# GPU Conversion of Quad Meshes to Smooth Surfaces

Ashish Myles*
Young In Yeo†
Jörg Peters‡
University of Florida

## Abstract

We convert any quad manifold mesh into an at least $C^1$ surface consisting of bi-cubic tensor-product splines with localized perturbations of degree bi-5 near non-4-valent vertices. There is one polynomial piece per quad facet, regardless of the valence of the vertices. Particular care is taken to derive simple formulas so that the surfaces are computed *efficiently in parallel* and match up precisely when computed independently on the GPU.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

**Keywords:** real-time, GPU conversion, quad mesh, bi-cubic, bi-5 perturbation, spline, surface

## 1 Introduction

Current efforts of chipmakers to provide parallel multi-core or SIMD GPU platforms force CAD vendors to rethink their basic representation to take advantage of the increased computational power. This is as much an opportunity as a challenge. Clearly CAD cannot stray too far from established standards, such as the tensor-product B-spline or Bernstein-Bézier (BB-) representation when it comes to surface representation. However, there is an opportunity to organize these representations differently to avoid CPU-to-GPU transfer and to confine primitive explosion to the GPU, leaving the CPU free to do other tasks.

We propose here to augment the existing BB- (or B-spline) representations by a localized perturbation; specifically perturbing a bi-cubic (bi-3) spline by coefficients of a bi-5 polynomial piece (Figure 1). Due to the locality of the perturbation, this representation, the *p-rep*, is novel as a construction even though bi-5 $C^1$ freeform surface constructions are themselves not new (see e.g. [Shi et al. 2004]). While the derivation of the perturbation is non-trivial the resulting formulas are simple and yield a consistent localized construction suitable for SIMD parallelism. On the GPU, we construct the p-rep in one pass using the vertex shader and the geometry shader of the DirectX 10 graphics pipeline, streaming out the 32 p-rep coefficients. In a second pass, we use instancing to evaluate the patches on the vertex shader.

The p-rep

1. yields watertight $C^1$ surfaces from quad meshes;

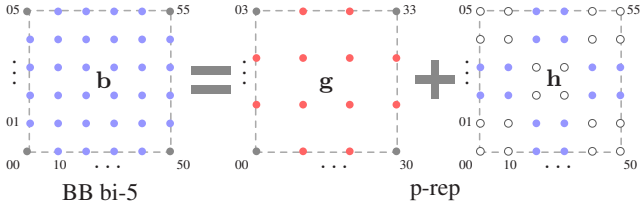


**Figure 1:** *Two representations of the degree bi-5 patch. (*left*) Standard Bézier form with* $6 \times 6$ *coefficients. (*right*) bi-3 patch with vertex-localized bi-5 perturbations (coefficients are the filled circles). The center four coefficients can be freely modified but our construction does not use this.*

2. does not require separation of non-4-valent vertices;
3. consists of bi-3 tensor-product splines with local offsets of degree bi-5 near non-4-valent vertices;
4. reduces to standard tensor-product bi-3 splines when all vertex valences are 4;
5. is designed to be both constructed and evaluated on the GPU, possibly preceded by GPU mesh deformation or evolution and followed by further computation (Figure 6);
6. localizes computation to maximize cache usage and avoid recomputation.

Section 2 introduces the two representations defining the patch to be constructed and discusses the derivations for $G^1$ continuity, which define the formulas for construction. Section 4 maps these formulas to the GPU. Results of an implementation and the discussion of design choices are in Sections 5 and 6.

### 1.1 Fast Conversion & Evaluation on the GPU

Computer Graphics has a head start on techniques for GPU acceleration. The natural recursive refinement which makes subdivision of quad meshes so intuitive to the designer is not a good match with deep graphics pipelines where recursion implies multiple passes. Evaluation via Stam's acceleration [Stam 1998] is used in modeling packages, but the overhead and size of the resulting shader routines are not appropriate for on-the-fly transformation of quad meshes. Using tables of surface fragments placed into textures, for *fixed valences and depth*, [Bolz and Schröder 2002] introduced a first efficient refinement method aimed specifically at the GPU. By carefully combining the fragments, the resulting mesh approximation of a Catmull-Clark surface can be made 'watertight', i.e. avoids pixel dropout [Bolz and Schröder ]. More recently, [Loop and Schaefer 2007] proposed an approach that allows arbitrarily fine evaluation on the fly, of a surface approximating the Catmull-Clark surface. The construction is conceptually easy and can be made watertight since it uses bi-3 (bi-cubic) patches everywhere. The construction generates just one geometry and two tangent patches per quad facet regardless of the valence of the vertices. This avoids CPU preprocessing of the quad mesh, used for example in [Bolz and Schröder 2002] and [Peters 2000] to separate extraordinary points, i.e. points

*e-mail:amyles@cise.ufl.edu
†e-mail:yiyeo@cise.ufl.edu
‡e-mail:jorg@cise.ufl.edu

that have fewer or more than 4 neighbors. This is possible since [Loop and Schaefer 2007] does not create a $C^1$ surface, but, in the spirit of bump mapping and PN triangles [Vlachos et al. 2001], creates a pair of tangent patches whose cross product is similar to but not identical to the normal of the bi-cubic geometric patch. Under OpenGL lighting, the differences to the true surface are typically difficult to spot. Clearly, while such approaches are great for computer graphics, they are not appropriate for engineering applications where the user will want to see exactly what is designed.

