Eurographics Symposium on Geometry Processing 2008
Pierre Alliez and Szymon Rusinkiewicz
(Guest Editors)

Volume 27 (2008), Number 5

Fast Parallel Construction of Smooth Surfaces from Meshes with Tri/Quad/Pent Facets

A. Myles and T. Ni and J. Peters

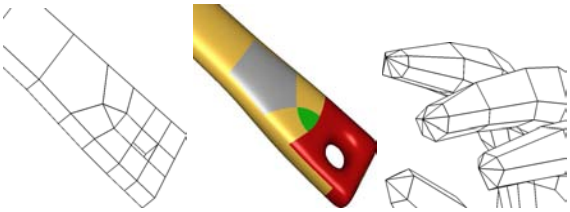
University of Florida

Abstract

Polyhedral meshes consisting of triangles, quads, and pentagons and polar configurations cover all major sampling and modeling scenarios. We give an algorithm for efficient local, parallel conversion of such meshes to an everywhere smooth surface consisting of low-degree polynomial pieces. Quadrilateral facets with 4-valent vertices are 'regular' and are mapped to bi-cubic patches so that adjacent bi-cubics join C^2 as for cubic tensor-product splines. The algorithm can be implemented in the vertex and geometry shaders of the GPU pipeline and does not use the fragment shader. Its implementation in DirectX 10 achieves conversion plus rendering at 659 frames per second with 42.5 million triangles per second on input of a model of 1300 facets of which 60% are not regular.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Motivation



(a) (b) (c)

Figure 1: ***Transition of feature lines.*** *(a,b) Axe handle detail (see Figure 12) using a triangle and a pentagon to transition between detailed and coarser areas. (c) Polar configurations naturally terminate parallel feature lines along elongations, like fingers.*

Compared to pure quad meshes, meshes allowing for triangles, quadrilaterals, and pentagons and polar configurations simplify remeshing of scanned data and enrich the design space for control meshes of smooth surfaces: while quads naturally model the flow of (parallel) feature lines and are therefore the main facet type in many models, triangular facets allow merging lines while pentagonal facets allow starting new lines without creating T-corners (as illustrated in Figure 1) or forcing refinement of intermediate models to satisfy connectivity or quad-layout constraints. *Polar configurations*, that is closed triangle fans whose outer vertices have valence 4 and central vertex has arbitrary (possibly large) valence, naturally model features such as finger tips and eyes (Figures 1, 2). This paper gives a local, parallel algorithm and describes its GPU implementation to automatically convert such a general polyhedral mesh into a piecewise polynomial surface

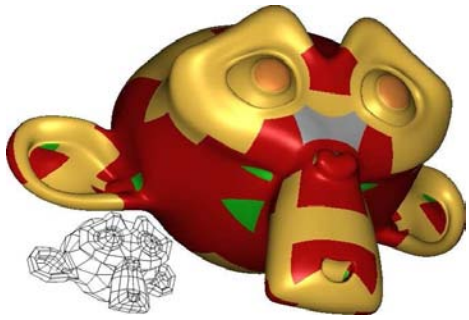


Figure 2: ***A quad/tri/pent model*** *converted to a smooth surface consisting of bi-cubic patches (*yellow*), polar patches (*orange*), and P_m-patches $m = 3$ (*green*), $m = 4$ (*red*), $m = 5$ (*gray*). Cf. Fig. 13.*

- that is C^1 everywhere and C^2 in regular regions that are converted to bi-cubic patches; and,
- consists of one patch for each triangle, quad, or pentagon in the mesh regardless of vertex valence; there is no need for preprocessing to isolate mesh points of different valence or to convert all facets into one facet type;
- is computed using only local mesh context so that the surfaces can be constructed and rendered in parallel on the GPU at high frame rates.

# coeffs	[LS08]	[NYM*08]	our method
triangle	$25 \times 3 = 75$	$24 \times 3 = 72$	13 or 19
pentagon	$25 \times 5 = 125$	$24 \times 5 = 120$	31

Table 1: ***Number of control points per n-gon.*** *Converting triangles and pentagons directly avoids a control-point amplifying subdivision step needed to apply [LS08, NYM*08].*

The advantage of not having to refine quad-dominant meshes [LKH08] to pure quad meshes is evident in Table 1; and simple T-corners in quad-dominant meshes can be replaced by 5-sided facets. We limit the facets to triangles, quads and pentagons due to current GPU constraints (see Section 8).

Section 3 introduces the patch representations, Section 4 the (continuity) constraints that imply the construction rules and Section 5 gives these rules explicitly, inside boxes, so that they are easy to implement. Section 6 explains an implementation on the GPU. The electronic Appendix explicitly verifies the correctness of the formulas and displays a gallery of configurations testing the surface quality.

2. Related Literature

Catmull-Clark subdivision [CC78] is an accepted standard for mesh smoothing. Techniques evaluating or approximating its limit surface such as [Sta98, BS02, Bun05, SJP05, Pet00] require separation of extraordinary vertices (those with valence $\neq 4$) to construct their data structures or reduce the number of cases, quadrupling the number of facets in the mesh. The constructions in [LS08, NYM*08] do not require such separation and have been implemented on the GPU at high frame rates: [LS08] yields bi-cubic C^0 surfaces with surrogate tangent patches for consistent lighting (see also [VPBM01]; and [BS07] for once CPU-refined Loop meshes); [NYM*08] generates C^1 surfaces that approximate Catmull-Clark surfaces well. For quad meshes, our construction reduces to [NYM*08].

[SL03, PS04, SW05] accept both triangles and quads and thereby avoid additional CPU subdivision into quad-only meshes; but we are unaware of efficient GPU implementations of these algorithms. The advantage of supporting polar configurations for high-valent vertices has been demonstrated in [MKP07]. We reinterpret this construction to fit GPU constraints.

