Curved PN Triangles

Alex Vlachos[1] Jörg Peters[3*] Chas Boyd[2] Jason L. Mitchell[1]

[1] ATI Research [2] Microsoft Corporation [3] University of Florida

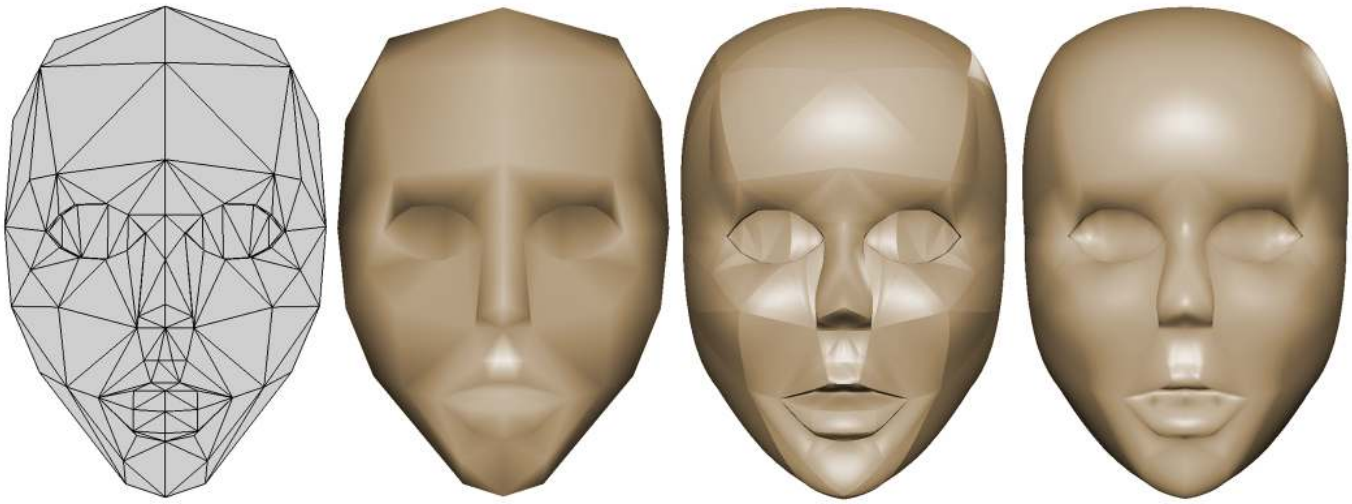

Figure 1: Which rendering would you prefer? (*from left to right*) (a) Input triangulation, (b) Gouraud shaded input triangulation, (c) geometric component of the PN triangles (shaded according to surface normal variation) (d) curved PN triangles (shaded with independently constructed quadratically varying normals).

Abstract

To improve the visual quality of existing triangle-based art in real-time entertainment, such as computer games, we propose replacing flat triangles with curved patches and higher-order normal variation. At the hardware level, based only on the three vertices and three vertex normals of a given flat triangle, we substitute the geometry of a three-sided cubic Bézier patch for the triangle's flat geometry, and a quadratically varying normal for Gouraud shading. These curved point-normal triangles, or PN triangles, require minimal or no change to existing authoring tools and hardware designs while providing a smoother, though not necessarily everywhere tangent continuous, silhouette and more organic shapes.

CR Categories: I.3.5 [surface representation, splines]: I.3.6—graphics data structures

Keywords: curved surface, PN Triangle, hardware, real-time, surface tessellation

*NSF NYI CCR-9457806, jorg@cise.ufl.edu, chasb@microsoft.com, AVlachos@ati.com, JasonM@ati.com

1 Introduction

In interactive 3D entertainment, software design cycles are long and hardware product release cycles are short. As a result, new techniques added to hardware must have a simple migration path if content already in development is to use the technique; conversely, the less change a new hardware feature imposes on the art pipeline and API, the more likely it is to be used.

A majority of computer games use triangles as their fundamental modeling primitive, and all use triangles for hardware-accelerated rendering. This is the result of existing tool support, hardware support and the artist's familiarity with triangles. It is increasingly common for applications to pass normals to the hardware to take advantage of transform-capable video cards.