Bischoff et al. [Bischoff et al. 2000] proposed a forward-differencing method for evaluating Loop subdivision on uniform samples; Schaefer et al. [Schaefer and Warren 2007] precomputed tables for the exact evaluation of arbitrary subdivision schemes at rational parameter values by applying a classical technique of [Cavaretta et al. 1991] and proposed a dual method for adaptive tessellation; and Boo et al. [Bóo et al. 2001] suggest hardware for adaptive tessellation. Loop and Blinn [Loop and Blinn 2006] use the fragment shader to accurately render certain algebraic surfaces. Since fragment shaders were historically more powerful due to their texture access and buffer writing capabilities, the fragment shader has been used to implement subdivision refinement. Shiue et al. [Shiue et al. 2005] define data structures that allow for recursive subdivision with creases as several passes in the fragment shader and Bunnell [Bunnell 2005] provides code for a fast adaptive tessellation of subdivision surfaces. Since the fragment shader is the last shader in the graphics pipeline, applying additional fragment shaders and passes means reading back *after* the primitive explosion inherent in refinement. Generating the geometric primitives earlier in the graphics pipeline not only opens the possibility of a single pass (accelerated by a hardware tessellator) but yields also a conceptually cleaner data flow. Evaluation of trimmed NURBs surfaces on the GPU by [Guthe et al. 2005] and [Krishnamurthy et al. 2007], corresponds to our second pass, evaluation stage and is therefore complementary to our focus, which is the generation of the surface on the GPU.

## 2 The P-rep Patch

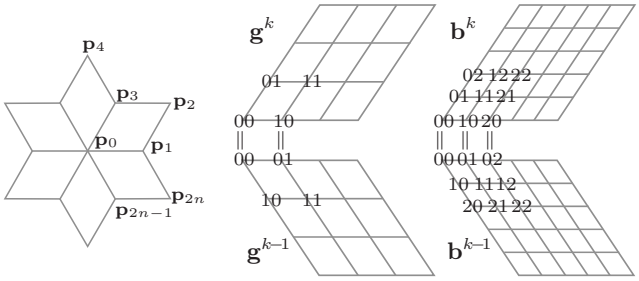


**Figure 2:** *Indices of (*left*) a one-ring of quad mesh points* $\mathbf{p}_i^{k+1}$ *at a vertex with valence* $n = 6$ *and (*right*) BB control points of two adjacent patches* $\mathbf{b}^k$.

Consider the standard tensor-product BB-representation [Farin 1988; Prautzsch et al. 2002]

$$b(u,v) := \sum_{i=0}^{d}\sum_{j=0}^{d} \mathbf{b}_{ij} \binom{d}{i}(1-u)^{d-i}u^i \binom{d}{j}(1-v)^{d-j}v^j. \tag{1}$$

For $d = 5$, $\mathbf{b}_{ij}^k \in \mathbb{R}^3$ is the $(i,j)^{\text{th}}$ BB control point of the $k^{\text{th}}$ patch $b^k(u,v)$ (Figures 1, *left*, and 2, *right*). We will verify smoothness of the construction in this representation in Section 3 but construct it below in *perturbation representation*, short **p-rep**, as a BB-patch $g^k(u,v)$ of degree $d = 3$ with BB control points $\mathbf{g}_{ij}^k \in \mathbb{R}^3$ plus a sparse bi-quintic perturbation $h^k(u,v)$ with BB points $\mathbf{h}_{ij}^k \in \mathbb{R}^3$ (Figure 1, *right*). The relationship between the representations is given by (10). If all four vertices of a quad have valence 4, we call it a B-quad and compute only $g$.

### 2.1 Construction at Vertices

Let $\{\mathbf{p}_k\}$ be the control points of the 1-ring of the design mesh at the vertex $\mathbf{p}_0$ of valence $n$. The p-rep interpolates the central limit point of Catmull-Clark subdivision for the 1-ring. That is (see Figure 2 for indices) [Halstead et al. 1993],

$$\mathbf{g}_{00}^k := \frac{\sum_{\ell=1}^{n}\left(n\mathbf{p}_0 + 4\mathbf{p}_{2\ell-1} + \mathbf{p}_{2\ell}\right)}{n(n+5)}, \quad k = 1\ldots n. \tag{2}$$

To have the tangent plane at $\mathbf{g}_{00}^k$ agree with that of Catmull-Clark subdivision, we define two vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ that span the tangent plane and express the tangent coefficients $\mathbf{g}_{10}^k$ in terms of these directions scaled by $\sigma \in \mathbb{R}$. With $\rho_n := \frac{1}{16}\left(c_n + 5 + \sqrt{(c_n+9)(c_n+1)}\right)$ the subdominant eigenvalue of Catmull-Clark subdivision and $d$ the degree of the BB-representation, e.g. $d = 3$ for $\mathbf{g}^k$, we set

$$\begin{aligned}
&w_n := 16\rho_n - 4, \qquad \sigma_n := \begin{cases} 0.53 & \text{if } n = 3, \\ 1/4\rho_n & \text{if } n > 3, \end{cases}\\
&c_n^k := \cos\frac{2\pi k}{n}, \quad s_n^k := \sin\frac{2\pi k}{n}, \quad c_n := c_n^1,\\
&\alpha_1 := w_n c_n^{j-1}, \qquad \beta_1 := c_n^{j-1} + c_n^j,\\
&\alpha_2 := w_n s_n^{j-1}, \qquad \beta_2 := s_n^{j-1} + s_n^j,\\
&\mathbf{e}_i := \frac{\sigma_n}{d(2+w_n)}\sum_{j=1}^{n}\left(\alpha_i\mathbf{p}_{2j-1} + \beta_i\mathbf{p}_{2j}\right), \text{ for } i = 1,2,\\
&\mathbf{g}_{10}^k := \mathbf{g}_{00}^k + \mathbf{e}_1 c_n^k + \mathbf{e}_2 s_n^k.
\end{aligned} \tag{3}$$

The formula makes explicit that, given $\mathbf{g}_{00}^k$, we need only compute $(\mathbf{e}_1, \mathbf{e}_2)$ in the vertex shader and hand it over to the geometry shader where a patch $k$ can generate its tangent coefficients, $\mathbf{g}_{01}^k$ and $\mathbf{g}_{10}^k$, for each corner by simple rotation. Finally, we compute

$$\mathbf{g}_{11}^k := \frac{1}{9}\left(4\mathbf{p}_0 + 2\left(\mathbf{p}_{2k+1} + \mathbf{p}_{2k+3}\right) + \mathbf{p}_{2k+2}\right). \tag{4}$$

We note that (2), (3) and (4) are to the standard bi-3 B-spline to BB-form conversion formulas for vertex, edge and face coefficients if the valence is $n = 4$.