3. Patch construction types

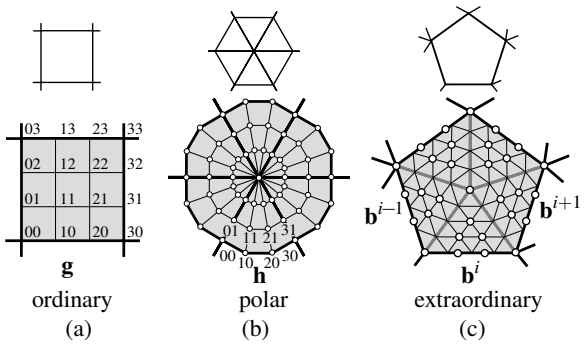


Figure 3: ***Mesh-to-patch conversion.*** *The input mesh (*top*) is converted to patches (*bottom*) as follows. (a) An ordinary facet is converted to a bi-cubic patch with 16 control points $\mathbf{g}_{ij}$. (b) Every triangle in polar configuration becomes a singular bi-cubic patch represented by 13 control points ∘. (c) An extraordinary facet with m sides is converted to a P_m-patch defined by $6m+1$ control points shown as ∘. The P_m-patch is equivalent to m C^1-connected degree-4 triangular patches $\mathbf{b}^i$, $i = 0 \ldots m-1$, having cubic outer boundaries.*

We have three classes of facets in our input mesh:

1. *Ordinary*: quads with all vertices having valence 4.
2. *Polar*: triangles in polar configurations.
3. *Extraordinary*: triangles, pentagons, and those quads that are not ordinary facets due to vertex valence.

Ordinary & Polar:
We convert ordinary quads and polar triangles to tensor-product degree bi-3 patches in Bernstein-Bézier form (BB-form),

$$\mathbf{g}(u,v) := \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{g}_{ij} \binom{3}{i} (1-u)^{3-i} u^i \binom{3}{j} (1-v)^{3-j} v^j,$$

defined by only 4×4 BB-coefficients $\mathbf{g}_{ij} \in \mathbb{R}^3$. The coefficients are indexed as in Figure 3(a). Since the edge 03—33 of the polar patch is collapsed to one vertex, the polar patch $\mathbf{h}$ is defined by 13 BB-coefficients $\mathbf{h}_{ij} \in \mathbb{R}^3$ (Figure 3(b)).

Extraordinary:
An extraordinary facet with m sides is converted into a P_m-patch. A P_m-patch is a piecewise degree 4 C^1 spline patch with m boundaries of degree 3. A P_m-patch is defined by $6m+1$ control points indicated as ∘ in Figures 3(c),4(a). That is, the P_m-patch corresponding to a triangular, quadrilateral or pentagonal facet is defined by a total of 19, 25 or 31 points, respectively.

For evaluation, we can write the i^{th} sector of a P_m-patch in triangular BB-form of total-degree 4 (Figure 3(b) and (c)),

$$\mathbf{b}(u,v) := \sum_{i+j+k=4} \mathbf{b}_{ijk} \frac{4!}{i!\,j!\,k!} u^i v^j (1-u-v)^k, \qquad (1)$$

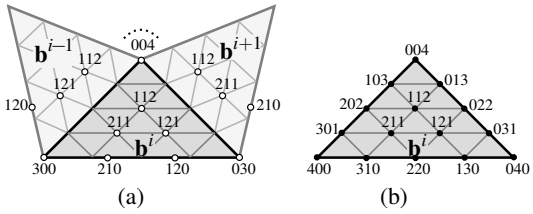


Figure 4: ***Sector of a P_m-patch.*** *The triangular* sectors *are listed in counter-clockwise order with a modulo-m superscript. (a) 14 control points from three consecutive sectors of a P_m-patch define (b) a single patch in triangular BB-form via relations* (2) *and* (3).

where the $\binom{4+2}{2}$ BB-coefficients $\mathbf{b}_{ijk} \in \mathbb{R}^3$ indexed as in Figure 4(b) are computed from the 14 coefficients labeled in 4(a) by simple averaging, namely

1. degree-raising the cubic boundary

$$[\mathbf{b}^i_{400}, \mathbf{b}^i_{310}, \mathbf{b}^i_{220}, \mathbf{b}^i_{130}, \mathbf{b}^i_{040}] = \quad (2)$$
$$[\mathbf{b}^i_{300}, \frac{\mathbf{b}^i_{300}+3\mathbf{b}^i_{210}}{4}, \frac{\mathbf{b}^i_{210}+\mathbf{b}^i_{120}}{2}, \frac{3\mathbf{b}^i_{120}+\mathbf{b}^i_{030}}{4}, \mathbf{b}^i_{030}]$$

2. and computing the shared BB-coefficients on the sector boundaries $\mathbf{b}^i_{3-l,0,1+l} = \mathbf{b}^{i-1}_{0,3-l,1+l}$ for $l = 0,1,2,3$ corresponding to indices 301, 202, 103 and 004 in Figure 4(b), from the C^1 constraints (cf. Section 4.1 below):

$$\mathsf{c}_m := \cos\frac{2\pi}{m}, \mu := 1-\mathsf{c}_m, \quad k_2 := \frac{1}{2\mu}, k_1 := 1-2k_2,$$
$$\mathbf{b}^i_{3-l,0,l+1} := k_1\mathbf{b}^i_{4-l,0,l} + k_2\left(\mathbf{b}^i_{3-l,1,l} + \mathbf{b}^{i-1}_{1,3-l,l}\right). \quad (3)$$

4. Smoothness constraints

In this section, we motivate the smoothness constraints that define P_m-patches. The challenge in choosing the smoothness constraints is threefold: (i) to guarantee solvability for the coefficients of the patches and their layout, (ii) to arrive at simple formulas for the construction and (iii) to obtain rules that obey the topology and reflect the geometry of the input mesh. Section 4.1 examines the intra-P_m-patch boundaries emanating from the center of the patch to each of its corners, and Section 4.2 addresses the boundary between adjacent patches.