In this paper, we introduce *curved point-normal triangles*, or short *PN triangles*, as an inexpensive means of improving visual quality by smoothing out silhouette edges and providing more sample points for vertex shading operations. Each PN triangle replaces one original flat triangle by a curved shape that is retriangulated into a programmable number of small (flat) subtriangles. We select this representation for a resource-limited hardware environment: the PN triangle is to be generated on-chip without any software assistance. Specifically, PN triangle generation and subtriangulation are to be inserted between the vertex-and-primitive-assembly stage and the vertex-shading stage of the graphics pipeline (Figure 2).

The geometry of a PN triangle is defined as one cubic Bézier patch. The patch matches the point and normal information at the vertices of the flat triangle. Its *normal is a separate* linear or quadratic Bézier interpolant of the data. No additional data beyond the position and normal data are used. Other data attached to the flat input triangle are forwarded to the subtriangles, mostly by linearly

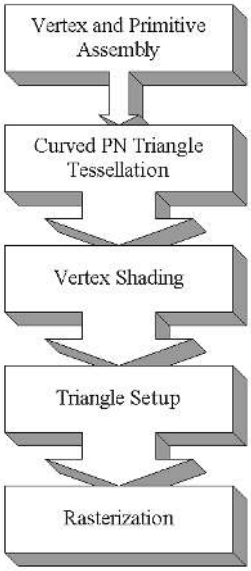

Figure 2: Curved PN triangle tessellation in the graphics pipeline

interpolating the values at the vertices of the flat input triangle.

The PN triangle emerged in comparison with more sophisticated surfacing methods to address requirements specific to the resource-limited hardware environment, e.g. that no information about neighboring triangles can be accumulated:

1. Improved visual appeal.

2. Compatibility with existing API data structures: vertex arrays plus triangle index streams where triangles arrive in unpredictable order. Minimal or no modification of existing APIs.

3. Backward compatibility of the models with hardware that does not support PN triangles; minimal or no effort to adapt existing models.

4. No setup step of the application, API, or hardware driver; in particular, hardware should not have to provide intermediate storage for random access; consequently *knowledge about neighboring facets is restricted to the shared vertices and vertex normals on the common edge*.

5. Applicability to shapes of arbitrary topology.

6. Fast execution and simple implementation in hardware.

In particular, it is *not* the purpose of PN triangles to convert triangulations into designer quality, everywhere smooth surfaces, but to soften triangle creases and improve the visual appeal by generating *smoother silhouettes and better shading*.

The following sections will define the API-level usage and placement of the curved PN triangles in the graphics pipeline (Section 2), give a precise derivation of the PN triangle and its properties (Section 3), and then discuss performance and visual impact to convince the reader that the requirements are well addressed by curved PN triangles.

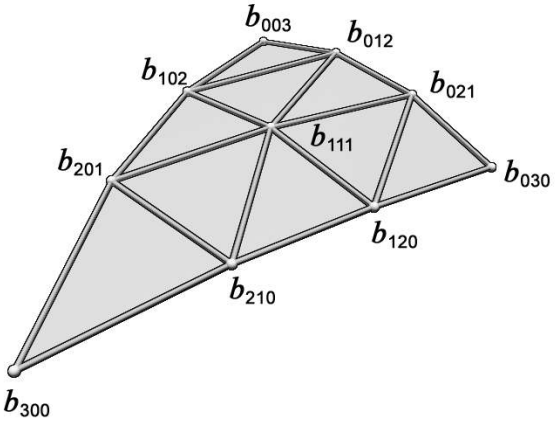

Figure 3: The coefficients or control points of a triangular Bézier patch arranged to form a control net.

2 Basic form, API-level usage and placement in the graphics pipeline

We want to leverage the existing (flat) triangle-based data structures such as *vertex arrays*: a stream of point and normal coordinates followed by sets of indices that allow picking the vertices of a triangle from the stream. That is, for each triangle, only the xyz coordinates of the position and normal of the three vertices of the triangles are available and there is a unique normal for each vertex.

