes are common in modeling for animation. Quad meshes are meshes consisting of quadrilateral facets without restriction on the valence of the vertices. Any polyhedral mesh can be converted into a quad mesh by one step of Catmull-Clark subdivision, but a good designer creates meshes with the quad-restriction in mind so that no global refinement is necessary.

For real-time applications such as gaming, interactive animation and morphing, it is convenient to offload smoothing and rendering to the GPU. In particular, when morphing is implemented on the GPU, it is inefficient to send large data streams on a round trip to the CPU and back. Smooth surfaces are needed, for example, as the base for displacement mapping in the surface normal direction [Lee et al. 2000] (Fig 3).

For GPU smoothing, we distinguish two types of quads: ordinary and extraordinary. A quad is *ordinary* if all four vertices have 4 neighbors. Such a facet will be converted into a degree 3 by 3 patch in tensor-product Bézier form by the standard B-spline to Bézier conversion rules [Farin 1990]. Therefore, any two adjacent patches derived from ordinary quads will join C^2 . The interesting aspect is the conversion of the *extraordinary* quads, i.e. quads having at least one and possibly up to four vertices of valence $n \neq 4$. We present a new algorithm for converting both types of quads on the fly so that

1. every ordinary quad is converted into a bicubic patch in tensor-product Bézier form, Figure 1, (b);
2. every extraordinary quad is converted into a composite patch (short **c-patch**) with cubic boundary and defined by 24 coefficients (Figure 1, c);
3. the surface is by default smooth everywhere (Lemma 1);
4. the shape follows that of Catmull-Clark subdivision;
5. conversion and evaluation can be mapped to the GPU to render at very high frame rates (at least an order of magnitude faster than for example [Bunnell 2005; Shiue et al. 2005] on current hardware).

1.1 Some Alternative Mesh Smoothing Techniques on the GPU

A number of techniques exist to smooth out quad meshes. Catmull-Clark subdivision [Catmull and Clark 1978] is an accepted standard, but does not easily port to the GPU. Evaluation using Stam's approach [Stam 1998] is too complex for large meshes on the GPU.

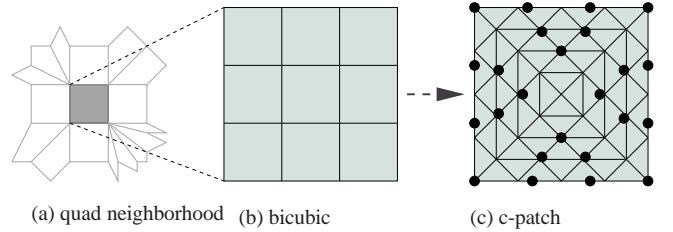


Figure 1: (a) A quad neighborhood defining a surface piece. (b) A bicubic patch with 4×4 control points. This patch is the output if the quad is ordinary, and used to determine the shape of a *c-patch* (c) if the quad is extraordinary. A *c-patch* is defined by 4×6 control points displayed as ● and can alternatively, for analysis, be represented as four C^1 -connected triangular pieces of degree 4 with degree 3 outer boundaries identical to the bicubic patch boundaries.

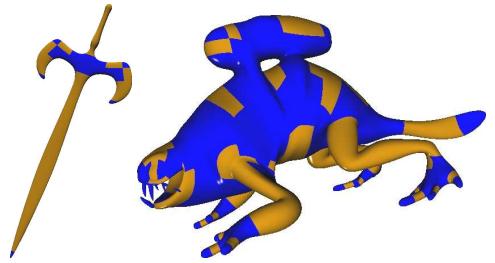


Figure 2: GPU smoothed quad surfaces: orange patches correspond to ordinary quads, blue patches to extraordinary quads.

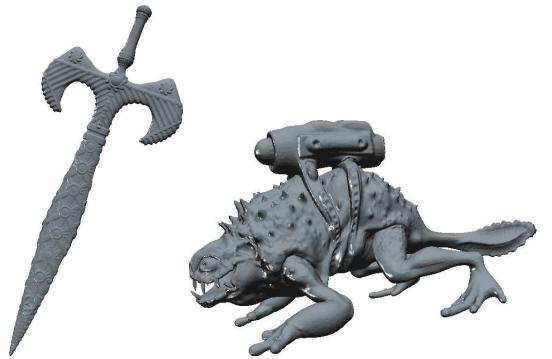


Figure 3: GPU smoothed quad surfaces with displacement mapping.