4.1. Internal P_m-patch smoothness

Figure 5 illustrates the transition between two sectors of the P_m-patch. At the central point of the P_m-patch, the unbiased choice of tangents implies

$$2\mathsf{c}_m(\mathbf{b}^i_{103} - \mathbf{b}^i_{004}) = (\mathbf{b}^{i-1}_{103} - \mathbf{b}^i_{004}) + (\mathbf{b}^{i+1}_{103} - \mathbf{b}^i_{004}). \quad (4)$$

Here *unbiased* means that the construction does not depend on which corner we choose for the starting index. By elementary transformation, and noting that $\mathbf{b}^i_{013} = \mathbf{b}^{i+1}_{103}$, this

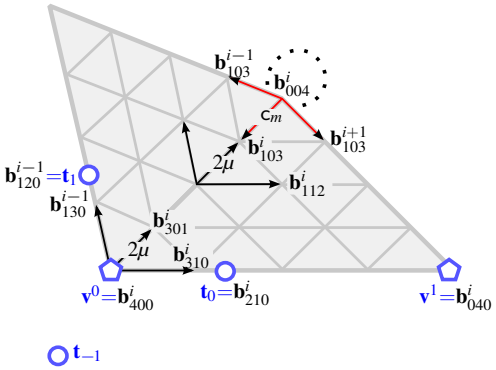


Figure 5: ***Internal P_m-patch constraints.*** *The isotropic C^1 constraint at the patch center $\mathbf{b}^i_{004}$ (red arrows, see Equation* (4)*) is propagated to the vertex $\mathbf{v}^0 = \mathbf{b}^i_{400}$. The tangent coefficients $\mathbf{t}_0$ and $\mathbf{t}_1$ correspond to consecutive external boundaries of the P_m-patch; $\mathbf{t}_{-1}$ belongs to the neighbor patch across the horizontal edge $\mathbf{v}^0$, $\mathbf{v}^1$.*

relation is equivalent to

$$\mathbf{b}_{004} = k_1\mathbf{b}^i_{103} + k_2\left(\mathbf{b}^i_{013} + \mathbf{b}^{i-1}_{103}\right), \quad (5)$$

i.e. relation (3) for $l = 3$. To obtain an internally C^1 P_m-patch, it is necessary and sufficient to enforce this relation along the internal sector boundaries:

$$2(1-\mathsf{c}_m)(\mathbf{b}^i_{3-l,0,l+1} - \mathbf{b}^i_{4-l,0,l}) = \quad (6)$$
$$(\mathbf{b}^i_{3-l,1,l} - \mathbf{b}^i_{4-l,0,l}) + (\mathbf{b}^{i-1}_{1,3-l,l} - \mathbf{b}^i_{4-l,0,l})$$

The relation holds due to the assignment (3) for $l = 0,1,2,3$.

4.2. Smoothness across patches

At $\mathbf{v}^0$, the internal C^1 constraints merge with the G^1 constraints across facet edges. We focus on the P_m-patch edge from $\mathbf{v}^0$ to $\mathbf{v}^1$. Let ∂_k be the partial derivative operator with respect to the k^{th} parameter (Figure 6, *left*). Then $\partial_1\mathbf{b}^i(0,0) = 3(\mathbf{t}_0 - \mathbf{v}^0) = 4(\mathbf{b}^i_{310} - \mathbf{v}^0)$ at $\mathbf{v}^0$ in the direction of $\mathbf{v}^1$; and similarly, $3(\mathbf{t}_1 - \mathbf{v}^0) = 4(\mathbf{b}^{i-1}_{130} - \mathbf{v}^0)$. For an unbiased choice of $\mathbf{t}_j$ and n^0 the valence at $\mathbf{v}^0$,

$$2\mathsf{c}_{n^0}(\mathbf{t}_0 - \mathbf{v}^0) = (\mathbf{t}_1 - \mathbf{v}^0) + (\mathbf{t}_{-1} - \mathbf{v}^0). \quad (7)$$

must hold. By relation (6) at $\mathbf{v}^0$, i.e. for $l = 0$,

$$\frac{4}{3k_2}(\mathbf{b}^i_{301} - \mathbf{v}^0) = (\mathbf{t}_0 - \mathbf{v}^0) + (\mathbf{t}_1 - \mathbf{v}^0). \quad (8)$$

Eliminating $(\mathbf{t}_1 - \mathbf{v}^0)$ between (8) and (7) yields

$$\mathbf{t}_{-1} - \mathbf{v}^0 = (1+2\mathsf{c}_n)(\mathbf{t}_0 - \mathbf{v}^0) - \frac{8\mu}{3}(\mathbf{b}^i_{301} - \mathbf{v}^0). \quad (9)$$

If, in particular $\mathbf{v}^0$ and $\mathbf{v}^1$ are the endpoints of the cubic boundary $\mathbf{b}^i(u,0) = \mathbf{g}(0,u)$ between a bi-cubic patch $\mathbf{g}$ and

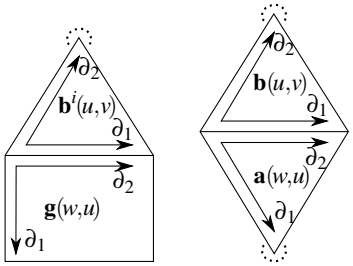


Figure 6: ***Derivatives along adjacent patches.*** *(*left*) The P_m-patch-ordinary and (*right*) P_m-patch-P_m-patch boundaries. ∂_k differentiates with respect to the k^{th} parameter.*

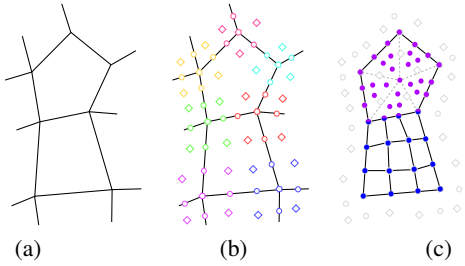


Figure 7: ***Algorithm Overview.*** *(a) Tri/quad/pent input mesh with vertices $\mathbf{p}_i$. (b) Per-vertex computation generating coefficients $\mathbf{v}^i$, $\mathbf{t}^i_j$ and $\mathbf{f}_j$. (c) Per-facet computation generating internal coefficients $\mathbf{b}^i_{211}$, $\mathbf{b}^i_{121}$, $\mathbf{b}^i_{112}$ and $\mathbf{b}^i_{004}$ (purple).*

a P_m-patch $\mathbf{b}^i$ then $n^0 = 4$ hence $\mathsf{c}_{n^0} = 0$; and at $(0,0)$ corresponding to $\mathbf{v}^0$,

$$\partial_1\mathbf{g}(0,0) = 3(\mathbf{t}_{-1} - \mathbf{v}^0), \quad \partial_2\mathbf{b}^i(0,0) = 4(\mathbf{b}^i_{301} - \mathbf{v}^0).$$

