Real-Time Creased Approximate Subdivision Surfaces

Denis Kovacs[*] New York University
Jason Mitchell[†] Valve
Shanon Drone[‡] Valve
Denis Zorin[§] New York University

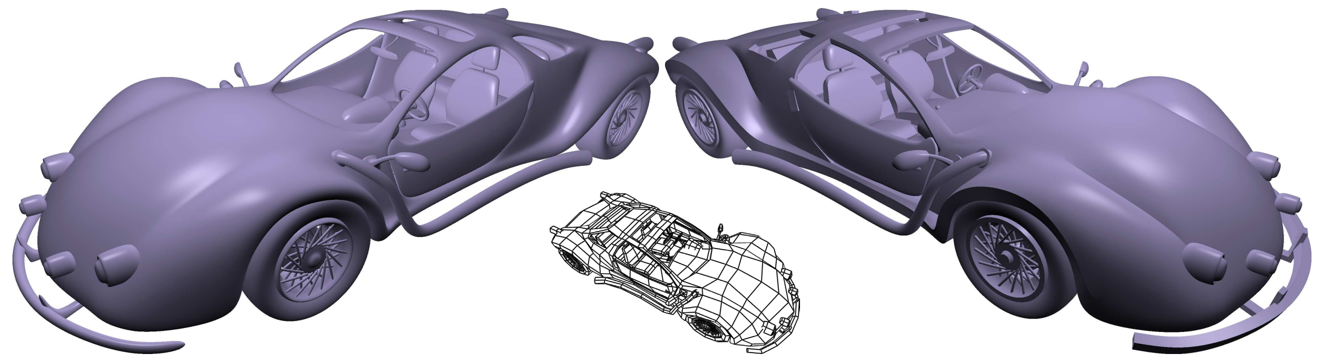

Figure 1: Left: *A car model rendered with smooth ACC.* Right: *The same model with creases and corners added.*

Abstract

We present an extension of recently developed Loop and Schaefer's approximation of Catmull-Clark surfaces (ACC) for surfaces with creases and corners which are essential for most applications. We discuss the integration of ACC into Valve's Source game engine and analyze performance of our implementation.

CR Categories: I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling

Keywords: subdivision surfaces, geometric modeling, GPU tessellation, hardware rendering, video games

1 Introduction

Catmull-Clark subdivision surfaces are the dominant higher-order surface type used in feature films, particularly in the area of character modeling [Catmull and Clark 1978] [DeRose et al. 1998]. Modeling with Catmull-Clark surfaces is familiar and intuitive to artists and the limit surface behaves well when the control mesh is animated. As a result, the real-time graphics community has become very interested in using Catmull-Clark subdivision surfaces and their approximations. A simple and efficient, yet high-quality bicubic approximation of Catmull-Clark surfaces (ACC) suitable for integration into existing game engines was recently introduced in [Loop and Schaefer 2008a].

Our goal is to extend ACC to support piecewise smooth surfaces with creases and boundaries with corners. Such surfaces are common in applications: most models found in computer games contain such features, which makes it essential to retain high visual surface quality of ACC near such points. The original paper presents a construction for a smooth boundary (excluding a common situation described in section 5). This construction can be used anywhere except at vertices where multiple creases meet, near corner vertices on the boundaries, or on interior creases.

There are two important differences between corner points and interior points:

- Catmull-Clark surfaces may not have well-defined tangents at corner points, and the tangents of modified Catmull-Clark surfaces [Biermann et al. 2000] turn out to be unsuitable for use in tangent Bezier patches for many meshes;
- Vertices of the control mesh tagged as corners need to be interpolated.

We demonstrate shading artifacts that result from using incorrect tangents at corner vertices, and present a formula for tangents that leads to good visual quality.

Interpolation of corner control points may result in artifacts ("overhangs") both in subdivision surfaces and their bicubic approximation. We discuss how these can be avoided. In addition to extending ACC, we briefly discuss integration of our implementation with a production game engine, Valve's Source engine (Section 7). We present performance comparisons of instanced versus native tessellation as well as a comparison with results published by [Ni et al. 2008] for a C^1 scheme in Section 8.

2 Related Work

Our work is a direct extension of the work of Loop and Schaefer [Loop and Schaefer 2008a] and we refer the reader to that paper for a more detailed discussion of related work; here we present a brief summary. The central idea of using separate tangent and position fields to define visually smooth geometry without constructing a C^1 surface explicitly was introduced in [Vlachos et al. 2001]. A number of algorithms were proposed in the past to generate smooth (typically C^1) piecewise polynomial surfaces, but few attempted to match the visual quality of Catmull-Clark surfaces. A C^1 (almost everywhere C^2) patch approximation of Catmull-Clark surfaces was proposed in [Peters 2000], but requires one or two additional subdivision steps. G^2-continuity is achieved in [Loop and Schaefer 2008b] but requires evaluation of relatively complex high-order patches. Techniques for direct evaluation of subdivision surfaces on GPUs [Bolz and Schröder 2002; Shiue et al. 2005] require

[*]e-mail: kov@cs.nyu.edu
[†]e-mail: jasonm@valvesoftware.com
[‡]e-mail: sdrone@valvesoftware.com
[§]e-mail: dzorin@mrl.nyu.edu

additional subdivision steps or multiple passes [Bunnell 2005].