[Bunnell 2005; Shiue et al. 2005; Bolz and Schröder 2002] require separated quad meshes, i.e. quad meshes such that each quad has at most one point with valence $n \neq 4$. To turn quad meshes into separated quad meshes usually means applying at least one Catmull-Clark subdivision step on the CPU and four-fold data transfer to the GPU. In more detail, [Shiue et al. 2005] implemented recursive Catmull-Clark subdivision using several passes via the pixel shader, using textures for storage and spiral-enumerated mesh fragments. [Bolz and Schröder 2002] tabulate the subdivision functions up to a given density and linearly combine them in the GPU. [Bunnell 2005] provides code for adaptive refinement. Even though this code was optimized for the previous generation GPUs that provided connectivity by textures read in the pixel shader, this implementation adaptively renders the Frog (Figure 2) in real-time. (See Section 5 for a comparison). The main difference between our and Bunnell’s implementation is that we decouple mesh conversion from surface evaluation and therefore do not have the primitive explosion before the second rendering pass. Moreover, we place conversion early in the pipeline so that the pixel shader is freed for additional tasks.

Two alternative smoothing strategies mimic Catmull-Clark subdivision by generating a finite number of bicubic patches. [Peters 2000] generates NURBS output, that could be rendered, for example by the GPU algorithm of [Guthe et al. 2005]. But this has not been implemented to our knowledge. The method of [Loop and Schaefer 2007] generates one bicubic patch per quad following the shape of Catmull-Clark surfaces. Since these bicubic patches typically do not join smoothly, they compute two additional patches whose cross product approximates the normal of the bicubic patch. As pointed out in [Vlachos et al. 2001], this trompe l’oeil represents a simple solution when true smoothness is not needed. Comparing the number of operations in construction and evaluation, the method of [Loop and Schaefer 2007] should run at comparable speeds to our GPU quad mesh smoothing (see also Section 6).

2 The Conversion Algorithm

Here we give the algorithm. Analysis and implementation follow in the next sections. Essentially, the algorithm consists of computing new points near a vertex using Table 1 and, for each extraordinary quad, additional points according to Table 2. In Section 3, we will verify that these new points define a smooth surface and in Section 4, we show how the two stages are mapped to the vertex shader and geometry shader, respectively.

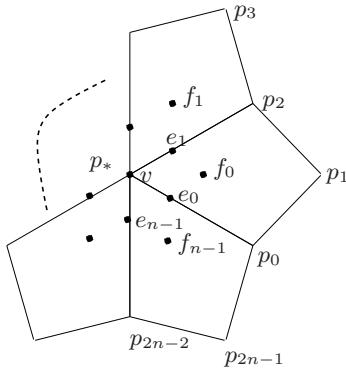


Figure 4: Smoothing the vertex neighborhood according to Table 1. The center point p_* , its direct neighbors p_{2j} and diagonal neighbors p_{2j+1} form a vertex neighborhood.

In the first part, we focus on a vertex neighborhood. A *vertex neighborhood* consists of a mesh point p_* and mesh points p_k ,

$f_j := (4p_* + 2p_{2j} + 2p_{2j+2} + p_{2j+1})/9$
$e_j := (f_j + f_{j-1})/2$
$v := \frac{1}{n} \sum_{j=0}^{n-1} e_j$
$t_j := v + \frac{1}{n\sigma_n} \sum_{\ell=0}^{n-1} \cos \frac{2\pi(j-\ell)}{n} e_\ell, j = 0, 1.$

Table 1: Computing control points v , e , f and t , the projection of e , at a vertex of valence n from the mesh points p_j of a vertex neighborhood; the subscripts are modulo $2n$. By default, $\sigma_n := (c_n + 5 + \sqrt{(c_n + 9)(c_n + 1)}) / 16$, the subdominant eigenvalue of Catmull-Clark subdivision.

$k = 0, \dots, 2n - 1$ of all quads surrounding p_* (Figure 4). A vertex v computed according to Table 1 is the limit point of Catmull-Clark subdivision as explained, for example, in [Halstead et al. 1993]. For $n = 4$, this choice is the limit of bicubic subdivision, i.e. B-spline evaluation. The rules for e_j and f_j are the standard rules for converting a uniform bicubic tensor-product B-spline to its Bézier representation of degree 3 by 3 [Farin 1990]. The points t_j are a projection of e_j into a common tangent plane (see e.g. [Gonzalez and Peters 1999]). The default scale factor σ is the subdominant eigenvalue of Catmull-Clark subdivision. We note that for $n = 4$, $e_{j+2} = 2v - e_j$ and $\sigma = 1/2$ so that the projection leaves the tangent control points invariant as $t_j = e_j$:

$$\text{for } n = 4, \quad t_j = v + \frac{2}{4}(e_j - e_{j+2}) = v + (e_j - v) = e_j. \quad (1)$$

In the second stage, we focus on the quads. Combining information from four vertex neighborhoods as shown in Figure 5, we can populate a tensor-product patch g of degree 3 by 3 in Bézier form [Farin 1990]:

$$g(u, v) := \sum_{k=0}^3 \sum_{\ell=0}^3 g_{k\ell} \binom{3}{k} u^k (1-u)^{3-k} \binom{3}{\ell} v^\ell (1-v)^{3-\ell}.$$