An interesting aspect of the curved PN triangle that may surprise the non graphics-immersed reader is that the *normal component of the curved PN triangle is independently specified* from the geometric component of a curved PN triangle. The geometry of a curved PN triangle defined by a cubic patch b (c.f. Figure 3)

$$
\begin{aligned}
b: \quad & R^2 \mapsto R^3, \quad \text{for } w = 1-u-v, \quad u,v,w \geq 0 \\
b(u,v) &= \sum_{i+j+k=3} b_{ijk} \frac{3!}{i!j!k!} u^i v^j w^k, \\
&= b_{300} w^3 + b_{030} u^3 + b_{003} v^3 \\
&\quad + b_{210} 3w^2 u + b_{120} 3w u^2 + b_{201} 3w^2 v \\
&\quad + b_{021} 3u^2 v + b_{102} 3wv^2 + b_{012} 3uv^2 \\
&\quad + b_{111} 6wuv.
\end{aligned}
$$

We group the b_{ijk} together as

vertex coefficients:	$b_{300}, b_{030}, b_{003},$
tangent coefficients:	$b_{210}, b_{120}, b_{021}, b_{012}, b_{102}, b_{201},$
center coefficient:	$b_{111}.$

Coefficients are also often called *control points* and are connected to form a control net or control polyhedron (see Figure 3). The normal component of a curved PN triangle is either a linear average of the vertex normals or a quadratic function of the positional and

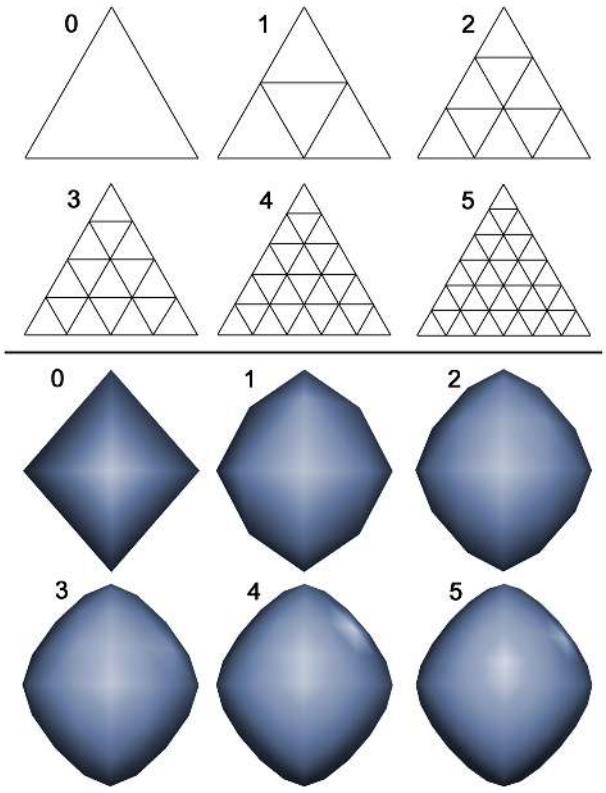

Figure 4: Subtriangulation (`lod` 0 through 5) induced by a uniform u, v grid (*top*) and the visual effect of two light sources on an octahedron (*bottom*).

normal data. The quadratic function n is defined as (c.f. Figure 7)

$$\begin{aligned} n: \quad & R^2 \mapsto R^3, \text{for } w = 1-u-v, \quad u,v,w \geq 0 \\ n(u,v) &= \sum_{i+j+k=2} n_{ijk} u^i v^j w^k, \\ &= n_{200} w^2 + n_{020} u^2 + n_{002} v^2 \\ &\quad + n_{110} wu + n_{011} uv + n_{101} wv. \end{aligned}$$

We call n_{110}, n_{011} and n_{101} mid-edge coefficients. The values of n are normalized before they are passed on to the vertex shading stage of the pipeline.

Section 3 describes in detail how to choose the coefficients b_{ijk}, respectively n_{ijk} so that the patches match the position and normal at each of the three corners of the original flat triangle. The PN triangle is a special three-sided, or "triangular Bézier" patch. Three-sided patches were invented by de Casteljau [3]. Farin [6] gives a thorough treatment of de Casteljau's patch and its applications.

Given the simple, closed form of the patches it is straightforward to *evaluate and tesselate* b and n. In particular, we can efficiently subtriangulate both patches over the same uniform u, v grid by prescribing just one number, the *level of detail* `lod`, defined as the number of evaluation points on one edge minus two (c.f. Figure 4). If `lod`=0 then the patch is just evaluated at the corners and this is equivalent to sending the original flat triangle down the graphics pipeline. An *application developer* enables PN triangles by a call that specifies `lod` and a flag that indicates whether n or the linearly varying normal is to be evaluated (c.f. reffig:oct).