An important recent approach for smoothing quad meshes with C^1 patches is [Ni et al. 2008]. In this work, all extraordinary quads (with at least one vertex of valence $\neq 4$) are converted to four triangular patches of total degree 4 each. As we target relatively low polygon count models common in games, most faces in such models tend to be extraordinary, which makes the lower complexity of control point setup and patch evaluation of ACC more appealing in our setting.

Subdivision surfaces with creases were introduced in [Hoppe et al. 1994], and rules for interior-independent creases and corners that ensure tangent plane continuity were introduced in Biermann [Biermann et al. 2000]. As we discuss in Section 4, these rules do not necessarily meet the needs of our application. Boubekeur [Boubekeur et al. 2005] adds smooth crease curves to PN triangles as well as additional parameters for crease shape control, but does not consider corners.

Our preferred implementation method is **instanced tessellation**, one of the two primary means for implementing tessellation on graphics hardware, available in shader model 3.0 hardware with vertex texture fetch capabilities such as NVIDIA GeForce 8x00 and ATI RADEON HD 2x00. The main alternative is **native tessellation** (ATI RADEON HD 2x00 and XBox 360) [Lee 2006] [Tatarchuk 2007]. In instanced tessellation, rendering multiple pretessellated meshes containing parametric and index data makes it possible to use the vertex shader to evaluate tessellated surface position [Forsyth 2003]. In [Grün 2005] and [Ni et al. 2008] hardware instancing is used to accelerate rendering of PN-triangles and a patch-based C^1 surface construction respectively.

For the upcoming release of DirectX 11, Microsoft has proposed adding three pipeline stages after the vertex shader: the *hull shader*, the *tessellator* and the *domain shader* [Gee 2008]. Developers will typically map vertex animation operations such as skinning to the vertex shader, basis transformations such as ACC to the hull shader and higher order surface evaluation to the domain shader.

3 Bicubic Approximation of Catmull-Clark surfaces

We briefly review the construction of [Loop and Schaefer 2008a] to set up the notation. *Geometry patches* are Bezier patches of bidegree 3, defined by a grid of 16 control points (Figure 2): 4 *corner*, 8 *edge* and 4 *interior* points. Their positions are determined using fixed-weight masks depending on the valence (see [Loop and Schaefer 2008a] for mask definitions). The weights are chosen so that each edge point is the midpoint of two adjacent interior points, and each corner point is the centroid of the adjacent endpoints of all nearby patches.

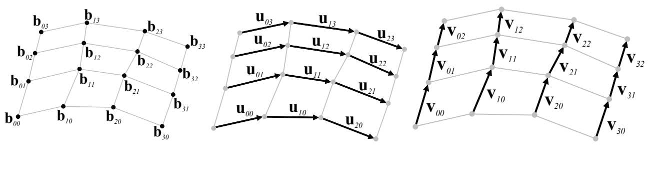

Figure 2: *Control points of geometry patches and control vectors of tangent patches.*

For boundaries, the weights for corners and edge points on the boundary are chosen to produce a B-spline curve on the boundary, with the exception of vertices of valence 2, which are forced to be corners (i.e., have 2 distinct tangents). The interior points are determined in the same way as for patches non-adjacent to the boundary, with the boundary vertex regarded as an interior vertex of valence $2k$, where k is the number of incident patches, which we call *face valence*. Informally, a smooth boundary vertex is regarded as interior vertex with a half-ring of incident patches.

Tangent patches are Bezier patches of degree (2,3), with 12 control vectors (Figure 2). Separate patches ∂_u and ∂_v are defined for parametric directions u and v; the formulas used for control vectors are the same, so we consider only the ∂_v tangent patch. The *corner vectors* are obtained using Catmull-Clark tangent mask weights applied to the ring of edge and face neighbor vertices of the corner point, with signs reversed for corners $\mathbf{v}_{02}$ and $\mathbf{v}_{32}$. For boundary points, tangent masks of modified the Catmull-Clark scheme [Biermann et al. 2000] are used.

All *edge* and *interior* control vectors are obtained from the geometry patch directly using the standard formula $\mathbf{v}_{ij} = 3\big(\mathbf{b}_{i,j+1} - \mathbf{b}_{ij}\big)$, excluding edge control vectors along the edges with two control points. These vectors are defined using correction factors that ensure tangent field continuity

$$\begin{aligned}\mathbf{v}_{1j} &= 3\big(\mathbf{b}_{1,j+1} - \mathbf{b}_{1j}\big) + \frac{1}{3}\big(2c_0\mathbf{u}_{1j} - c_1\mathbf{u}_{0j}\big)\\ \mathbf{v}_{2j} &= 3\big(\mathbf{b}_{2,j+1} - \mathbf{b}_{2j}\big) + \frac{1}{3}\big(2c_0\mathbf{u}_{2j} - c_1\mathbf{u}_{1j}\big)\end{aligned} \qquad (1)$$

We rederive these formulas in a slightly more general form to construct tangent fields at corner vertices in Section 4. Interior and edge control vectors are computed in the same way for patches adjacent to creases or boundaries.