The patch is defined by its 16 control points $g_{k\ell}$. If the quad is ordinary, the formulas of Table 1 make this patch the Bézier representation of a bicubic spline in B-spline form. For example, in the notation of Figure 7, $(g_{k0})_{k=0..3} = (v^0, t_0^0, t_1^1, v^1)$. If the quad is

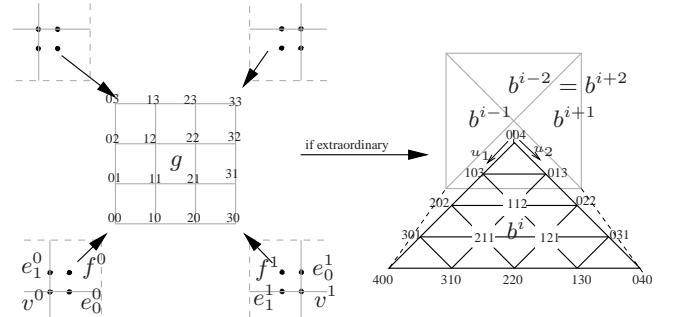


Figure 5: Patch construction. On the left, four vertex neighborhoods with vertices v^i each contribute one sector to assemble the 4×4 coefficients of the Bézier patch g , for example $g_{00} = v^0$, $g_{10} = e_0^0$, $g_{11} = f^0$, $g_{30} = v^1$, $g_{31} = e_0^1$ (we use superscripts to indicate vertices; see also Figure 7). On the right, the same four sectors are used to determine a c-patch if the underlying quad is extraordinary. The indices of the control points of g and b^i are shown. Note that only a subset of the coefficients of the four triangular pieces b^i is actually computed to define the c-patch. The full set of coefficients displayed here is only used to analyze the construction.

extraordinary, we use the bicubic patch to outline the shape as we

$b_{211}^i := b_{310}^i + \frac{1+c^i}{4}(t_1^{i+1} - t_0^i) + \frac{1-c^i}{8}(t_0^i - v^i)$
$+ \frac{3}{4(s^i+s^{i+1})}(f^i - e_0^i)$
$b_{121}^i := b_{130}^i + \frac{1+c^i}{4}(t_0^i - t_1^{i+1}) + \frac{1-c^i}{8}(t_1^{i+1} - v^{i+1})$
$+ \frac{3}{4(s^i+s^{i+1})}(f^{i+1} - e_1^{i+1})$
$b_{112}^i := g_* + 3(b_{211}^i + b_{121}^i - b_{121}^{i+1} - b_{211}^{i-1})/16$
$+ (b_{211}^{i+1} + b_{121}^{i-1} - b_{121}^{i+2} - b_{211}^{i-2})/16$

Table 2: Formulas for the 4×3 interior control points that, together with the vertex control points v^i and the tangent control points t_j^i , define a *c-patch*. See also Figures 7 and 8. Here $c^i := \cos \frac{2\pi}{n_i}$, $s^i := \sin \frac{2\pi}{n_i}$ and superscripts are modulo 4. By default, $g_* := (\sum_{i=0}^3 v^i + 3(e_0^i + e_1^i) + 9f^i)/64$, the central point of the ordinary patch.

replace it by a c-patch (Figure 1, *c*). A c-patch has the right degrees of freedom to cheaply and locally construct a smooth surface. We introduce the c-patch in terms of a well-known Bézier form of a polynomial piece b^i of total degree 4 [Farin 1990]:

$$b^i(u_1, u_2) := \sum_{k+\ell+m=4 \atop k, \ell, m \geq 0} b_{k\ell m}^i \frac{4!}{k!\ell!m!} u_1^k u_2^\ell (1-u_1-u_2)^m. \quad (2)$$

The c-patch is equivalent to the union of four b^i , $i = 0, 1, 2, 3$ of total degree 4, but defined by only 4×6 c-coefficients constructed in Tables 1 and 2:

$$v^i, t_0^i, t_1^i, b_{211}^i, b_{121}^i, b_{112}^i, i = 0, 1, 2, 3.$$

These c-coefficients imply the missing interior control points by C^1 continuity between the triangular pieces: for $j = 0, 1, 2, 3$ and $i = 0, 1, 2, 3$,

$$b_{3-j,0,1+j}^i = b_{0,3-j,1+j}^{i-1} := (b_{3-j,1,j}^i + b_{1,3-j,j}^{i-1})/2; \quad (3)$$

and the boundary control points $b_{k\ell 0}^i$ are implied by degree-raising [Farin 1990]:

$$\begin{aligned} b_{400}^i &:= v^i, & b_{310}^i &:= (v^i + 3t_0^i)/4, & b_{220}^i &:= (t_0^i + t_1^{i+1})/2, \\ b_{130}^i &:= (v^{i+1} + 3t_1^{i+1})/4, & b_{040}^i &:= v^{i+1}. \end{aligned} \quad (4)$$

In particular, a tensor-product patch g and a c-patch have *identical boundary curves of degree 3* where they meet. Basis functions corresponding to the 24 c-coefficients of the c-patch can be read off by setting one c-coefficient to one and all others to zero and then applying (3) and (4).