Then (9) simplifies to

$$\partial_1\mathbf{g}(0,0) = \partial_1\mathbf{b}^i(0,0) - 2\mu\partial_2\mathbf{b}^i(0,0) \tag{10}$$

This defines the first constraint of the G^1 conditions (11). The analogous constraint at $\mathbf{v}^1$ needs to be recast so that the direction of the derivative is consistent with the constraint at $\mathbf{v}^0$, resulting in the linearly blended left hand side $(1-2\mathsf{c}_m u)\,\partial_1\mathbf{b}^i(u,0)$ of the G^1 conditions

$$(1-2\mathsf{c}_m u)\,\partial_1\mathbf{b}^i(u,0) = 2\mu\partial_2\mathbf{b}^i(u,0) + \partial_1\mathbf{g}(0,u). \tag{11}$$

When a $P_{m_\mathbf{b}}$-patch sector $\mathbf{b}$ meets a $P_{m_\mathbf{a}}$-patch sector $\mathbf{a}$ (Figure 6, *right*), we need to replace $\mathbf{t}_{-1} - \mathbf{v}^0$ by the substitution analogous to (8). The result are the scalar μ, ν weights in the G^1 conditions

$$\ell(u)\partial_1\mathbf{b}(u,0) = \mu\partial_2\mathbf{b}(u,0) + \nu\partial_1\mathbf{a}(0,u), \tag{12}$$

where

$$\mu := 1 - \mathsf{c}_{m_\mathbf{b}},\ \nu := 1 - \mathsf{c}_{m_\mathbf{a}},\ \xi^i := 1 + \mathsf{c}_{n^i},$$
$$\ell(u) := \big((1-u)\ell_0 + u\ell_1\big),\ \ell_0 := \xi^0,\ \ell_1 := 2\mu - \xi^1.$$

5. Construction

The patches are constructed in two stages (that will correspond to shaders in the GPU implementation) and that are summarized in Figure 7. The algorithm can be implemented using only the boxed formulas in this section.

5.1. Per-vertex computation

The first stage computes local information around a vertex $\mathbf{p}_*$. For consistent labeling irrespective of the number n of surrounding facets or their number m of sides, we use indices $\mathbf{p}_j$ as labeled in Figure 8(a). That is, we have $3n$ indices and label redundantly in the case of a triangle or quad as shown in Figure 8(b). Labeling the one-ring uniformly results in

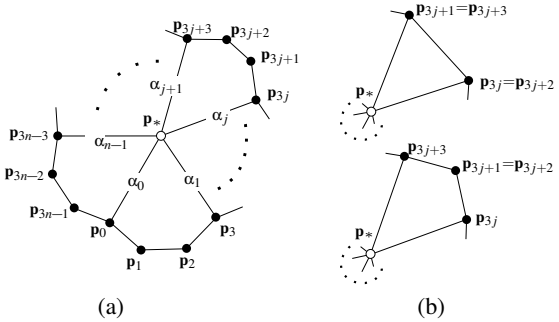


Figure 8: ***Per-vertex input.*** *(a) When every facet at the mesh point $\mathbf{p}_*$ is 5-sided, we need $3n$ indices. (b) Triangles and quads are redundantly indexed.*

simple generic formulas and does not result in noticeable overhead for the GPU implementation (see Section 6.1). For each edge emanating from the $\mathbf{p}_*$, there is one scalar to support creases.

At each vertex $\mathbf{p}_*$ of the input mesh, we compute (see Figure 9) a corresponding point $\mathbf{v}$ (by default the Catmull-Clark limit point) and face vertices $\mathbf{f}_j$ positioned according to the well-known B-spline-to-BB-form conversion formulas:

$$\begin{aligned}
\mathbf{p}_{3j*} &:= \left(\mathbf{p}_{3j+1} + \mathbf{p}_{3j+2}\right)/2 \\
\mathbf{v} &:= \frac{\sum_{j=0}^{n-1}\left(n\mathbf{p}_* + 4\mathbf{p}_{3j} + \mathbf{p}_{3j*}\right)}{n(n+5)} \\
\mathbf{f}_j &:= \frac{1}{9}\left(4\mathbf{p}_* + 2(\mathbf{p}_{3j} + \mathbf{p}_{3j+3}) + \mathbf{p}_{3j*}\right)
\end{aligned} \tag{13}$$

To adjust the sharpness of feature lines, similar to blend ratios [Pet95] and semi-smooth creases [DKT98], we can generalize (13) by giving each facet corner two scalars

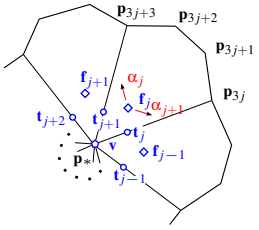
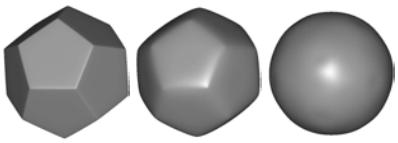


Figure 9: ***Per-vertex computation.*** *For every vertex in the mesh, the control points* $\mathbf{v}$ *and* $\mathbf{f}_j$ *are defined by* (13)*. The co-dependent co-planar* $\mathbf{t}_j$*s are not computed but represented by vectors* $\boldsymbol{\tau}_1$ *and* $\boldsymbol{\tau}_2$*. Indices run counter-clockwise and are interpreted modulo n.*