4 Creases and corners

A control mesh for a piecewise smooth surface of the type described in [Biermann et al. 2000] has a number of edges tagged as *crease edges*. A vertex of a crease edge can be tagged as either a *crease smooth vertex* (default), or a *crease corner vertex*. Crease corner vertices are interpolated, and the crease curves may have two distinct tangents at the corner. We consider only the case of interior-independent sharp crease curves, which are completely defined by the control points on the crease.

If a vertex has more than two incident crease edges, we always tag it as a corner, so that it is interpolated, although some of the incident crease curves may have common tangents. In this setup, locally near a crease vertex, we can split the control mesh into independent parts (sectors), each of which can be treated as a surface with boundary (Figure 3). For smooth crease vertices, the rules of Loop and Schaefer are used, excluding the case of face valence 2, for which Loop and Schaefer use a special-case corner rule.

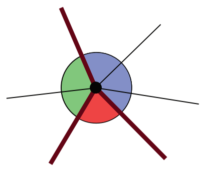

Figure 3: *Three sectors defined at a crease vertex*

To define corner rules, we first consider what can be regarded as the desired behavior at corner vertices.

Surface behavior near crease corners. Intuitively, one expects the smooth surface to "follow" the control mesh; this natural requirement, combined with crease independence from the interior leads to unexpected difficulties at crease corner vertices. If one requires the surface to have a well-defined tangent plane at corners, and the tangent curves do not depend on control points away from the mesh crease, then the tangents of the two crease curves meeting at the corner determine the tangent plane of the surface. Furthermore, there are two types of possible local surface behavior [Biermann et al. 2000]: convex and concave corners (Figure 4). One can observe that neither of these options matches the control mesh behavior: one intuitively expects the normal to the surface to be as close as possible to being perpendicular to the incident mesh edges. In contrast, at interior vertices, the tangent masks effectively average the tangent directions, so the resulting normal can be regarded as the average of normals of planes spanned by all possible pairs of incident edges.

The situation is substantially different at corner vertices at the crease, as the crease independence requirement forces a tangent plane independent of interior control points. For the type of surface shown in Figure 4, this results in surface normals nearly parallel to mesh edges. For Bezier-interpolated normals, the problem is further exacerbated (Figure 5). Two possible approaches to resolving this contradiction are:

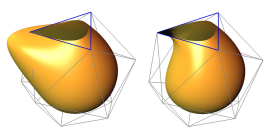
Figure 4: *Left: concave corner. Right: convex corner. [Biermann et al. 2000]*

- Relax the tangent plane continuity requirement at crease corner vertices: introduce *cones*, so that a different normal corresponds to each edge direction at the corner vertex, but the normal is continuous everywhere else;
- Allow normal directions that are dependent on interior control points.

Depending on the desired appearance, either option may be suitable. However, in the context of ACC, the first option turns out to be unusable because of the limitations of the Bezier representation.

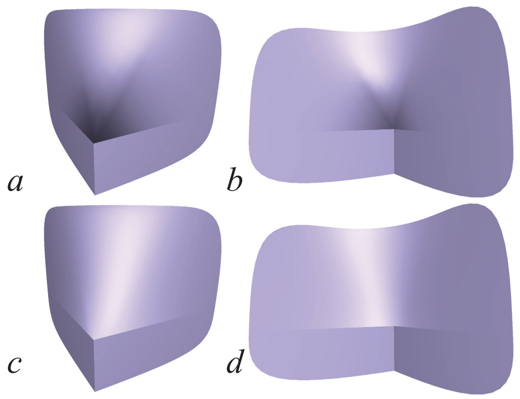

Figure 5: *Comparison of different tangent definitions for a corner: a,b: Edge tangents are computed as linear combinations of two crease tangents (modified Catmull-Clark). c,d: Our scheme used for tangent control vectors for the same control meshes.*

Cones. Making corner crease vertices into cones presents two difficulties: first, there are a few smooth configurations for which a cone is not the best possible behavior. Second, even more importantly, *it is impossible to produce a single-point normal discontinuity with nondegenerate tangent Bezier patches.* Indeed, tangents $\mathbf{t}_i(s)$, $i = u, v$, along the boundary of a patch parametrized by s, are given by cubic or quadratic polynomials, so (non-unit length) normal $\mathbf{n}(u) = \mathbf{t}_u \times \mathbf{t}_v$ is a polynomial of degree 5. The condition that normals on two sides of the boundary are collinear can be expressed as $\mathbf{n}(u) \times \hat{\mathbf{n}}(u) = D(u) = 0$, where $\hat{\mathbf{n}}$ is the normal computed for the same boundary curve for the adjacent patch. $D(u)$ is a polynomial of degree at most 10. If the normals are not collinear and nondegenerate at the corner vertex $u = 0$, then $D(u) \neq 0$ at $u = 0$, and can vanish at most at 10 points along the boundary, which makes normal continuity at *all* boundary points away from the corner vertex impossible. $D(0) = 0$ implies either collinear normals (which means the point is not a cone), or singular parameterization. It is clear how one can construct cones by collapsing control points of a Bezier patch to a single control point at one of the patch boundaries.