### 2.2 Smoothness across Edges

Since we set $\mathbf{b}_{0i}^{k-1} = \mathbf{b}_{i0}^k$ along an edge between patches $b^k$ and $b^{k-1}$ (Figures 2, 3), the patches match continuously. The well-known sufficient symmetric conditions for $G^1$ continuity between two BB patches meeting along $b^k(u,0) = b^{k-1}(0,u)$, $u \in [0,1]$, are $\frac{\partial}{\partial u}b^k(u,0) \neq 0$, $\frac{\partial}{\partial v}b^k(u,0) \neq 0$, $\frac{\partial}{\partial w}b^{k-1}(0,u) \neq 0$ and, for some scalar-valued function $\lambda(u)$,

$$\frac{\partial}{\partial v}b^k(u,0) + \frac{\partial}{\partial w}b^{k-1}(0,u) = \lambda(u)\ \frac{\partial}{\partial u}b^k(u,0). \tag{5}$$

Since the conditions equate curves along an edge, we abbreviate

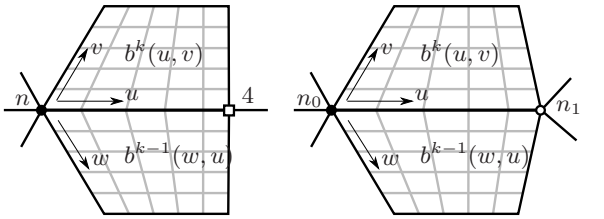


**Figure 3:** *Edge between two bi-5 patches (BB control nets in* gray*). The extraordinary vertices are denoted by • and ∘, while the valence 4 vertex is denoted by a □. (*left*) valence $\langle n, 4\rangle$, (*right*) valence $\langle n_0, n_1\rangle$.*

$\mathbf{g}^-_{ij} := \mathbf{g}^{k-1}_{ij}$, $\mathbf{h}_{ij} := \mathbf{h}^k_{ij}$, write a curve of degree $d$ with parameter $u$ and BB control points $\mathbf{c}_0, \ldots, \mathbf{c}_d$ as

$$\mathcal{B}\big(\mathbf{c}_0, \ldots, \tbinom{d}{i}\mathbf{c}_i, \ldots, \mathbf{c}_d\big) \text{ and use } \quad \langle \alpha, \beta\rangle$$

to indicate that the valence at one endpoint of the boundary curve is $\alpha$ and at the other endpoint is $\beta$. We define *endpoint-symmetry* as replacing (see Figure 2 )

$$(n_0, \mathbf{h}_{ij}, \mathbf{g}_{ij}, \mathbf{g}^-_{11}) \text{ by } (n_1, \mathbf{h}_{5-i,j}, \mathbf{g}_{3-i,j}, \mathbf{g}^-_{12}).$$

If the tuple is $\langle 4, 4\rangle$, we choose $\lambda = 0$ and the boundaries and cross-boundary derivatives of degree 3 defined by (2), (3), and (4) reduce to the standard B-spline to BB conversion formulas. In general, a valence 4 vertex needs to be handled with care to ensure that the $C^1$ conditions and not just $G^1$ conditions hold with the neighboring bi-3 patch. Hence we distinguish two cases (see Figure 3):

$$\langle n, 4\rangle \qquad \lambda(u) := \mathcal{B}(2c_n, 0, 0)\,, \tag{6}$$
$$\langle n_0, n_1\rangle \qquad \lambda(u) := \mathcal{B}(2c_{n_0}, -2c_{n_1}) \tag{7}$$

Both choices of parameterization result in $C^1$ conditions when both ends of the edge have valence 4. We have four cases and use for clarity $\mathrm{dot}([\mathbf{c}_0, \ldots], [\gamma_0, \ldots]) := \sum_{i=0} \mathbf{c}_i\gamma_i$.

**Case 1:** $n_0{=}n_1{=}4$: $\mathbf{h}_{20} := \mathbf{h}_{30} := \mathbf{h}_{21} := \mathbf{h}_{31} := (0, 0, 0)$.

**Case 2:** $n_0{\neq}4$ **and** $n_1{=}4$

$$\begin{aligned}
\mathbf{h}_{20} &:= \mathbf{h}_{30} := \mathrm{dot}\Big(\big[\,\mathbf{g}_{00},\,\mathbf{g}_{10},\,\mathbf{g}_{20},\,(\mathbf{g}_{11}{+}\mathbf{g}^-_{11})\,\big]\,,\\
&\qquad \tfrac{3}{40}\big[\,-2,\,-6/c_{n_0}{+}6,\,-4,\,3/c_{n_0}\,\big]\Big)\\
\mathbf{h}_{21} &:= (1 - 2c_{n_0}/5)\,\mathbf{h}_{20}+ \qquad (8)\\
&\quad \mathrm{dot}\Big(\big[\,\mathbf{g}_{00},\,\mathbf{g}_{10},\,\mathbf{g}_{20},\,\mathbf{g}_{30},\,(\mathbf{g}_{11}{+}\mathbf{g}^-_{11}),\,(\mathbf{g}_{21}{+}\mathbf{g}^-_{12})\,\big]\,,\\
&\quad \tfrac{3}{100}\big[\,2c_{n_0},\,12{-}6c_{n_0},\,6{+}2c_{n_0},\,2c_{n_0},\,-6,\,-3\,\big]\Big)\\
\mathbf{h}_{31} &:= \mathbf{h}_{30} + \mathrm{dot}\Big(\big[\,\mathbf{g}_{00},\,\mathbf{g}_{10},\,\mathbf{g}_{20},\,\mathbf{g}_{30},\,\mathbf{g}_{01},\,\mathbf{g}_{21},\,\mathbf{g}_{31},\,\mathbf{g}^-_{11},\,\mathbf{g}^-_{12}\,\big]\,,\\
&\quad \tfrac{3}{100}\big[\,-2{+}2c_{n_0},\,6{-}2c_{n_0},\,12{-}2c_{n_0},\,2{+}2c_{n_0},\,2,{-}9,{-}2,{-}6,{-}3\,\big]\Big)
\end{aligned}$$