To derive the formulas for b_{211}^i and its symmetric counterpart b_{121}^i note that the formulas must guarantee a smooth transition between b^i and its neighbor patch on an adjacent quad, regardless whether the adjacent quad is ordinary or extraordinary. That is, the formulas are derived to satisfy *simultaneously* two types of smoothness constraints (see Section 3). By contrast, b_{112}^i is not pinned down by continuity constraints. We could

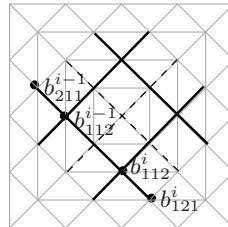


Figure 6: Dark lines cover the control points involved in the C^2 constraints (5). The points on dashed lines are implied by averaging.

choose each b_{112}^i arbitrarily without changing the formal smoothness of the resulting surface. However, we opt for increased smoothness at the center of the c-patch and additionally use the freedom to closely mimic the shape of Catmull-Clark subdivision surfaces, as we did earlier for vertices. First, we approximately satisfy four C^2 constraints across the diagonal boundaries at the central point b_{004} by enforcing

$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_{112}^0 \\ b_{112}^1 \\ b_{112}^2 \\ b_{112}^3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} b_{211}^0 - b_{121}^1 - q \\ b_{211}^1 - b_{121}^2 - q \\ b_{211}^2 - b_{121}^3 - q \\ b_{211}^3 - b_{121}^0 - q \end{bmatrix}, \quad (5)$$

where $q := \frac{1}{4} \sum_{i=0}^3 (b_{211}^i - b_{121}^i)$. The perturbation by q is necessary, since the coefficient matrix of the C^2 constraints is rank deficient. After perturbation, the system can be solved with the last equation implied by the first three. We add the constraint that the average of b_{112}^i matches $g_* := g(\frac{1}{2}, \frac{1}{2})$, the center position of the bicubic patch. Now, we can solve for the b_{112}^i , $i = 0, 1, 2, 3$ and obtain the formula of Table 2.

3 Smoothness Verification

In this section we formally verify the following lemma. For the purpose of the proof, we view the c-patch in its equivalent representation as four Bézier patches of total degree 4.

Lemma 1 Two adjacent polynomial pieces a and b defined by the rules of Section 2 (Table 1, Table 2, (3), (4)) meet at least

- (i) C^2 if a and b correspond to two ordinary quads.
- (ii) C^1 if a and b are adjacent pieces of a c-patch;
- (iii) C^1 if a and b correspond to two qads, exactly one of which is ordinary;
- (iv) with tangent continuity if a and b correspond to two different extraordinary quads;

Proof (i) If a and b are bicubic patches corresponding to ordinary quads, they are part of a bicubic spline with uniform knots and therefore meet C^2 . (ii) If a and b are adjacent pieces of a c-patch then Equations (3) enforce C^1 continuity.

For the remaining cases, let b be a triangular piece. Let u the parameter corresponding to the quad edge between $b_{400} = v^0$, where $u = 0$ and the valence is n_0 and $b_{040} = v^1$ where $u = 1$ and the valence is n_1 (see Figures 7 for (iii) and 8 for case (iv)). By construction, the common boundary $b(u, 0) = a(0, u)$ is a curve of degree 3 with Bézier control points (v^0, t_0^0, t_1^1, v^1) so that bicubic patches on ordinary quads and triangular patches on extraordinary quads match up exactly.

Denote by $\partial_1 b$ the partial derivative of b along the common boundary and by $\partial_2 b$ the partial derivative in the other variable. Since $b(u, 0) = a(0, u)$, we have $\partial_1 b(u, 0) = \partial_2 a(0, u)$. The partial derivative in the other variable of a is $\partial_2 a$. We will verify that the following conditions hold, that imply tangent continuity:

- if one quad is ordinary (case (iii)),
$$\partial_1 b(u, 0) = 2\partial_2 b(u, 0) + \partial_1 a(0, u); \quad (6)$$
- if both quads are extraordinary (case (iv)),
$$((1-u)\lambda_0 + u\lambda_1)\partial_1 b(u, 0) = \partial_2 b(u, 0) + \partial_1 a(0, u), \quad (7)$$

where $\lambda_0 := 1 + c^0$, $\lambda_1 := 1 - c^1$, and $c^i := \cos(\frac{2\pi}{n_i})$.

Both equations, (6) and (7), equate vector-valued polynomials of degree 3 (we write $\partial_1 b(u, 0)$ in degree-raised form [Farin 1990]). The equations hold, if and only if all Bézier coefficients are equal. Off hand, this means checking four vector-valued equations for each of (6) and (7). However, in both cases, the setup is symmetric with respect to reversal of the direction in which the boundary $b(u, 0)$ is traversed. That means, we need only check the first two equations (6') and (6'') of (6) and the first two equations (7') and (7'') of (7). We verify these equations by inserting the formulas of Tables 1 and 2.