$\alpha_j, \alpha_{j+1} \in [0,1]$ (see Figure 9) and define

$$\begin{aligned}\mathbf{f}_j(\alpha_j,\alpha_{j+1}) &:= (1-\alpha_j)(1-\alpha_{j+1})\mathbf{p}_* \\ &+ (1-\alpha_j)\alpha_{j+1}\frac{\mathbf{p}_*+\mathbf{p}_{3j}}{2} + \alpha_j(1-\alpha_{j+1})\frac{\mathbf{p}_*+\mathbf{p}_{3j+3}}{2} \\ &+ \alpha_j\alpha_{j+1}\frac{\mathbf{p}_*+\mathbf{p}_{3j}+\mathbf{p}_{3j+3}+\mathbf{p}_{3j*}}{4}.\end{aligned} \tag{14}$$

Setting $\alpha_j = \alpha_{j+1} = 2/3$, we recover formula (13) for $\mathbf{f}_j$ which, in turn, for $m = 4$ reduces to the well-known formula for bi-cubic B-spline-to-BB conversion. Setting $\alpha_j = \alpha_{j+1} = 0$, we sharpen the transition across the facet's edges sharing $\mathbf{v}$. If all scalar weights of a model are zero, the control polyhedron is reproduced (see Figures 10 and 12).



$\alpha = 1/10$ $\alpha = 2/5$ $\alpha = 4/5$

Figure 10: ***Sharpness adjustment.*** *The dodecahedron with various blend ratios.* $\alpha = 0$ *results in a* C^0 *crease.*

We abbreviate the scalars

$$\mathsf{c}_{j,n} := \cos\frac{2\pi j}{n}, \quad \mathsf{s}_{j,n} := \sin\frac{2\pi j}{n}, \quad \mathsf{c}_n := \mathsf{c}_{1,n}, \quad \mathsf{s}_n := \mathsf{s}_{1,n},$$

$$\lambda_n := \frac{1}{16}\left(\mathsf{c}_n + 5 + \sqrt{(\mathsf{c}_n+9)(\mathsf{c}_n+1)}\right),$$

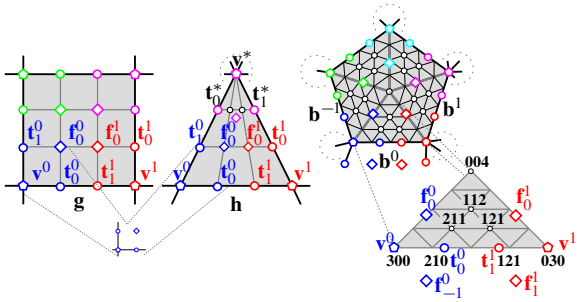
and define two vectors $\boldsymbol{\tau}_1$ and $\boldsymbol{\tau}_2$ that span the tangent plane at the limit point by averaging:

$$\begin{aligned}\mathbf{e}_j &:= (\mathbf{f}_j + \mathbf{f}_{j-1})/2, \\ \boldsymbol{\tau}_1 &:= \frac{1}{n\lambda_n}\sum_{j=0}^{n-1}\mathsf{c}_{j,n}\mathbf{e}_j, \qquad \boldsymbol{\tau}_2 := \frac{1}{n\lambda_n}\sum_{j=0}^{n-1}\mathsf{s}_{j,n}\mathbf{e}_j,\end{aligned} \tag{15}$$

For $n = 4$, it is easy to check that $\mathbf{e}_0 + \mathbf{e}_2 = 2\mathbf{v} = \mathbf{e}_1 + \mathbf{e}_3$ and these vectors simplify to $\boldsymbol{\tau}_1 := \frac{1}{4}(\mathbf{f}_3 + \mathbf{f}_0 - \mathbf{f}_1 - \mathbf{f}_2) = \mathbf{e}_0 - \mathbf{v}$ and $\boldsymbol{\tau}_2 := \mathbf{e}_1 - \mathbf{v}_0$ as in standard B-spline-to-BB conversion.

5.2. Per-patch computation

Let n^i denote the input mesh valence associated with vertex $\mathbf{v}^i$. To keep the notation simple, we assume without loss of generality that the edge under consideration has endpoints $\mathbf{v}^0$ and $\mathbf{v}^1$ and that the P_m-patch sector with this edge has index 0 as in Figure 11. Superscripts have precedence over subscripts, so that, for instance, $\mathbf{f}_1^1$ refers to the successor neighbor facet when counting counterclockwise around $\mathbf{v}^1$ as illustrated in Figure 11(c).



(a) (b) (c)

Figure 11: ***Patch assembly.*** *(a) For an ordinary facet, four vertex corners are combined to assemble the bi-cubic patch* $\mathbf{g}$*. (b) Polar triangles are converted similarly, except* $\mathbf{h}_{12}$ *and* $\mathbf{h}_{22}$*, marked* $\circ$*, are assigned to satisfy* C^1 *constraints at* $\mathbf{v}^*$*. (c) The triangle sectors of a* P_m*-patch use two vertex corners to determine the cubic boundary,* $\mathbf{b}^i_{211}$ *and* $\mathbf{b}^i_{121}$*.*

First, for each corner $i = 0 \ldots m{-}1$, we compute the tangent control points $\mathbf{t}_0^i$ and $\mathbf{t}_1^i$ using (15) in the local context of the corner (established by the index $j = j^k, k = 0 \ldots m{-}1$)

$$\mathbf{t}_j := \mathbf{v} + \boldsymbol{\tau}_1\mathsf{c}_{j,n} + \boldsymbol{\tau}_2\mathsf{s}_{j,n}. \tag{16}$$

Then the cubic boundary curve between $\mathbf{v}^0$ and $\mathbf{v}^1$ of any bi-cubic or P_m-patch is defined by

$$\mathbf{b}^0_{300} := \mathbf{v}^0,\ \mathbf{b}^0_{210} := \mathbf{t}^0_0,\ \mathbf{b}^0_{120} := \mathbf{t}^1_0,\ \mathbf{b}^0_{030} := \mathbf{v}^1. \tag{17}$$