As singular parametrization is highly undesirable for tessellation (especially based on instancing as discussed in Section 7), and texture mapping, we consider the latter option impractical.

Interior-dependent tangent patches. We adopt the second possible option, that is, introduce dependency of normals at crease corners on the mesh interior. Recall that the primary reason to use an interior-independent construction for edge and corner points on the crease is to ensure that patches on different sides of creases match perfectly. At the same time, the normals are discontinuous across creases, so no such constraint is essential for tangent patches, although it is still desirable that the normals on the crease depend only on control points on the same side. This observation leads to the following overall approach:

- Compute geometry patches in the same way as boundary patches in [Loop and Schaefer 2008a], but make sure that the crease corner point $\mathbf{c}$ is interpolated;
- Define a suitable tangent plane P for $\mathbf{c}$;
- Project crease edges incident at $\mathbf{c}$ to P, and obtain tangents for interior edges incident at c by interpolation of two normals.

To define the tangent plane at interior and smooth crease vertices, the tangents are obtained by applying fixed modified Catmull-Clark tangent weights, and the normal is computed from the tangents. In the case of corners, this solution is not available, as the modified Catmull-Clark surface of [Biermann et al. 2000] the tangent plane is spanned by the two crease tangents, and the original Catmull-Clark surface in general is not tangent-plane continuous.

Instead, we use the average of the normals to geometry patches directly. If we choose the indices in each patch so that the crease corner vertex is $\mathbf{b}_{00}$, and number patches from 0 to k,

$$\mathbf{n} = \mathrm{norm}\Big(\sum_{i=0}^{k-1} \mathrm{norm}\big((\mathbf{b}_{10}^{(i)} - \mathbf{b}_{01}^{(i)}) \times (\mathbf{b}_{10}^{(i)} - \mathbf{b}_{00}^{(i)})\big)\Big) \qquad (2)$$

where $\mathrm{norm}(\mathbf{x}) = \mathbf{x}/|\mathbf{x}|$. In the case when normals average to zero, there is no meaningful common normal, and we use the cross-product of two tangents.

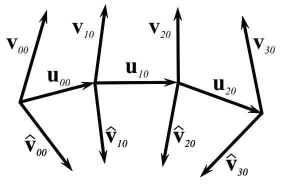

Figure 6: *Control vectors of tangent fields* $\mathbf{u}(t)$, $\mathbf{v}(t)$ *and* $\hat{\mathbf{v}}(t)$.

While this direct procedure is more expensive than computing a linear average of tangents using fixed weights, we found that the results are substantially better for all choices of averaging of tangents with which we have experimented. If the expense of this calculation is a concern, for all control point configurations we have examined in practical models just averaging the normal to the first and last geometry patches still yields a tangent plane superior to other alternatives.

Tangent interpolation. Once the tangent plane P with unit normal $\mathbf{n}$ is defined, we project the crease tangents $\mathbf{t}_0^{init} = \mathbf{p}_0 - \mathbf{c}$ and $\mathbf{t}_k^{init} = \mathbf{p}_k - \mathbf{c}$, where $\mathbf{p}_0$ and $\mathbf{p}_k$ are two crease vertices adjacent to $\mathbf{c}$, and k is its face valence, to the tangent plane P to obtain two basis tangents $\mathbf{t}_i^P = \mathrm{norm}\big(\mathbf{t}_i^{init} - (\mathbf{n} \cdot \mathbf{t}_i^{init})\mathbf{n}\big)$, $i = 0, k$. Next, we interpolate these tangents to obtain corner control vectors for each tangent patch incident at the crease corner vertex, in a way that insures tangent plane continuity.

We use a slightly more general form of patch continuity conditions of [Loop and Schaefer 2008a] used to derive (1). Consider three tangent fields defined on an edge e shared by two patches: the quadratic ∂_u field $\mathbf{u}(t)$, and two cubic ∂_v fields $\mathbf{v}(t)$ and $\hat{\mathbf{v}}(t)$ (Figure 6). To ensure normal field continuity across the edge, these three fields have to be linearly dependent at each t. We can multiply $\mathbf{u}(t)$ by any linear function $(a+bt)$ without changing its direction. If three linear polynomials $(a+bt)\mathbf{u}(t)$, $\mathbf{v}(t)$, $\hat{\mathbf{v}}(t)$ are linearly dependent, but pairwise independent (which is the case whenever there are no degeneracies in the patches), we can write the dependency condition as $(a+bt)\mathbf{u}(t) = c\mathbf{v}(t) + d\hat{\mathbf{v}}(t)$, with $c, d \neq 0$. As a and b are arbitrary, we can set c to one. As the choice of order between $\mathbf{v}$ and $\hat{\mathbf{v}}$ is arbitrary, it is natural to require that $c = d$, and c can be chosen to be 1. As for polynomials to coincide, their Bezier points have to coincide, we arrive at the conditions of a form similar to [Loop and Schaefer 2008a], but with undefined a and b.