**Case 3:** $n_0{=}4$ **and** $n_1{\neq}4$ is endpoint-symmetric to Case 2.

**Case 4:** $n_0{\neq}4$ **and** $n_1{\neq}4$

$$\begin{aligned}
\mathbf{h}_{20} &:= \mathrm{dot}\Big(\big[\,\mathbf{g}_{00},\,\mathbf{g}_{10},\,\mathbf{g}_{20},\,(\mathbf{g}_{11}{+}\mathbf{g}^-_{11})\,\big]\\
&\quad \tfrac{3}{40}\big[\,-2c_{n_1}/c_{n_0},\,4{+}(2c_{n_1}{-}6)/c_{n_0},\,-4,\,3/c_{n_0},\,3/c_{n_0}\,\big]\Big)\\
\mathbf{h}_{30} &\text{ is endpoint-symmetric to } \mathbf{h}_{20}.\\
\mathbf{h}_{21} &:= \Big(1 - \tfrac{3}{5}c_{n_0} - \tfrac{2}{5}c_{n_1}\Big)\,\mathbf{h}_{20} + \Big(\tfrac{3}{5}c_{n_0}\Big)\,\mathbf{h}_{30}+ \qquad (9)\\
&\quad \mathrm{dot}\Big(\big[\,\mathbf{g}_{00},\,\mathbf{g}_{10},\,\mathbf{g}_{20},\,\mathbf{g}_{30},\,(\mathbf{g}_{11}{+}\mathbf{g}^-_{11}),\,(\mathbf{g}_{21}{+}\mathbf{g}^-_{12})\,\big]\\
&\quad \tfrac{3}{100}\big[\,4c_{n_1},\,12{-}8c_{n_0},\,6{+}6c_{n_0}{-}4c_{n_1},\,2c_{n_0},\,-6,\,-3\,\big]\Big)\\
\mathbf{h}_{31} &\text{ is endpoint-symmetric to } \mathbf{h}_{21}.
\end{aligned}$$

## 3 Verification of Smoothness

Here we derive formulas for the coefficients $\mathbf{b}^k_{20}$, $\mathbf{b}^k_{21}$, $\mathbf{b}^k_{30}$, and $\mathbf{b}^k_{31}$ so that the $G^1$ conditions (5) hold. The formulas for $\mathbf{h}_{20}$, $\mathbf{h}_{21}$, $\mathbf{h}_{30}$ and $\mathbf{h}_{31}$ of the p-rep then follow from (see Figure 1)

$$\begin{aligned}
&\mathbf{g}_{00} = \mathbf{b}_{00}, \qquad \mathbf{g}_{10} = \tfrac{5}{3}\mathbf{b}_{10} - \tfrac{2}{3}\mathbf{b}_{00}, \qquad (10)\\
&\mathbf{g}_{11} = \tfrac{25}{9}\mathbf{b}_{11} - \tfrac{10}{9}\mathbf{b}_{10} - \tfrac{10}{9}\mathbf{b}_{01} + \tfrac{4}{9}\mathbf{b}_{00},\\
&\mathbf{h}_{20} = \mathbf{b}_{20} - \mathbf{a}_{20}, \quad \mathbf{h}_{21} = \mathbf{b}_{21} - \mathbf{a}_{21},
\end{aligned}$$

where $\mathbf{a}_{2i} := \mathrm{cub}_2(\mathbf{b}_{0i}, \mathbf{b}_{1i}, \mathbf{b}_{4i}, \mathbf{b}_{5i})$ and

$$\mathrm{cub}_2(\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_4, \mathbf{c}_5) := -\frac{3}{10}\mathbf{c}_0 + \mathbf{c}_1 + \frac{1}{2}\mathbf{c}_4 - \frac{1}{5}\mathbf{c}_5$$

takes the first two and the last two control points of a quintic curve and returns the third control point on the assumption that the quintic curve is a degree-raised cubic. That is, we obtain $\mathbf{h}_{2i}$ from $\mathbf{b}_{2i}$ by subtracting the cubic component $\mathbf{a}_{2i}$. Since the four central control points of $\mathbf{b}$ have no bearing on first order smoothness, we can choose their perturbations as zero. We abbreviate differences of control points as illustrated in Figures 4 and 5:

$$\mathbf{v}_i := \mathbf{b}^k_{i1} - \mathbf{b}^k_{i0}, \quad \mathbf{w}_i := \mathbf{b}^{k-1}_{1i} - \mathbf{b}^{k-1}_{0i}, \quad \mathbf{u}^{(5)}_i := \mathbf{b}^k_{i+1,0} - \mathbf{b}^k_{i0}.$$

The subsections below are labeled by the case in the geometry shader procedure (subsection 4.2). Case 1 need not be discussed because the perturbation is zero.