If the facet is ordinary, the control points $\mathbf{v}^i$, $\mathbf{t}^i_0$, $\mathbf{t}^i_1$ and $\mathbf{f}^i_0$ for $i = 0, \ldots, 3$ completely determine the bi-cubic patch $\mathbf{g}$ as shown in Figure 11(a). We note that (see the remark following (15)) equations (13) and (16) reduce to the standard bi-cubic B-spline-to-BB conversion formulas in the ordinary case. The polar case proceeds identically, except that, with n^* the valence at $\mathbf{v}^*$, the coefficients shown as black $\circ$ in Figure 11(b) are chosen based on B-spline-to-Bézier curve

conversion formulas to ensure a C^1 fit with adjacent polar patches:

$$\begin{aligned}\mathbf{h}_{12} &:= \frac{1}{2+\mathsf{c}_{n^*}}\left(2\mathbf{h}_{02}+\mathbf{h}_{32}+(\mathsf{c}_{n^*}-1)\mathbf{v}^*\right)\\ \mathbf{h}_{22} &:= \frac{1}{2+\mathsf{c}_{n^*}}\left(2\mathbf{h}_{32}+\mathbf{h}_{02}+(\mathsf{c}_{n^*}-1)\mathbf{v}^*\right).\end{aligned} \tag{18}$$

Creases are supported by replacing $\mathbf{h}_{12}$ and $\mathbf{h}_{22}$ by

$$\hat{\mathbf{h}_{12}} := \frac{3}{2}\alpha_j\mathbf{h}_{12}+\left(1-\frac{3}{2}\alpha_j\right)\mathbf{h}_{02} \tag{19}$$

with scalars α_j and $\hat{\mathbf{h}_{22}}$, computed similarly.

If the facet is extraordinary (Figure 11(c)), we determine $\mathbf{b}^0_{211}, \mathbf{b}^0_{121}, \mathbf{b}^0_{004}$ (or simply, $\mathbf{b}_{004}$ since it is common to all the sectors) by the following uniform set of formulas for $m = 3,4,5$ that follow from the constraints of Section 4.2.

$$\begin{aligned}\mu &:= 1-\mathsf{c}_m, \quad \xi^i := 1+\mathsf{c}_{n^i},\\ \mathbf{b}^0_{211} :=& \mathbf{b}^0_{310}+\frac{\xi^0}{4\mu}(\mathbf{t}^1_1-\mathbf{t}^0_0)+\frac{2\mu-\xi^1}{8\mu}(\mathbf{t}^0_0-\mathbf{v}^0)\\ &+\frac{3}{8\mu(\mathsf{s}_{n^0}+\mathsf{s}_{n^1})}(\mathbf{f}^0_0-\mathbf{f}^0_{-1})\\ \mathbf{b}^0_{121} :=& \mathbf{b}^0_{130}+\frac{\xi^1}{4\mu}(\mathbf{t}^0_0-\mathbf{t}^1_1)+\frac{2\mu-\xi^0}{8\mu}(\mathbf{t}^1_1-\mathbf{v}^1)\\ &+\frac{3}{8\mu(\mathsf{s}_{n^0}+\mathsf{s}_{n^1})}(\mathbf{f}^1_0-\mathbf{f}^1_1)\\ w^{(3)} &:= 2, \quad w^{(4)} := 1, \quad w^{(5)} := -3,\\ \mathbf{b}_{004} :=& \frac{1}{m(15+w^{(m)})}\sum_{i=0}^{m-1}\left(w^{(m)}\mathbf{v}^i+3(\mathbf{t}^i_0+\mathbf{t}^i_1)+9\mathbf{f}^i\right)\end{aligned} \tag{20}$$

For $m = 4$, the central BB-coefficient $\mathbf{b}_{004}$ of the P_m-patch is chosen to be the midpoint of the bi-cubic patch defined by Figure 11(a). For $m \neq 4$, $\mathbf{b}_{004}$ is parameterized by $w^{(m)}$.

The final coefficients $\mathbf{b}^i_{112}$ are in principle free to choose. We obtained best results by approximating C^2 constraints at $\mathbf{b}_{004}$. For $m = 3$, this enforces the C^2 constraints exactly. Here $\mathbf{b}^2_{103}$ (for $m = 3$) and $\mathbf{b}^i_{202}$ (for $m = 5$) are defined by formula (3). The Appendix verifies that the boxed formulas enforce the continuity constraints.

For $m = 3$

$$\mathbf{b}^0_{112} := \mathbf{b}_{004}+\frac{1}{2}(\mathbf{b}_{004}-\mathbf{b}^2_{103})$$

For $m = 4$

$$\begin{aligned}\mathbf{b}^0_{112} :=& \mathbf{b}_{004}+3(\mathbf{b}^0_{211}+\mathbf{b}^0_{121}-\mathbf{b}^1_{121}-\mathbf{b}^{-1}_{211})/16\\ &+(\mathbf{b}^1_{211}+\mathbf{b}^{-1}_{121}-\mathbf{b}^2_{211}-\mathbf{b}^2_{121})/16\end{aligned} \tag{21}$$

For $m = 5$

$$\begin{aligned}\mathbf{b}^0_{112} :=& (1-\mathsf{c}_m)\Big(\mathbf{b}_{004}+\frac{1}{5}\Big(\mathbf{b}^3_{202}\\ &-4\mathsf{c}_{2,m}\left(\mathbf{b}^0_{202}+\mathbf{b}^1_{202}\right)-4(\mathsf{c}_{2,m})^2\left(\mathbf{b}^2_{202}+\mathbf{b}^4_{202}\right)\Big)\Big)\end{aligned} \tag{22}$$

6. Implementation

The construction of the $6m+1$ coefficients of the P_m-patch is mapped to one GPU pass. A second pass is used to render.

6.1. Construction on the GPU

The per-vertex computation of Section 5.1 maps to the GPU vertex shader and the per-patch computation of Section 5.2 to the geometry shader.

The **vertex shader** takes as input

- a *texture or vertex buffer* $\mathcal{M}$ listing all mesh points,
- a *texture* $\mathcal{I}$ listing for each mesh point the indices into $\mathcal{M}$ of its one-ring (see Figure 8),
- a *texture* listing consecutively for each one-ring all the scalars α_j as labeled in Figure 8, and
- an *input stream* containing for each mesh point its start-index into $\mathcal{I}$ and its valence.