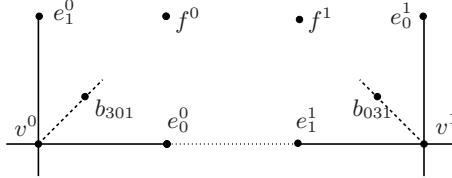


Figure 7: C^1 transition between a triangular and a bicubic patch.

To verify (6), the key observation is that $n_0 = n_1 = 4$ if one quad is ordinary. Hence $c^0 = c^1 = 0$ and $s^0 = s^1 = 1$ (cf. Table 2) and $t_j^i = e_j^i$. Therefore, for example (cf. Figure 7)

$$\begin{aligned} 2\partial_2 b(0, 0) &= 2 \cdot 4(b_{301} - v^0) = 8 \frac{3}{4} \left(\frac{e_0^0 + e_1^0}{2} - v^0 \right) \\ &= 3(e_0^0 + e_1^0) - 6v^0, \end{aligned}$$

where the factor $\frac{3}{4}$ stems from raising the degree from 3 to 4; and the second Bézier coefficient of $\partial_1 b(u, 0)$ (in degree-raised form) and of $2\partial_2 b(u, 0)$ are respectively (cf. Figure 7)

$$\begin{aligned} 3 \frac{(e_0^0 - v^0) + 2(e_1^1 - e_0^0)}{3} \quad \text{and} \\ 2 \cdot 4(b_{211} - b_{310}) &= 8 \left(\frac{e_1^1 - e_0^0}{4} + \frac{e_0^0 - v^0}{8} + 3 \frac{f^0 - e_0^0}{8} \right). \end{aligned}$$

Then, comparing the first two Bézier coefficients of $\partial_1 b(u, 0)$ and $2\partial_2 b(u, 0) + \partial_1 a(0, u)$ yields equality and establishes C^1 continuity:

$$\begin{aligned} 3 \underbrace{(e_0^0 - v^0)}_{\partial_1 b(0, 0)} &= 3 \underbrace{(e_0^0 + e_1^0) - 6v^0}_{2\partial_2 b(0, 0)} - 3 \underbrace{(e_1^0 - v^0)}_{\partial_1 a(0, 0)} \quad (6') \\ (e_0^0 - v^0) + 2(e_1^1 - e_0^0) &= 2(e_1^1 - e_0^0) + (e_0^0 - v^0) + 3(f^0 - e_0^0) \\ &\quad - 3(f^0 - e_0^0). \quad (6'') \end{aligned}$$

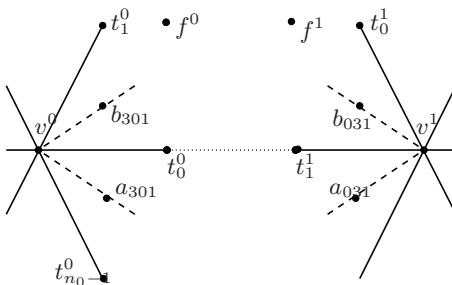


Figure 8: G^1 transition between two triangular patches.

The equations for (7) are similar, except that we need to replace e_j by t_j and keep in mind that, by definition,

$$(t_{n_0-1}^0 - v^0) + (t_1^0 - v^0) = 2c^0(t_0^0 - v^0).$$

Hence, for example,

$$\begin{aligned} \partial_2 b(0, 0) + \partial_1 a(0, 0) &= 4(b_{301} - v^0 + a_{301} - v^0) \\ &= \frac{3}{4} \cdot 2c^0(t_0^0 - v^0). \end{aligned}$$

The first of the four coefficient equations of (7) then simplifies to

$$\begin{aligned} 3(1 + c^0)(t_0^0 - v^0) &= 4(b_{301} + a_{301} - 2v^0) \\ &= 3\left(\frac{t_1^0 + t_0^0}{2} - v^0 + \frac{t_1^{n_0-1} + t_0^0}{2} - v^0\right) \\ &= 3\frac{1}{2}(2c^0(t_0^0 - v^0) + 2(t_0^0 - v^0)). \quad (7') \end{aligned}$$

Noting that terms $(f_0 - e_0^0)/(8(s^0 + s^1))$ in the expansions of b_{211} and a_{211} cancel, the second coefficient equation is

$$\begin{aligned} 6\lambda_0(t_1^1 - t_0^0) + 3\lambda_1(t_0^0 - v^0) &= 12(b_{211} + a_{211} - 2b_{310}) \\ &= \frac{12 \cdot 2(1 + c^0)}{4}(t_1^1 - t_0^0) + \frac{12 \cdot 2(1 - c^1)}{8}(t_0^0 - v^0). \quad (7'') \end{aligned}$$

It is easy to read off that the equalities hold. So the claim of smoothness is verified. |||

4 GPU Implementation

We implemented our scheme in DirectX 10 using the vertex shader to compute vertex neighborhoods according to Table 1 and the geometry shader primitive *triangle with adjacency* to accumulate the coefficients of the bicubic patch or compute a c-patch according to Table 2. We implemented conversion plus rendering in two variants: a 1-pass and a 2-pass scheme.