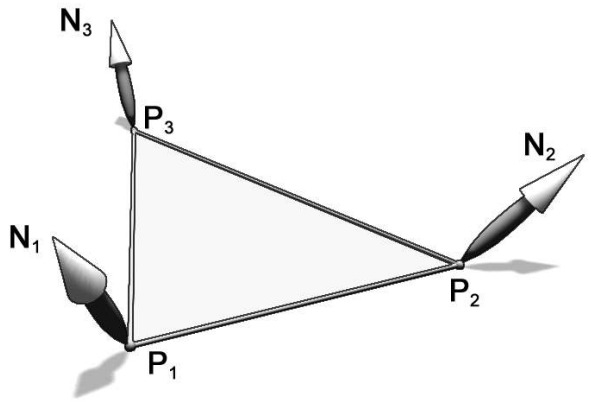

Figure 5: Input data: points P_i and normals N_i.

2.1 Why choose cubically varying geometry and quadratically varying normals?

In opting for cubic geometry and quadratic normals, we balanced the simplicity needed for hardware realization with a modeling range that would allow us to improve contours and shading.

Simplicity means that we do not generate and store neighborhood information, as would be required by generalized subdivision algorithms (e.g. [10, 8]) or surface splines [13]. Simplicity made us opt for a closed-form, polynomial surface representation that is fast to evaluate with little memory use. While it is possible to derive smoothly joining patches without being able to look at the neighbor (by deriving a unique normal along the common boundary for each of the abutting patches) the corresponding surface representations are expensive: Nielson's adaptation of the Coon's patch [11] for example (or [7], [1]) yields surface pieces of high rational degree and with (removable) singularities at the vertices. The exact surface normals are a challenge to compute. Polynomial tangent continuous constructions require several pieces per triangle, e.g. a Clough-Tocher-like split [2], [16],[15] since the tangents of one edge must not take information from the neighboring edge into account. (Both the original Powell-Sabin [14] and Clough-Tocher constructions do not apply in the setting of free-form surfaces.) Since tangent continuity does not imply overall better visual impression (smooth interpolating schemes often generate extraneous creases, see Figure 12), we settled for a least cost surface with one polynomial piece per triangle. Its geometric shape is not necessarily normal-continuous, but we improve the shading monotonicity by prescribing a separate, continuous normal. In [17] a more complex construction with quadratic pieces is used.

The *modeling range* of our surface representation should, for example, capture inflections implied by the triangle positional and normal data. This requires at least cubic geometry (quadratics do not have strict inflections) and quadratic normals. The choice of the center coefficient b_{111}, in particular, allows us to bulge the PN triangle. There is no additional data to suggest the use of higher degree patches and so we settled on the form of b and n given above. Of course there are many possible choices for the coefficients. The particular choice described in the next section has the merit of keeping the curved patch provably close to the flat triangle while interpolating its corner position and normals.

3 PN triangles

This section specifies and discusses the properties of the choice of PN triangle geometry and normals respectively. The geometry com-

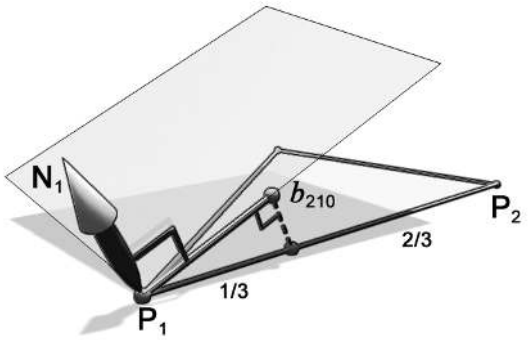

Figure 6: Construction of a tangent coefficient: project $(2P_1 + P_2)/3$ into the tangent plane at P_1.

ponent will influence, in particular, the object silhouette and the normal component influences the lighting calculation (vertex shading). The algorithm assumes an input in the form of vertex arrays, i.e. there is a unique normal for each vertex; however, there are situations when two triangulations are to be connected across sharp edges. The last subsection discusses this case.

3.1 Geometry coefficients of the PN triangle

We are given the positions $P_1, P_2, P_3 \in \mathbf{R}^3$ and normals $N_1, N_2, N_3 \in \mathbf{R}^3$ of the triangle corners as shown in Figure 5. Our choice of coefficients (control points) b_{ijk} is as follows.

0 Spread the b_{ijk} out uniformly over the flat triangle, i.e. place b_{ijk} in the intermediate position $(iP_1 + jP_2 + kP_3)/3$.