### 3.1 Cases 2 and 3: $\langle n, 4\rangle$

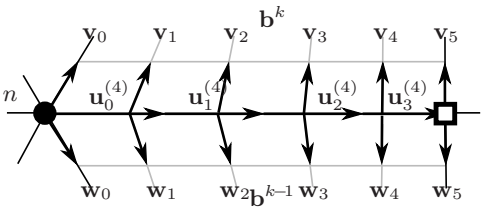


**Figure 4:** *Indices of control points of the derivatives along a shared boundary in Figure 3 $\langle n, 4\rangle$ case. The boundary is of degree 4 and $\mathbf{u}^{(4)}_i$ are its first differences.*

To preserve $C^1$ continuity at the valence 4 vertex corresponding to $\mathbf{b}^k_{50} = b^k(1, 0) = b^{k-1}(0, 1)$, we chose $\lambda(u) := \mathcal{B}(2c_n, 0, 0)$ and

chose the degree of the shared boundary $b^k(u,0) = b^{k-1}(0,u)$ to be 4 by setting

$$\mathbf{b}^k_{30} := \tfrac{1}{10}\mathbf{b}^k_{00} - \tfrac{1}{2}\mathbf{b}^k_{10} + \mathbf{b}^k_{20} + \tfrac{1}{2}\mathbf{b}^k_{40} - \tfrac{1}{10}\mathbf{b}^k_{50} \qquad (11)$$

and defining

$$\mathbf{u}^{(4)}_0 := \tfrac{5}{4}\left(\mathbf{b}^k_{10} - \mathbf{b}^k_{00}\right), \quad \mathbf{u}^{(4)}_1 := \tfrac{5}{3}\mathbf{b}^k_{20} - \tfrac{25}{12}\mathbf{b}^k_{10} + \tfrac{5}{12}\mathbf{b}^k_{00}, \qquad (12)$$
$$\mathbf{u}^{(4)}_2 := -\tfrac{5}{3}\mathbf{b}^k_{30} + \tfrac{25}{12}\mathbf{b}^k_{40} - \tfrac{5}{12}\mathbf{b}^k_{50}, \quad \mathbf{u}^{(4)}_3 := \tfrac{5}{4}\left(\mathbf{b}^k_{50} - \mathbf{b}^k_{40}\right).$$

Equating the 6 coefficients of the polynomial equation (5) is equivalent to the 6-tuple of equations

$$\begin{aligned} &5\,\mathcal{B}(\mathbf{v}_0 + \mathbf{w}_0, 5(\mathbf{v}_1 + \mathbf{w}_1), 10(\mathbf{v}_2 + \mathbf{w}_2), \\ &\qquad 10(\mathbf{v}_3 + \mathbf{w}_3), 5(\mathbf{v}_4 + \mathbf{w}_4), \mathbf{v}_5 + \mathbf{w}_5) \\ &= 5\,\mathcal{B}\Big(2c_n\mathbf{u}^{(4)}_0, 6c_n\mathbf{u}^{(4)}_1, 6c_n\mathbf{u}^{(4)}_2, 2c_n\mathbf{u}^{(4)}_3, 0, 0\Big). \end{aligned}$$

The first equation, $\mathbf{v}_0 + \mathbf{w}_0 = 2c_n\mathbf{u}^{(4)}_0$, is enforced by (2) and (3) and the last two equations, $\mathbf{v}_4 + \mathbf{w}_4 = 0$ and $\mathbf{v}_5 + \mathbf{w}_5 = 0$ hold by $C^1$ continuity at the valence 4 vertex. (Recall that the $G^1$ constraints simplify to $C^1$ constraints where the valence is 4). The remaining equations, corresponding to $i \in \{1,2,3\}$, are

$$2c_n\binom{3}{i}\mathbf{u}^{(4)}_i = \binom{5}{i}(\mathbf{v}_i + \mathbf{w}_i) = \binom{5}{i}(\mathbf{b}^{k-1}_{1i} + \mathbf{b}^k_{i1} - 2\mathbf{b}^{k-1}_{0i})$$

which simplifies to

$$0 = \mathbf{b}^{k-1}_{0i} - \frac{\mathbf{b}^{k-1}_{1i} + \mathbf{b}^k_{i1}}{2} + c_n\frac{(5-i)(4-i)}{25}\mathbf{u}^{(4)}_i. \qquad (13)$$

When $i = 1$, we insert (12) and solve for

$$\mathbf{b}^k_{20} := \tfrac{5}{4}\mathbf{b}^k_{10} - \tfrac{1}{4}\mathbf{b}^k_{00} - \frac{15}{c_n(5-i)(4-i)}\left(\mathbf{b}^k_{10} - \frac{\mathbf{b}^k_{11} + \mathbf{b}^{k-1}_{11}}{2}\right).$$

When $i = 4$, the assignment of $\mathbf{g}_{21}$ and $\mathbf{g}^-_{12}$ in the bi-3 construction already implies the constraint $\mathbf{v}_4 + \mathbf{w}_4 = 0$. For $i = 2, 3$, (13) follows from the assignments

$$\begin{aligned} \mathbf{b}^k_{21} &:= \mathbf{b}^k_{20} + c_n\tfrac{6}{25}\mathbf{u}^{(4)}_2 + \tfrac{1}{2}(\mathbf{a}^k_{21} - \mathbf{a}^{k-1}_{12}), \\ \mathbf{b}^{k-1}_{12} &:= \mathbf{b}^{k-1}_{02} + c_n\tfrac{6}{25}\mathbf{u}^{(4)}_2 + \tfrac{1}{2}(\mathbf{a}^{k-1}_{12} - \mathbf{a}^k_{21}) \end{aligned} \qquad (14)$$

$$\begin{aligned} \mathbf{b}^k_{31} &:= \mathbf{b}^k_{30} + c_n\tfrac{2}{25}\mathbf{u}^{(4)}_3 + \tfrac{1}{2}(\mathbf{a}^k_{31} - \mathbf{a}^{k-1}_{13}), \\ \mathbf{b}^{k-1}_{13} &:= \mathbf{b}^{k-1}_{03} + c_n\tfrac{2}{25}\mathbf{u}^{(4)}_3 + \tfrac{1}{2}(\mathbf{a}^{k-1}_{13} - \mathbf{a}^k_{31}) \end{aligned} \qquad (15)$$

where

$$\begin{aligned} \mathbf{a}^k_{21} &:= \mathrm{cub}_2(\mathbf{b}^k_{01}, \mathbf{b}^k_{11}, \mathbf{b}^k_{41}, \mathbf{b}^k_{51}) \\ \mathbf{a}^{k-1}_{12} &:= \mathrm{cub}_2(\mathbf{b}^{k-1}_{10}, \mathbf{b}^{k-1}_{11}, \mathbf{b}^{k-1}_{14}, \mathbf{b}^{k-1}_{15}) \end{aligned} \qquad (16)$$

and $\mathbf{a}^k_{31}$ and $\mathbf{a}^{k-1}_{13}$ are endpoint-symmetric to $\mathbf{a}^k_{21}$ and $\mathbf{a}^{k-1}_{12}$, respectively.