Since every vertex has a different valence, and hence, a different number of points in its one-ring $\mathbf{p}_{0\ldots3n-1}$, the start-index avoids wasteful padding to the largest valence. The start-index is also used to look up the scalars α_0, α_1. The memory overhead of redundant one-ring indices in the case of quad or triangle facets is comparable to that of packing and has less control overhead; and subsequent redundant vertex loads are always from the cache.

The vertex shader outputs, for each one-ring,

- the points $\mathbf{v}$ and $\mathbf{f}_j$ for $j = 0\ldots n-1$ using (13);
- the vectors $\boldsymbol{\tau}_1$ and $\boldsymbol{\tau}_2$ using (15); and
- the valence n of $\mathbf{v}$.

We need only a single vertex shader for all the cases, but create two: one simplified to 4-valent vertices to be used only for ordinary patches. The scalars for a crease edge emanating from a polar vertex are passed to the geometry shader as the w-coordinate of the $\mathbf{f}_j$ vertices.

The **geometry shader** takes as input

- the output of the vertex shader
- an integer *texture* of indices j^k, $k = 0\ldots m-1$, for each face to pick its $\mathbf{f}_j$ and compute the m corner tangents (16).

We use a separate geometry shader for each patch type. If the facet is ordinary, the geometry shader simply streams out $\mathbf{v}^i$, $\mathbf{t}^i_0$, $\mathbf{t}^i_1$ and $\mathbf{f}^i_0$ for $i = 0,\ldots,3$ in order. Polar triangles output three fewer control points. If the facet is extraordinary, we specialize the geometry shaders to each m for efficiency. The

shader computes the $6m+1$ control points of the P_m-patch using (20), (21) (22) and streams them out. We note that the amplification in the geometry shader is minimal.

6.2. Evaluation and rendering on the GPU

The patches are rendered in a second pass by sending in a single pre-tessellated (u,v) domain using DirectX 10 instancing and evaluating the geometry and normal in the vertex shader. This leaves the geometry and pixel shaders available for further processing and lighting computations.

To render without pixel drop-out between ordinary bi-cubic patches and P_m-patches, we evaluate the external P_m-patch boundary as a cubic and keep the ordering of computations consistent. This guarantees a watertight construction of control points along shared boundaries. The corresponding if-statement at the end of the evaluation shader does not result in noticeable degradation in performance.

For bi-cubic patches and P_4-patches we use a uniform tessellation of the domain $[0,1]^2$, while for P_3-patches and polar patches, we use the standard triangular domain consisting of the half of $[0,1]^2$ with $u+v \leq 1$. For P_5-patches, we use the triangular domain with 5x instancing for each of its five sectors. In each P_m-patch case, the (u,v) domain is transformed to a local sector's domain before evaluation. We apply equations (2) and (3) only now to obtain separate patches in the form (1). Evaluation and normal computation of degree 4 triangular patches is comparable in cost to tensor-product bi-cubic patches: in the triangular case we have 15 control points and in the tensor-product case 16.

The rendering pass can be sped up and enhanced by the use of the X-Box 360 hardware tessellation unit [Lee06], which supports both triangles and quads and offers fast continuous adaptive tessellation for free.

7. Results

We implemented the algorithm on a 2.4Ghz quad-core CPU with 3GB RAM and 32-bit Windows Vista and a NVidia 8800 GT GPU using DirectX 10. Table 2 shows the performance of our implementation, without creases (14), on a variety of meshes (shown in Figure 13).

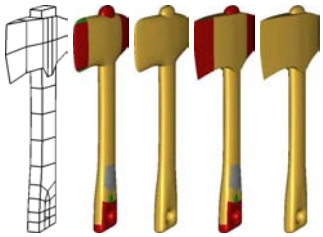
Patch construction on the 10 dodecahedra takes the least amount of time since it has the fewest facets. However, more points are evaluated for an P_m-patch as m increases. The frog was chosen due to its large number of extraordinary facets. The construction pass is fast enough to yield rendering at 659fps. Only when > 1000 points are evaluated per patch does the frame rate drop below 100.

In our implementation, we created a DirectX 10 'technique' for each patch case, but loaded and used only those that were required to render the current model. The apparent overhead associated with the load and use of an additional technique accounts for the poorer performance in the patch construction stage of the head and the monkey models, as compared to the frog model, which has 2-3 times as many facets but only quad patches. It is not clear whether in the short term one should consider merging the shaders for triangle, quad, or pentagon construction or wait for future hardware to mitigate this overhead.

Model	Verts	Faces	Number of each patch type				
			ord.	polar	3	4	5
10 Dodecahedra	200	120	0	0	0	0	120
Hand	405	417	304	33	0	74	7
Monkey	578	590	340	16	22	210	2
Frog	1308	1292	528	0	0	764	0

Model	Frames per second				
	N=5	9	17	33	65
10 Dodecahedra	780	780	675	265	72
Hand	480	480	390	231	63
Monkey	480	486	452	163	47
Frog	659	510	230	79	22

Table 2: ***Construction and rendering performance*** *on various models with each triangle, quad, and pentagon P_m-patch evaluated on a grid of size $\frac{1}{2}N\times(N+1)$, $N\times N$, and $\frac{5}{2}N\times(N+1)$, respectively.*



(a) (b) (c) (d) (e)

Figure 12: ***Axe with sharp creases.*** *(a) Mesh. (b),(c) Surface without crease support, and (d),(e) with crease scalar set to $\alpha = 1/10$ along the edge of the blade. The axe has bicubic patches (*yellow*), P_m-patches with $m = 3$ (*green*), $m = 4$ (*red*), and $m = 5$ (*gray*).*

Shape can be controlled locally using crease scalars α. Uniform adjustment everywhere allows turning a dodecahedron into its control polyhedron when $\alpha = 0$, or making it sphere-like when $\alpha = 4/5$ (Figure 10). Applied locally, creases allow creating sharp features, as in Figure 12, on an otherwise smooth object.