1 Leave the vertex coefficients in place so they match the corner positions.

2 For each corner, project the two tangent coefficients closest to the corner into the tangent plane defined by the normal at the corner. Here projection means: find the closest point on the plane to the point. This is illustrated by Figure 6. Recall that the projection of a point Q onto a plane with normal N attached to a point P is $Q' = Q - wN$ where $w = (Q-P)\cdot N$ and $\cdot$ denotes a the dot product.

3 Move the center coefficient from its intermediate position V to the average of all six tangent points and continue its motion in the same direction for $1/2$ the distance already traveled.

In formulas for implementation, coefficients or control points of the curved PN triangle are defined as follows:

$$
\begin{aligned}
b_{300} &= P_1, \\
b_{030} &= P_2, \\
b_{003} &= P_3, \\
w_{ij} &= (P_j - P_i)\cdot N_i \in \mathbf{R} \qquad \text{here `}\cdot\text{' is the scalar product,} \\
b_{210} &= (2P_1 + P_2 - w_{12}N_1)/3, \\
b_{120} &= (2P_2 + P_1 - w_{21}N_2)/3, \\
b_{021} &= (2P_2 + P_3 - w_{23}N_2)/3, \\
b_{012} &= (2P_3 + P_2 - w_{32}N_3)/3,
\end{aligned}
$$

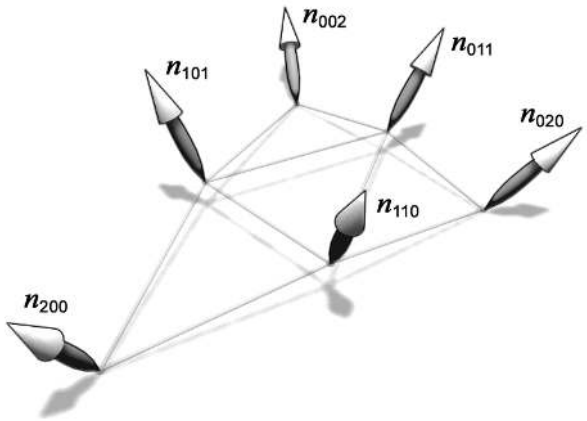

Figure 7: Coefficients of the normal component of a curved PN triangle.

$$
\begin{aligned}
b_{102} &= (2P_3 + P_1 - w_{31}N_3)/3, \\
b_{201} &= (2P_1 + P_3 - w_{13}N_1)/3, \\
E &= (b_{210} + b_{120} + b_{021} + b_{012} + b_{102} + b_{201})/6 \\
V &= (P_1 + P_2 + P_3)/3, \\
b_{111} &= E + (E - V)/2.
\end{aligned}
$$

3.2 Properties of the PN triangle geometry

While smoothing out contours, the curved PN triangle should not deviate too much from the original triangle to preserve the shape and avoid interference with other curved triangles. Evidently, the final geometry interpolates the vertices of the original flat triangles. Let L be the length of the longest triangle edge. The lemma below shows that each cubic boundary curve stays close to its edge, because its Bézier coefficients stay within a radius of $L/6$ of the edge. Therefore E is within a distance $L/6$ from the flat triangle and, since V lies on the triangle, b_{111}'s distance to the triangle is at most $L/4$. An improved estimate, since the coefficients are only approximate, yields $L/6$ the for the center and $L/8$ for the boundary.

Lemma 3.1 *The coefficient b_{210} lies on a circle of radius $\|P_1 - P_2\|/6$ about $P_1 + \frac{1}{6}(P_2 - P_1)$.*

Proof [Thales of Milet, 500 B.C.] The locus of all right triangle corners above a line of length $L/3$ is a semicircle of radius $L/6$. ⋈

Symmetry suggests a formula of the type $b_{111} = (1-\alpha)E + \alpha V$ for the central coefficient. The specific choice $\alpha = -1/2$ which results in $b_{111} = (1-\alpha)E + \alpha V = 3E/2 - V2 = E + (E-V)/2$ reproduces quadratic polynomials exactly as shown by Farin[5]. In other words, if the other nine coefficients were already chosen to represent a quadratic polynomial patch, then $\alpha = -1/2$ makes the PN triangle equal to the quadratic polynomial. Alternatively, b_{111} is the average of the three choices that each make one transversal tangent function linear.