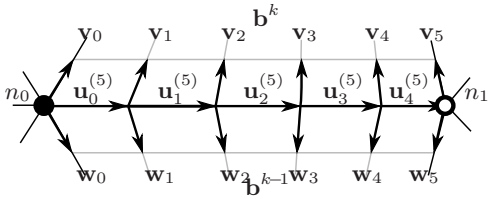


**Figure 5:** *Indices of control points of the derivatives along a shared boundary in Figure 3 in the $\langle n_0, n_1\rangle$ case.*

### 3.2 Case 4: $\langle n_0, n_1\rangle$

Since we chose $\lambda(u) := \mathcal{B}(\lambda_0, \lambda_1)$ where $\lambda_0 := 2c_{n_0}$ and $\lambda_1 := -2c_{n_1}$ (see Figure 3, *right*), the constraints are formally symmetric when endpoints are exchanged. Hence only the formulas for $\mathbf{b}^k_{i0}$ and $\mathbf{b}^k_{i1}$, $i \le 2$ need to be derived. Equating the coefficients of the polynomial equation (5) we arrive at the 6-tuple of equations

$$\begin{aligned} &5\,\mathcal{B}(\mathbf{v}_0 + \mathbf{w}_0, 5(\mathbf{v}_1 + \mathbf{w}_1), 10(\mathbf{v}_2 + \mathbf{w}_2), \\ &\qquad 10(\mathbf{v}_3 + \mathbf{w}_3), 5(\mathbf{v}_4 + \mathbf{w}_4), \mathbf{v}_5 + \mathbf{w}_5) \\ &= \mathcal{B}(\lambda_0, \lambda_1)\ 5\,\mathcal{B}\Big(\mathbf{u}^{(5)}_0, 4\mathbf{u}^{(5)}_1, 6\mathbf{u}^{(5)}_2, 4\mathbf{u}^{(5)}_3, \mathbf{u}^{(5)}_4\Big) \\ &= 5\,\mathcal{B}(\lambda_0\mathbf{u}^{(5)}_0, 4\lambda_0\mathbf{u}^{(5)}_1 + \lambda_1\mathbf{u}^{(5)}_0, 6\lambda_0\mathbf{u}^{(5)}_2 + 4\lambda_1\mathbf{u}^{(5)}_1, \\ &\qquad 4\lambda_0\mathbf{u}^{(5)}_3 + 6\lambda_1\mathbf{u}^{(5)}_2, \lambda_0\mathbf{u}^{(5)}_4 + 4\lambda_1\mathbf{u}^{(5)}_3, \lambda_1\mathbf{u}^{(5)}_4\Big). \end{aligned}$$

The first and last equations are satisfied by (2) and (3). Comparing the remaining coefficient pairs for $i \in \{1,2,3,4\}$ yields

$$\lambda_0\binom{4}{i}\mathbf{u}^{(5)}_i + \lambda_1\binom{4}{i-1}\mathbf{u}^{(5)}_{i-1} = \binom{5}{i}(\mathbf{v}_i + \mathbf{w}_i)$$

which simplifies to

$$0 = \mathbf{b}^{k-1}_{0i} - \frac{\mathbf{b}^{k-1}_{1i} + \mathbf{b}^k_{i1}}{2} + \frac{\lambda_0}{2}\frac{5-i}{5}\mathbf{u}^{(5)}_i + \frac{\lambda_1}{2}\frac{i}{5}\mathbf{u}^{(5)}_{i-1}$$

since $\frac{\lambda_0}{2} = c_{n_0}$ and $\frac{\lambda_1}{2} = -c_{n_1}$. We enforce the equation for $i = 1$ by substituting for $\mathbf{u}^{(5)}_1$ and solving for

$$\mathbf{b}^k_{20} := \mathbf{b}^k_{10} + \frac{1}{4c_{n_0}}\left(c_{n_1}\mathbf{u}^{(5)}_0 + 5\left(\frac{\mathbf{b}^k_{11} + \mathbf{b}^{k-1}_{11} - 2\mathbf{b}^k_{10}}{2}\right)\right). \qquad (17)$$

For $i = 2$, we can satisfy the equations by setting $\mathbf{b}^k_{21}$ and $\mathbf{b}^{k-1}_{12}$ using the following symmetric pair of equations.

$$\begin{aligned} \mathbf{b}^k_{21} &:= \mathbf{b}^k_{20} + c_{n_0}\tfrac{3}{5}\mathbf{u}^{(5)}_2 - c_{n_1}\tfrac{2}{5}\mathbf{u}^{(5)}_1 + \tfrac{1}{2}(\mathbf{a}^k_{21} - \mathbf{a}^{k-1}_{12}), \\ \mathbf{b}^{k-1}_{12} &:= \mathbf{b}^{k-1}_{01} + c_{n_0}\tfrac{3}{5}\mathbf{u}^{(5)}_2 - c_{n_1}\tfrac{2}{5}\mathbf{u}^{(5)}_1 + \tfrac{1}{2}(\mathbf{a}^{k-1}_{12} - \mathbf{a}^k_{21}) \end{aligned} \qquad (18)$$

where $\mathbf{a}^k_{21}$ and $\mathbf{a}^{k-1}_{12}$ are defined in (16).