As illustrated in [NYM*08], the P_4-patch construction approximates the Catmull-Clark limit surface well (which is one of the reasons we took the P_4-patch as a starting point for developing the surface conversion).

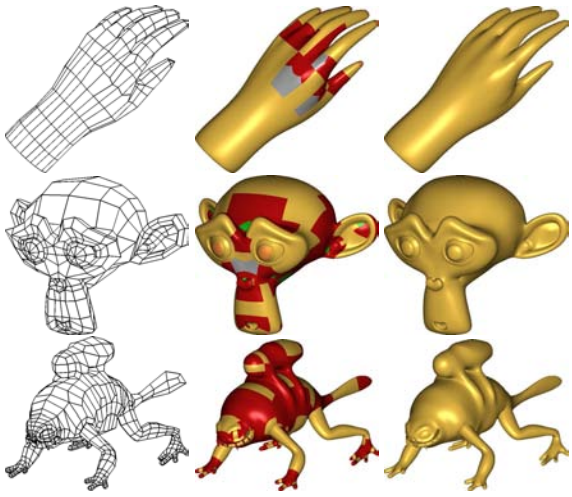
The **accompanying video** illustrates scripted animation on the frog, hand and monkey models evaluated with $N = 9$. The **electronic supplement** analyzes the surface shape for a gallery of test configurations. Except for the saddles, the highlight lines do not reveal patch transitions. No internal seams are visible in polar and P_m-patches.

8. Discussion of extensions, limitations and uses

Our approach would benefit from a larger buffer for intermediate vertex shader output to prevent vertex-shader re-computations; and from the ability to switch out the geometry shader without clearing this buffer. (We did not attempt to find an optimal mesh facet order to reduce vertex shader re-computation.) We restrict the number of facet sides to less than six since six is the maximum number of vertex shader inputs to the geometry shader and since we did not find local formulas that always result in good shape for m-sided facets when $m > 5$. The vertex shader's current 16 output vector streams in DirectX 10 limits the maximum valence to $n \leq 13$ (without packing the face vertices). While the valence at vertices is formally unrestricted, the inclusion of all three facet types allows the designer or remeshing algorithm to keep valences low, which is desirable, or isolate high-valence vertices in polar configurations. When sharper features skew the parametrization, distortions of the textures are avoided by applying the same sharpening rules to the texture coordinates.

The conversion fits well into a GPU morphing pipeline: only the texture or buffer containing the list of input mesh points needs to be updated before the surface construction pass. In particular, vertex buffer swapping constructs and renders scripted animations as fast as static ones.

Acknowledgments: This work was supported by NSF Grant CCF-0430891. Young In Yeo helped implement the technique. The work additionally benefited from CGAL's half-edge data structure and used Bay Raitt's monster frog and Blender's monkey (blender.org).



(a) (b) (c)

Figure 13: ***Hand, Monkey and Frog.*** *(a) Input mesh, (b) surface patches colored by type (ordinary in yellow, polar in orange, extraordinary triangle in green, extraordinary quad in red, and pentagons in gray), (c) final surface.*

References

[BS02] BOLZ J., SCHRÖDER P.: Rapid evaluation of Catmull-Clark subdivision surfaces. In *Proc. Web3D 2002*, ACM Press, pp. 11–17.

[BS07] BOUBEKEUR T., SCHLICK C.: Qas: Real-time quadratic approximation of subdivision surfaces. In *Pac. Gr. 2007* IEEE, pp. 453–456.

[Bun05] BUNNELL M.: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley, 2005

[CC78] CATMULL E., CLARK J.: Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design 10* (1978), 350–355.

[DKT98] DEROSE T., KASS M., TRUONG T.: Subdivision surfaces in character animation. In *SIGGRAPH 1998*, ACM Press, pp. 85–94.

[Far90] FARIN G.: *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, 1990.

[HB00] HAHMANN S., BONNEAU G.-P.: Triangular G^1 interpolation by 4-splitting domain triangles. *Computer Aided Geometric Design 17*, 8 (2000), 731–757.

[Lee06] LEE M.: Next generation graphics programming on Xbox 360, 2006.

[LKH08] LAI Y.-K., KOBBELT L., HU S.-M.: An incremental approach to feature aligned quad dominant remeshing. In *2008 ACM Symp. on Solid and Physical Modeling*, ACM, to appear.

[LS08] LOOP C., SCHAEFER S.: Approximating Catmull-Clark subdivision surfaces with bicubic patches. *ACM Trans. Graph. 27*, 1 (2008), 1–11.

[MKP07] MYLES A., KARČIAUSKAS K., PETERS J.: Extending Catmull-Clark subdivision and PCCM with polar structures. In *Pac.Gr. 2007*, IEEE, pp. 313–320.

[NYM*08] NI T., YEO Y. I., MYLES A., GOEL V., PETERS J.: Smooth surfaces from 4-sided facets. In *Proc. IEEE SMI* 2008

[Pet95] PETERS J.: C^1-surface splines. *SIAM Journal on Numerical Analysis 32*, 2 (1995), 645–666.

[Pet00] PETERS J.: Patching Catmull-Clark meshes. In *Siggraph 2000*, ACM Press, 255–258.

[PS04] PETERS J., SHIUE L.: Combining 4- and 3-direction subdivision. *ACM Trans. Graph. 23*, 4 (2004), 980–1003

[SJP05] SHIUE L.-J., JONES I., PETERS J.: A realtime GPU subdivision kernel. *ACM Trans. Graph. 24*, 3 (2005), 1010–1015.

[SL03] STAM J., LOOP C. T.: Quad/triangle subdivision. *Computer Graphics Forum 22*, 1 (2003), 79–86.

[Sta98] STAM J.: Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In *SIGGRAPH* (1998), pp. 395–404.

[SW05] SCHAEFER S., WARREN J. D.: On C^2 triangle/quad subdivision. *ACM Trans. Graph. 24*, 1 (2005), 28–36.

[VPBM01] VLACHOS A., PETERS J., BOYD C., MITCHELL J. L.: Curved PN triangles. In *I3D* (2001), ACM Press, pp. 159–166.