$$a\mathbf{u}_{00} = \mathbf{v}_{00} + \hat{\mathbf{v}}_{00},\ b\mathbf{u}_{20} = \mathbf{v}_{30} + \hat{\mathbf{v}}_{30} \tag{3}$$

$$\frac{1}{3}\left(b\mathbf{u}_{00} + 2a\mathbf{u}_{10}\right) = \mathbf{v}_{10} + \hat{\mathbf{v}}_{10},\ \frac{1}{3}\left(2b\mathbf{u}_{10} + a\mathbf{u}_{20}\right) = \mathbf{v}_{20} + \hat{\mathbf{v}}_{20} \tag{4}$$

As in [Loop and Schaefer 2008a], equations (4) can be satisfied by a suitable choice of $\mathbf{v}_{10}$, $\mathbf{v}_{20}$, $\hat{\mathbf{v}}_{10}$, and $\hat{\mathbf{v}}_{20}$ as functions of other control vectors. We index the edges e incident at the crease corner $\mathbf{c}$ counterclockwise starting with a crease edge. The vectors $\mathbf{u}_{00}$, $\hat{\mathbf{v}}_{00}$, and $\mathbf{v}_{00}$ are tangents along an edge e_j, the previous edge e_{j-1} and the next edge e_{j+1}, respectively. If we denote them by $\mathbf{t}_j$, $\mathbf{t}_{j-1}$ and $\mathbf{t}_{j-1}$, each of the equations (3) has the form

$$\mathbf{t}_{j+1} = a_j \mathbf{t}_j - \mathbf{t}_{j-1}$$

$j = 1 \ldots k-1$, where k is the face valence of $\mathbf{c}$. In [Loop and Schaefer 2008a] two choices of formulas for $\mathbf{t}_j$ (Catmull-Clark tangent formulas for interior vertices and smooth boundaries) satisfy these equations for a fixed a. Three additional natural assumptions determine a unique answer: (1) $a_j = a$ independent of j; (2) $\mathbf{t}_0 = \mathbf{t}_0^P$ and $\mathbf{t}_k = \mathbf{t}_k^P$ so that the tangent control vectors on the crease are aligned with the crease curve tangents as well as possible (3) if these tangents are of equal length, we expect all inferred t_j $j = 1 \ldots k$ are of equal length. In this case,

$$\mathbf{t}_i = \frac{1}{\sin\theta}\left(\sin\frac{(i-k)\theta}{k}\mathbf{t}_0^P + \sin\frac{i\theta}{k}\mathbf{t}_k^P)\right) \tag{5}$$

where θ is the angle between the tangents $\mathbf{t}_0$ and $\mathbf{t}_k$. The angle θ is measured in counterclockwise direction if we look at the tangent plane from the direction of the averaged normal $\mathbf{n}$. Equation (5) leads to numerical difficulties if θ is close to π, although resulting tangent vectors are well-behaved even in the limit $\theta = \pi$. If t_0 and t_k are normalized to be of the same length, which we take to be the average of their lengths, one can write this expression in a less symmetric but more stable form as

$$\mathbf{t}_i = \cos\frac{i\theta}{k}\mathbf{t}_0^P + \sin\frac{i\theta}{k}\mathbf{n} \times \mathbf{t}_0^P \tag{6}$$

Formulas (2) and (6) define tangent control vectors at crease corner vertices. We have found this approach to be quite robust and insensitive to perturbations and degenerate cases. Examples of applying these formulas are shown in Figures 5 and 11 (left).

5 Smooth Creases for Face Valence 1

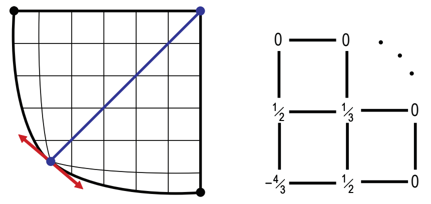

Figure 7: *Face valence 1 crease vertices.*

Smooth creases can be thought of as boundaries inside the mesh. Loop and Schaefer describe geometry patch stencils and limit tangents for boundaries, but choose to define a crease vertex of valence 2 to be a corner, defining a special-case rule. However, we have observed in practice (Figure 7) that it is often desirable to have smooth boundaries with face valence 1 vertices.

Defining the crease to be a uniform B-Spline curve as in the other cases results in the problem that now the tangents and bitangents of the geometry patch are collinear and no longer define a normal.