## 4 P-rep construction on the GPU

We compute (2), (3) and (4) in the vertex shader and assemble and compute the p-rep according to (8) or (9) in the geometry shader. The result is streamed to the subsequent evaluation pass. The vertex list $\mathbf{P}$ can alternatively be a stream-out vertex buffer from a previous computation or a mesh evolution pass.

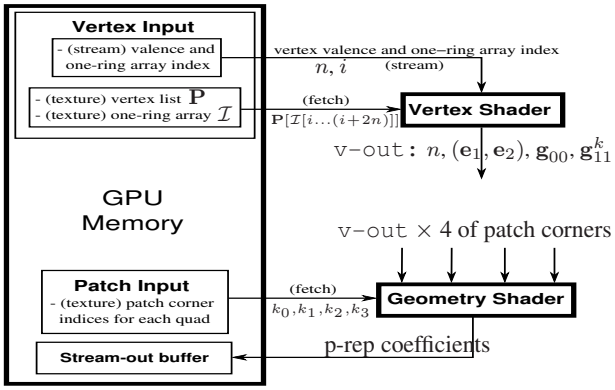


**Figure 6:** *Conversion pass in the GPU pipeline. The vertex shader computes* (2)*,* (3) *and* (4)*. The geometry shader computes the coefficients of the p-rep, (see* (8) *or p-rep,* (9)*).*

The conversion is the first of two passes. The second pass tessellates and renders the patches using instancing. That is, on input of the pre-tessellated domain and patch identifiers, the vertex shader loads, for each patch id and domain point, the appropriate control points and evaluates the patch.

The conversion pass is illustrated in Figure 6. For B-quads, we specialize the shaders to the case where all valences equal 4 to keep this simple case fast.

### 4.1 Vertex Shader: Consistent Tangents

**Input** passed to the vertex shader:
— an array (*texture*) $\mathbf{P}$ containing the vertices.
— A *texture* $\mathcal{I}$ containing, for each vertex, the indices of its one-ring of neighbors (see Figure 2) in $\mathbf{P}$, and
— an *input stream* of, for each vertex, its valence $n$ and index $i$ into the one-ring texture.
Since the number of vertices in the one-ring over a vertex depends on its valence, we use an index $i$ above for each vertex to pack all the one-ring indices into $\mathcal{I}$ (as opposed to wastefully padding to some maximum one-ring size and looking up based on the vertex id).

**Output:** For each vertex, the vertex shader outputs
— its valence $n$,
— the Catmull-Clark limit point $\mathbf{v} \in \mathbb{R}^3$ of that vertex,
— $(\mathbf{e}_1, \mathbf{e}_2) \in (\mathbb{R}^3, \mathbb{R}^3)$ used to compute $\mathbf{g}^k_{10} \in \mathbb{R}^3$ for $k \in \{0, 1, \ldots, n-1\}$ and
— $\mathbf{g}^k_{11} \in \mathbb{R}^3$ for $k \in \{0, 1, \ldots, n-1\}$.

**Procedure:** For each vertex, let $i$ be its one-ring-array index and $n$ its valence.
0. Fetch the one-ring $\mathbf{p}$ with indices $\mathcal{I}[i, \ldots, (i + 2n)]$ from the vertex list $\mathbf{P}$ and compute
1. the extraordinary point using (2),
2. $(\mathbf{e}_1, \mathbf{e}_2)$ using (3) with $d = 3$, and
3. $\mathbf{g}^k_{11}$ using (4).

### 4.2 Geometry Shader: Patch Assembly and Perturbation

**Input** of the geometry shader:
— all the streams output by the vertex shader, and
— a *texture* containing, for each patch, its integer indices $(k_0, k_1, k_2, k_3)$ around each of its four corner vertices.

**Output:** The shader specialized to B-quads outputs the 16 control points of $\mathbf{g}$. The general shader outputs an additional 16 control points for the perturbation $\mathbf{h}$.

**Procedure:** We first load the four patch indices $k_0, k_1, k_2, k_3$. The following computation has to be done for each of the four corners of $\mathbf{g}$ and $\mathbf{h}$. For simplicity, our discussion refers to the $\mathbf{g}_{00}$ corner and to the patch index $k_0$. The indices are as in Figure 2 with $\mathbf{g}$ playing the role of $\mathbf{g}^k$ and $\mathbf{g}^-$ the role of the preceding neighbor $\mathbf{g}^{k-1}$ (e.g. $\mathbf{g}^-_{11}$ and $\mathbf{g}^-_{12}$ are selected from the output of vertex shader Step 3 via the patch index).

**Bi-3 patch:** For the $\mathbf{g}_{00}$ corner,
1. assign to $\mathbf{g}_{00}$ the appropriate extraordinary point (from vertex shader step 1);
2. compute $\mathbf{g}_{10}$ and $\mathbf{g}_{01}$ using (3) with patch index $k_0$; and,
3. select $\mathbf{g}_{11}$ from the output of the vertex shader using the patch index $k_0$.

The other three corners of $\mathbf{g}$ are assembled using the obvious symmetries. The conversion for B-quads stops here. The full p-rep construction computes the

**Bi-5 perturbation:** Compute $\mathbf{h}_{20}$, $\mathbf{h}_{30}$, $\mathbf{h}_{21}$, and $\mathbf{h}_{31}$ according to $n_0$ and $n_1$ in (8), resp. (9).

## 5 Results

| Model | Verts | Faces | Frames per second | | | |
|---|---|---|---|---|---|---|
| | | | $N$=5 | 9 | 17 | 33 |
| 10 Triple Donuts | 400 | 440 | 524 | 456 | 249 | 104 |
| Shaft | 1107 | 1105 | 811 | 590 | 262 | 87 |
| Frog | 1308 | 1292 | 247 | 205 | 102 | 41 |

**Table 1:** *Frames per second for various models with each patch evaluated on a grid of size $N \times N$. The percentages of B-quads in the Triple Donut, Shaft, and Frog are 40.9, 90.6, and 40.9, respectively.*

We implemented the p-rep construction and subsequent evaluation in DirectX 10 on the NVidia GeForce 8800 GTX graphics card. For speed, we used specialized shaders to construct and evaluate the case without perturbation. Nevertheless, we tested our method on meshes with a small percentage of B-quads to test the efficiency of our method. Performance of the current implementation is demonstrated in Table 1 on the models in Figures 7 and 8. Availability of the Xbox 360 continuous adaptive tessellation unit [Lee ] will speed up the second pass and decrease the penalty for large tessellation factors $N$.