The *2-pass implementation* constructs the patches in the first pass using the vertex shader and the geometry shader and evaluates positions and normals in the second pass. Pass 1 streams out only the 4×6 coefficients of a c-patch and not the $4 \times \binom{4+2}{2}$ Bézier control points of the equivalent triangular pieces. The data amplification necessary to evaluate takes place by instancing a (u, v) -grid on the vertex shader in the *second pass*. That is, we *do not stream back large data sets after amplification*. Position and normal are computed on the (u, v) domain $[0..1]^2$ of the bicubic or of the c-patch (not on any triangular domains). In our implementation, the number of ALU ops for this evaluation is 59 both for the bicubic patch and for the c-patch. Table 3 lists the input, output and the computations of each pipeline stage. Figure 9 illustrates this association of computations and resources. Overall, the 2-pass implementation has small stream-out, short geometry shader code and minimal amplification on the geometry shader.

In the *1-pass implementation*, the evaluation immediately follows conversion in the geometry shader, using the geometry shader's ability to *amplify*, i.e. output multiple point primitives for each facet (Figure 10). While a 1-pass implementation sounds more efficient than a 2-pass implementation, DX10 limits data amplification in the geometry shader so that the maximal evaluation density is 8×8 per quad. Moreover, maximal amplification in the geometry shader slows the performance. We observed a minimum of 25% better performance of the 2-pass implementation.

5 Results

We compiled and executed the implementation on the latest graphics cards of both major vendors under DirectX10 and tested the performance for several industry-sized models. Two surface models and models with displacement mapping are shown in Figure 2

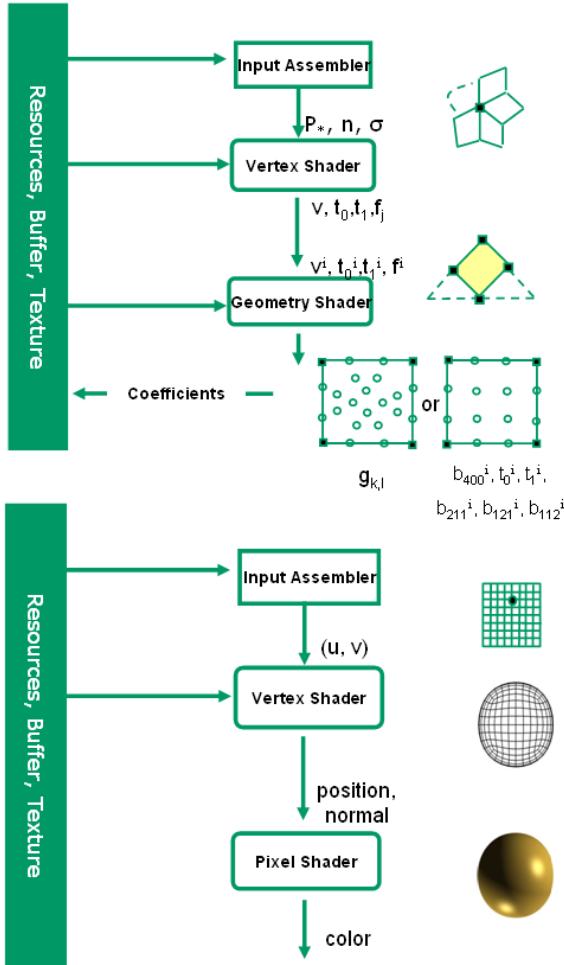


Figure 9: 2-pass implementation detailed in Table 3. The first pass converts, the second renders. Note that the geometry shader only computes at most 24 coefficients per patch and does not create (amplify to) evaluation point primitives.

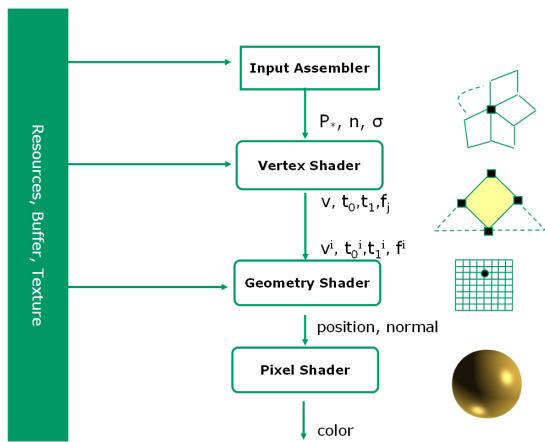


Figure 10: At present, the 1-pass conversion-and-rendering must place patch assembly and evaluation on the geometry shader. This is not efficient.