The limit normal, however, still exists and can be found by considering tangents along the curve $\gamma(s) = B(s,s)$ where $B(u,v)$ is the Bezier patch. A direct computation shows that up to higher order terms, the tangent is $\gamma'(s) = s\mathbf{t}_3 + O(s^2)$, where $\mathbf{t}_3$ can be computed from the control points using the mask in Figure 7. This allows us to obtain the normal to the surface at the smooth vertex with $k = 1$ as the cross product of the tangent to the crease curve and $\mathbf{t}_3$.

In practice, one can avoid a special-case code for the normal computation by perturbing the boundary tangents instead, setting them to $\mathbf{t}_0 + \varepsilon\mathbf{t}_3$ and $\mathbf{t}_1 + \varepsilon\mathbf{t}_3$ respectively. We use $\varepsilon = 10^{-4}$. As a result, the tangent control vectors at the boundary are no longer perfectly collinear; we could not find any cases where a perturbation of this magnitude would cause artifacts, yet it is sufficiently large to make the normal computation stable. The result of applying this procedure on a car model are shown in Figure 11(right).

6 Artifacts and how to fix them

Just as for subdivision surfaces, many different types of visual quality defects can be identified. We address the two most significant types of artifacts that occur near corners.

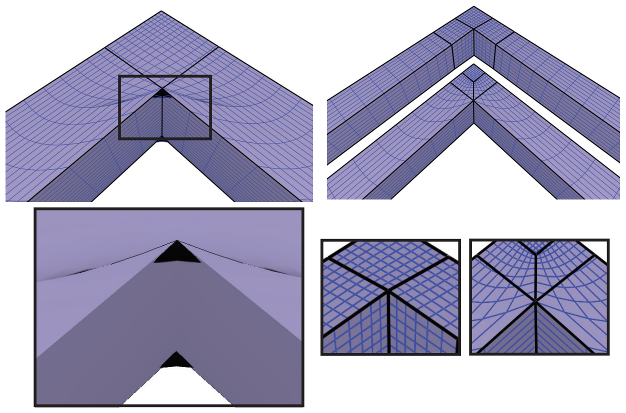

Figure 8: *Left: An ACC surface with an overhang. Right: a modification of the control mesh eliminating the overhang.*

Overhangs. Crease corner points are interpolated by an ACC surface, while nearby noncorner mesh vertices are moved to averaged positions of their neighbors. This, in turn, affects placement of interior and edge Bezier points near a corner. As the edge Bezier points determine tangent directions at patch boundaries, for highly nonuniform meshes, extreme shifts of tangents may result in patches with angles between tangents exceeding 180 degrees. For reasons discussed in [Biermann et al. 2000], a Bezier patch, which always has convex corners in the parameteric domain, cannot have a concave corner, so the patch folds over and approaches the boundary curves from the smaller angle side, developing *overhangs* shown in Figure 8a. The problem disappears if the designer chooses a more uniform set of quads near the crease corner or increases its valence (Figure 8b).

Lack of smoothness near convex corners. For highly non-planar corners of low valence, in some cases, the bicubic patches cannot approximate the behavior of the subdivision surface well, even if a least-squares fit is used as close as possible to the surface. In this case, the angle between actual normals of geometry patches is significant and cannot be fully masked by using tangent patches (Figure 9). In this case, increasing the valence of the crease corner or decreasing the size of adjacent faces solves the problem.

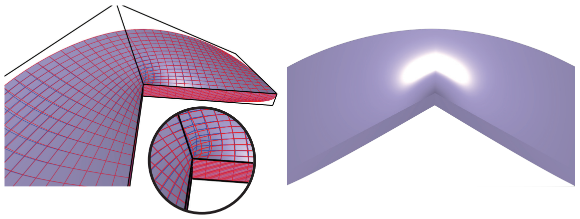

Figure 9: *Nonsmooth appearance near a corner. Note the mismatch between the Catmull-Clark surface (red) and bicubic patches (blue), and the sharp angle between parameteric lines on adjacent patches.*

7 Implementation

We have implemented the above extensions to ACC in Valve's Source engine. We have mapped the DirectX 11 pipeline onto DirectX 9, including instanced and native tessellation codepaths where the vertex shader and hull shader are implemented in software on the CPU and the remaining stages are executed on the GPU.

The CPU-side vertex shading operations include skinning, vertex morphing and other operations which are appropriate to perform at the control mesh level. Post-transform control mesh vertices are then sent to the software hull shader where they are mapped to a set of Bezier patches using our technique. This data is copied asynchronously to GPU memory. Domain points are instantiated with appropriate mesh connectivity on the GPU. The vertex shader—playing the role of *domain shader*—evaluates Bezier patch point positions and tangent frames using Bezier control points fetched from the floating point texture generated earlier by the CPU-side hull shader.

ATI's hardware tessellator instantiates the vertex shader at u, v points in the $[0..1]$ domain and provides the shader with access to all of the "super-primitive" data from the input vertices [Lee 2006] [Tatarchuk 2007]. The shader can use the input super-primitive data and the Bezier patch data to evaluate patch attributes. The remainder of the graphics pipeline is unchanged, so an implementor need only alter existing vertex shaders and vertex buffer layouts.