## 6 Discussion

The p-rep can be used both in the traditional CPU setting and, as we demonstrated, efficiently constructed on the GPU thanks to formulas like $(\mathbf{e}_1, \mathbf{e}_2)$ that are specifically developed to minimize recomputation and passing in the GPU pipeline. Keeping the patch in p-rep rather than in standard BB representation results in *watertightness* between neighboring bi-5 and bi-3 patches: the numerical round-off error is identical when computing patch boundaries independently from either side, because there is no perturbation along that edge and the boundary is computed identically by the vertex shader.

The presented approach fits well into a GPU pipeline where the GPU deforms the quad mesh and outputs to a stream-out vertex

buffer. This buffer can directly be read, in place of a texture, to create patches and render them with minimal CPU overhead and CPU-GPU transfer. Here, deformation of vertices with many dependencies should be confined to a separate pass for synchronization and to avoid re-computation. At the other end, the geometry shader is unused in the second pass. We are currently exploring its use for intersection testing and finite element computations.

## References


BISCHOFF, S., KOBBELT, L. P., AND SEIDEL, H.-P. 2000. Towards hardware implementation of loop subdivision. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM Press, New York, NY, USA, 41–50.

BOLZ, J., AND SCHRÖDER, P. Evaluation of subdivision surfaces on programmable graphics hardware. http://www.multires.caltech.edu/pubs/GPUSubD.pdf.

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.

BÓO, M., AMOR, M., DOGGETT, M., HIRCHE, J., AND STRASSER, W. 2001. Hardware support for adaptive subdivision surface rendering. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM Press, New York, NY, USA, 33–40.

BUNNELL, M. 2005. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley, Reading, MA, ch. Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping.

CAVARETTA, A. S., DAHMEN, W., AND MICCHELLI, C. A. 1991. Stationary subdivision. *Memoirs of the American Mathematical Society 93*, 453, 1–186.

FARIN, G. 1988. *Curves and Surfaces for Computer Aided Geometric Design — a Practical Guide*. Academic Press, Boston, MA.

GUTHE, 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.

KRISHNAMURTHY, A., KHARDEKAR, R., AND MCMAINS, S. 2007. Direct evaluation of nurbs curves and surfaces on the gpu. In *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling*, ACM, New York, NY, USA, 329–334.

LEE, M. 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, 2006.

LOOP, C., AND BLINN, J. 2006. Real-time GPU rendering of piecewise algebraic surfaces. *ACM Trans. Graph. 25*, 3, 664–670.

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 '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 255–258.

PRAUTZSCH, H., BOEHM, W., AND PALUZNY, M. 2002. *Bézier and B-Spline Techniques*. Springer Verlag.

SCHAEFER, S., AND WARREN, J. 2007. Exact evaluation of non-polynomial subdivision schemes at rational parameter values. In *Proceedings of Pacific Graphics 2007*.

SHI, X., WANGA, T., AND YU, P. 2004. A practical construction of $G^1$ smooth biquintic B-spline surfaces over arbitrary topology. *Computer Aided Design 36*, 5, 413–424.

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. 2001. Curved PN triangles. In *Proceedings of Symposium on Interactive 3D graphics*, 159–166.


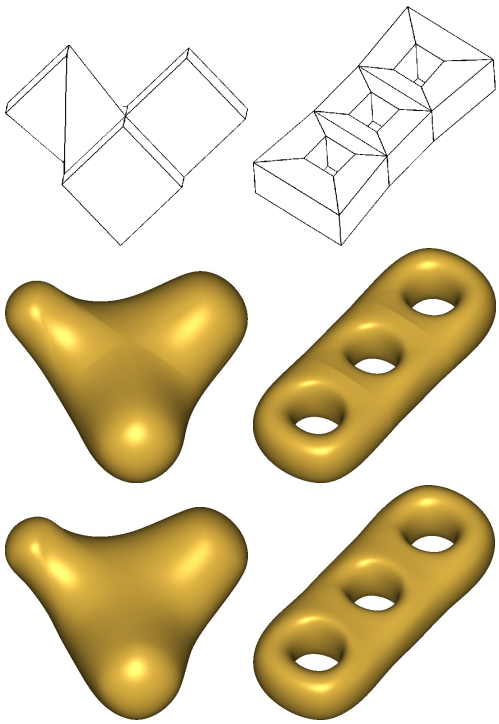

**Figure 7:** *Twist and Triple donut without (*middle*) and with (*bottom*) the bi-5 perturbation.*

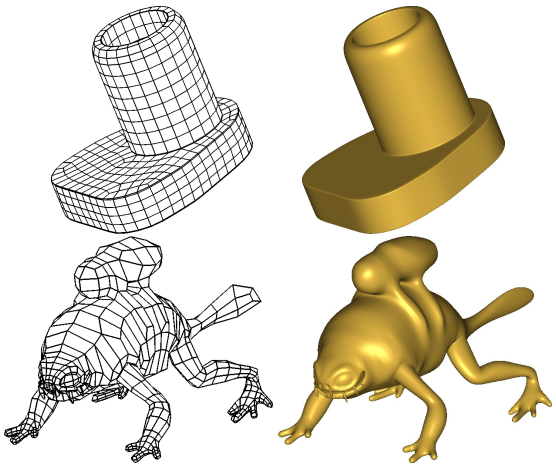

**Figure 8:** *Quad mesh and p-rep surface of Shaft and Frog.*