Pass 1		Conversion
VS In	P _* , n, σ	
VS	Use texture lookup to retrieve p_{2j}, p_{2j+1}	
VS Out	Compute v, e_j, f_j, t_0, t_1 (Table 1)	
	$v, t_0, t_1, f_j, j = 0..n - 1$	
GS In	$v^i, t_0^i, t_1^i, f^i, i = 0..3$	
GS	if ordinary quad assemble $g_{kl}, k, l = 0..3$ (Figure 5)	
GS Out	else compute $b_{211}^i, b_{121}^i, b_{112}^i$ (Table 2)	
	if ordinary quad, stream out $g_{kl}, k, l = 0..3$.	
	else stream out $b_{400}^i, t_0^i, t_1^i, b_{211}^i, b_{121}^i, b_{112}^i, i = 0..3$.	
Pass 2		Evaluating Position and Normal
VS In	(u, v)	
VS	if ordinary quad compute normal and position at (u, v) by the tensored de Casteljau's algorithm	
VS Out	else Compute the remaining Bézier control points (3) Compute normal and position at (u, v) by de Casteljau's algorithm adjusted to c-patches. position, normal	
PS In	position, normal	
PS	compute color	
PS Out	color	

Table 3: 2-Pass conversion: VS=vertex shader, GS=geometry shader, PS=pixel shader. VS Out of Pass 1 outputs n points f_j for one vertex (hence the subscript) and GS In of Pass 1 retrieves four points f^i , each generated by a different vertex of the quad (hence the superscript).

and 3 respectively. Table 4 summarizes the performance of the 2-pass algorithm for different granularities of evaluation. The frog model, in particular, provides a challenge due to the large number of extraordinary patches. The Frog Party shown in Figure 14 cur-

Mesh (verts,quads, eqs)	Frames per second			
	N = 5	9	17	33
Sword (140,138, 38%)	965	965	965	703
Head (602,600, 100%)	637	557	376	165
Frog (1308,1292, 59%)	483	392	226	87

Table 4: Frames per second for some standard test meshes with each patch evaluated on a grid of size $N \times N$; eqs = percentage of extraordinary quads. Sword and Frog are shown in Figure 2, Head in Figure 11.

rently renders at 50 fps for uniform evaluation for $N=9$, i.e. on a 9×9 grid. That is, the implementation converts $1292 * 9$ quads, of which 59% are extraordinary, and renders 1 million polygons 50 times per second. On the same hardware, we measured Bunnell's efficient implementation (distribution accompanying [Bunnell 2005]) featuring the single frog model, i.e. 1/9th of the work of the Frog Party, running at 44 fps with three subdivisions (equivalent to tessellation factor $N=9$). That is, GPU smoothing of quad meshes is an order of magnitude faster. Compared to [Shiue et al. 2005], the speed up is even more dramatic. While the comparison is not among equals since both [Shiue et al. 2005] and [Bunnell 2005] implement recursive Catmull-Clark subdivision, it is nevertheless fair to observe that the speedup is at least partially due to our avoiding stream back after amplification (data explosion due to refinement).

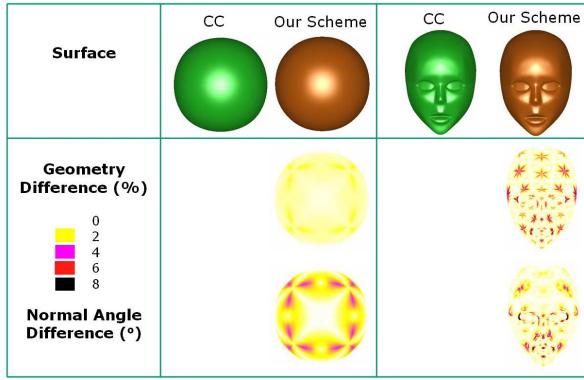


Figure 11: Comparison between the Catmull-Clark (CC) subdivision limit surface and the smoothed quad mesh surface for the same input.

We expect that more careful storage of vertex neighborhoods, in retrieving order, will further improve our use of texture cache and thereby improve the frames per second (fps) count.

Figure 11 compares the smoothed quad mesh surfaces with densely refined Catmull-Clark subdivision surfaces based on the same mesh. Both geometric distance, as percent of the local quad size, and normal distance, in degrees of variation, are compared. Especially after displacement, large models rendered by subdivision and quad smoothing appear visually indistinguishable. The relatively small examples, without displacement, shown in Figure 11 are also important to support our observation that c-patches do not create shape problems compared to a single bicubic patch: despite the lower degree and internal C^1 join, their visual appearance is remarkably similar to that of bicubic patches.