It is also possible to emulate tessellation by using instancing hardware to replicate a pretessellated patch across the input mesh, one instance for each ACC patch [Grün 2005] [Ni et al. 2008]. The vertex shader then evaluates the bicubic patch in the same manner as in the native tessellated version.

8 Performance

In Table 1, we compare instanced and native tessellation performance of the datasets shown in Figure 10 using the ATI RADEON 4870 X2, which is able to run both codepaths. Due to differences in the hardware interfaces, the native tessellation shader uses roughly 16% more instructions than the instanced patch shader. In our measurements, we have seen that the instanced patch performance is frequently as much as twice as fast as the native hardware tessellation performance. We conclude that the performance improvement seen when using instanced patches is not entirely related to shader length, but rather, related to driver or hardware implementation. (Both perform the same number of texture operations.) We have also compared our systems performance to that of [Ni et al. 2008] as shown in Table 2. (In this case, we have used a GeForce 8800 GT in order to remain consistent with the hardware used by Ni et al.)

	Native Tessellation			**Instanced Tessellation**		
Mesh	N=3	N=9	N=15	N=3	N=9	N=15
Car	1344	1296	589	1550	1301	846
Ship	1245	326	137	1196	473	222
Poly	747	160	65	532	304	132

Table 1: ***Performance Comparisons*** *- The Car, Ship and Poly models contain 1164, 5180 and 10618 quad faces. Performance numbers are in frames per second, measured on an Intel Quad Core Q9450 2.66GHz and ATI RADEON 4870 X2. N = number of tessellated vertices per control mesh edge.*

	Instanced Results			**Ni et al.**		
Mesh	N=9	N=17	N=33	N=9	N=17	N=33
Sword	1480	1480	539	965	965	703
Frog	728	256	63	392	226	87

Table 2: *Instanced results using our technique for the Sword and Frog datasets which contain 138 and 1292 quad faces respectively. Performance data from [Ni et al. 2008] is shown on the right.*

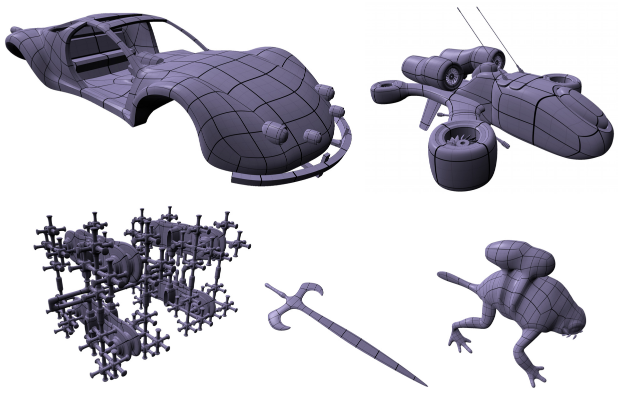

Figure 10: *Car, Ship, Poly, Sword and Rocket Frog models*

Using NVPerfHUD, we have determined that both the instanced and native hardware tessellation shaders are vertex texture fetch bound. Each invocation of the domain shader performs 30 fetches of packed control point data (12 for the control points, and 9 for each of the two tangent patches). For regular patches (with all vertices of valence 4), we can avoid fetching the tangent patch control points and simply use the de Casteljau algorithm to compute positions and normals. This saves 18 texture fetches for these patches at the expense of drawing regular and extraordinary patches with two API calls rather than one. In this case, we measured a 20% (1.9ms) performance improvement in GPU evaluation cost at N = 33 for the rocket frog mesh by splitting evaluation of regular and extraordinary patches. In addition, we avoid calculating tangent patches for regular patches when converting from Catmull-Clark to ACC. This saves an additional 0.26 ms on the rocket frog.

9 Future Work

We are currently working on layering of displacement maps onto ACC patches to provide high frequency detail. Additionally, we intend to address adaptive subdivision as this will be critical for performance and LOD management, particularly if we move past characters and into terrain rendering.

While overall we have obtained good results with extended ACC, the artifacts such as those discussed in Section 6 require designers

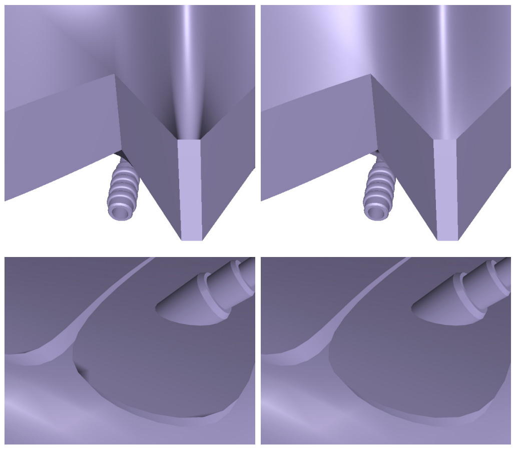

Figure 11: *Top: an example of corner artifacts on the spaceship model (left) eliminated by our technique (right). Bottom: an example of face valence 1 smooth crease vertex with vanishing normal (left) and with the normal computed using our technique (right).*

to adapt the models; while some of the limitations are fundamental (it is impossible to create cones or approximate well the subdivision surface near certain types of corners without refinement), one can hope to design techniques to deal with some of these problems automatically; this is a promising direction for future work.

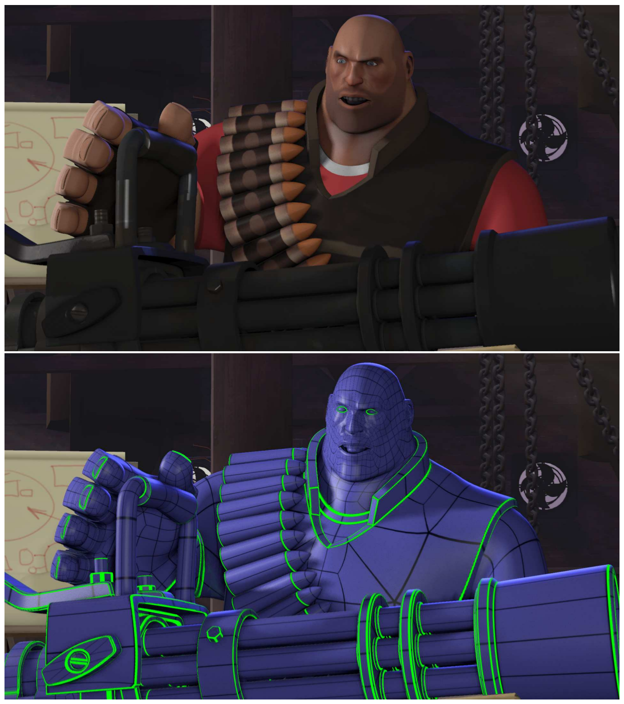

Figure 12: *Character from the game* Team Fortress 2 *modeled as a Catmull-Clark subdivision surface and rendered with our technique. The character and his weapon contain sharp features which require crease support to render correctly. In the second image, the black lines indicate patch edges with tagged crease edges highlighted in green.*

References

BIERMANN, H., LEVIN, A., AND ZORIN, D. 2000. Piecewise Smooth Subdivision Surfaces With Normal Control. 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, 113–120.

BOLZ, J., AND SCHRÖDER, P. 2002. Rapid Evaluation of Catmull-Clark Subdivision Surfaces. In *Proceedings of the seventh international conference on 3D Web technology*, ACM New York, NY, USA, 11–17.

BOUBEKEUR, T., REUTER, P., AND SCHLICK, C. 2005. Scalar Tagged PN Triangles. In *EUROGRAPHICS 2005 (Short Papers)*, Eurographics.

BUNNELL, M. 2005. Adaptive Tessellation of Subdivision Surfaces With Displacement Mapping. *GPU Gems 2*, 109–122.

CATMULL, E., AND CLARK, J. 1978. Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer-Aided Design*, 350–355.

DEROSE, T., KASS, M., AND TRUONG, T. 1998. Subdivision Surfaces in Character Animation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 85–94.

FORSYTH, T. 2003. Practical Displacement Mapping. In *Game Developers Conference*.

GEE, K. 2008. DirectX 11 Tessellation. In *Microsoft GameFest*.

GRÜN, H. 2005. Efficient Tessellation on the GPU Through Instancing. *Journal Of Game Development 1*, 3.

HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J., AND STUETZLE, W. 1994. Piecewise Smooth Surface Reconstruction. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 295–302.

LEE, M. 2006. Next-Generation Graphics Programming on XBox 360. In *Microsoft GameFest*.

LOOP, C., AND SCHAEFER, S. 2008. Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches. *ACM Trans. Graph. 27*, 1, 1–11.

LOOP, C., AND SCHAEFER, S. 2008. G^2 Tensor Product Splines Over Extraordinary Vertices. In *Proceedings of the 2008 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*.

NI, T., YEO, Y. I., MYLES, A., GOEL, V., AND PETERS, J. 2008. GPU Smoothing of Quad Meshes. In *IEEE International Conference on Shape Modeling and Applications*.

PETERS, J. 2000. Patching Catmull-Clark Meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 255–258.

SHIUE, L., JONES, I., AND PETERS, J. 2005. A Realtime GPU Subdivision Kernel. *Proceedings of ACM SIGGRAPH 2005 24*, 3, 1010–1015.

TATARCHUK, N. 2007. Real-Time Tessellation on the GPU. In *SIGGRAPH Advanced Real-Time Rendering in 3D Graphics and Games Course*, ACM.

VLACHOS, A., PETERS, J., BOYD, C., AND MITCHELL, J. 2001. Curved PN Triangles. In *I3D 2001: Proceedings of the 2001 Symposium on Interactive 3D Graphics*, ACM, New York, NY, USA, 159–166.