The **accompanying video** (see screen shots in Figures 12, 13, 14) illustrates real time displacement and animation. It was captured with a camcorder to show real time performance. The fps rates shown are lower than the ones in Table 4 since we captured it before we separated ordinary and extraordinary quad conversion in the implementation.

6 Discussion

Smoothing quad meshes on the GPU offers an alternative to highly refined facet representations transmitted to the GPU and is preferable for interactive graphics and integration with complex morphing and displacement. The separation into vertex and patch construction means that the number of scaled vertex additions (adds) per patch is independent of the valence. The cost of computing the control points *per patch*, i.e. with the cost of vertex computations distributed, is $4 \times (4 + 1 + 1 + 2) = 32$ adds per bicubic construction and computing t_j from t_0 and t_1 and determining b_{211}^i , b_{121}^i and b_{112}^i according to Table 2 amounts to an additional $4 \times (2 + 6 + 6 + 12) = 104$ adds per c-patch. The data transfer between passes in the 2-pass conversion is low since only 4×6 control points are intermittently generated. This compares favorably to, say [Loop and Schaefer 2007] where 16+12+12 coefficients are generated.

Since we only compute and evaluate in terms of the 24 c-patch coefficients, the computation of the cubic boundaries shared by a bicubic and a c-patch is mathematically identical. An explicit ‘if’-statement in the evaluation guarantees the exact same ordering of computations since boundary coefficients are only computed once, in the vertex shader, according to Table 1. That is, there is no pixel

drop out or gaps in the rendered surface. The resulting surface is watertight.

We advertised a 2-pass scheme, since, as we argued, the DX10 geometry shader is not well suited for the data amplification for evaluation after conversion. The 1-pass scheme outlined in Section 4 may become more valuable with availability of a dedicated hardware tessellator [Lee 2006]. Such a hardware amplification will also benefit the 2-pass approach in that the (u, v) domain tessellation, fed into the second pass will be replaced by the amplification unit.

Acknowledgment:

References

- BOLZ, J., AND SCHRÖDER, P. 2002. Rapid evaluation of Catmull-Clark subdivision surfaces. In *Web3D '02: Proceeding of the seventh international conference on 3D Web technology*, ACM Press, New York, NY, USA, 11–17.
- BUNNELL, M. 2005. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley, Reading, MA, ch. 7. Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping.
- CATMULL, E., AND CLARK, J. 1978. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design* 10, 350–355.
- FARIN, G. 1990. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press.
- GONZALEZ, C., AND PETERS, J. 1999. Localized hierarchy surface splines. In *ACM Symposium on Interactive 3D Graphics*, S. S. J. Rossignac, Ed., 7–15.
- GUTHÉ, M., BALÁZS, A., AND KLEIN, R. 2005. GPU-based trimming and tessellation of NURBS and T-spline surfaces. *ACM Trans. Graph.* 24, 3, 1016–1023.
- HALSTEAD, M., KASS, M., AND DEROSE, T. 1993. Efficient, fair interpolation using Catmull-Clark surfaces. *Proceedings of SIGGRAPH 93* (Aug), 35–44.
- LEE, A., MORETON, H., AND HOPPE, H. 2000. Displaced subdivision surfaces. In *Siggraph 2000, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, K. Akeley, Ed., Annual Conference Series, 85–94.
- LEE, M., 2006. Next generation graphics programming on Xbox 360. http://download.microsoft.com/download/d/3/0/d30d58cd-87a2-41d5-bb53-baf560aa2373/next_generation_graphics_programming_on_xbox_360.ppt.
- LOOP, C., AND SCHAEFER, S. 2007. Approximating Catmull-Clark subdivision surfaces with bicubic patches. Tech. rep., Microsoft Research, MSR-TR-2007-44.
- PETERS, J. 2000. Patching Catmull-Clark meshes. In *Siggraph 2000, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, K. Akeley, Ed., Annual Conference Series, 255–258.
- SHIUE, L.-J., JONES, I., AND PETERS, J. 2005. A realtime GPU subdivision kernel. *ACM Trans. Graph.* 24, 3, 1010–1015.
- STAM, J. 1998. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In *SIGGRAPH*, 395–404.
- VLACHOS, A., PETERS, J., BOYD, C., AND MITCHELL, J. L. 2001. Curved PN triangles. In *2001, Symposium on Interactive*

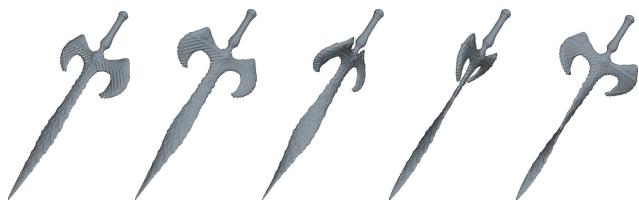


Figure 12: Real time displacement on the twisting Sword model. See the video.



Figure 13: Real time displacement on the twisting Frog model. See the video.



Figure 14: Asynchronous animation of nine Frogs. See the video.