If we assume that the triangles stem from a triangulation that has one normal associated with each vertex, then the boundary curves of abutting PN triangles are generated by the same algorithm, and

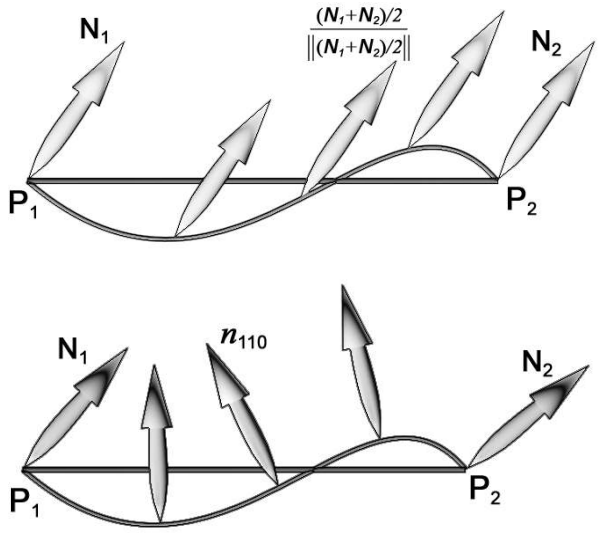

Figure 8: Linear interpolation of the normals at the endpoints (*top*) ignores inflections in the curve while the quadratic normal construction of curved PN triangles (*bottom*) picks up such shape variations.

the surface is continuous, i.e without cracks. PN triangles do not usually join with tangent continuity except at the corners. If they always did, we would have solved an open problem of approximation theory: whether the vertices of a triangulation can be interpolated by a C^1 function with one cubic piece per triangle.

3.3 Normal coefficients of the PN triangle

The normal to the geometry component of the PN triangles does not generally vary continuously from triangle to triangle (c.f. Figure 1 (c)). Therefore we define an independent linear or quadratic normal variation. The value of the normal at a parameter point u, v under *linear normal variation* is the normalized average of the values at the vertices:

$$N(u,v) = \frac{\mathbb{N}}{\|\mathbb{N}\|}, \qquad \mathbb{N} = (1-u-v)N_1 + uN_2 + vN_3.$$

Since the shading stage of the pipeline receives subtriangles with normals, say $N(u,v), N(u+h,v), N(u,v+h)$ for some steplength h, linear variation of the normal approximates Phong shading.

Linearly varying normals ignore inflections in the geometry as illustrated in Figure 8 (*top*). We therefore provide the option of *quadratic normal variation*. To capture inflections as in Figure 8 (*bottom*) a mid-edge coefficient for the quadratic map n is constructed following e.g. [12]: the average of the end-normals is reflected across the plane perpendicular to the edge as illustrated in Figure 9. Recall that the reflection A' of a vector A across a plane with (unnormalized) normal direction B is $A' = A - 2vB$ where $v = (B \cdot A)/(B \cdot B)$ and $\cdot$ denotes a the dot product. Figure 1 (d) illustrates this choice of normal.

In formulas for implementation, coefficients or control points of the normal component of a curved PN triangle are defined as follows for $\|N_i\| = 1$ (c.f. Figure 7):

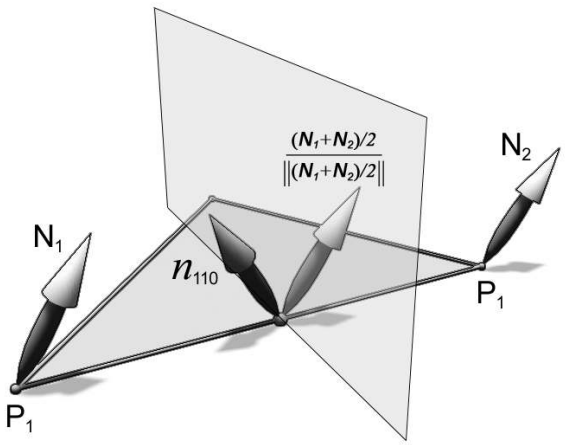

Figure 9: Construction of the mid-edge normal coefficient n_{110} for quadratically varying normals: the average of the end-normals is reflected across the plane perpendicular to the edge.

$$\begin{aligned}
n_{200} &= N_1, \\
n_{020} &= N_2, \\
n_{002} &= N_3, \\
v_{ij} &= 2\frac{(P_j - P_i)\cdot(N_i + N_j)}{(P_j - P_i)\cdot(P_j - P_i)} \in \mathbf{R} \\
n_{110} &= h_{110}/\|h_{110}\|, \quad h_{110} = N_1 + N_2 - v_{12}(P_2 - P_1), \\
n_{011} &= h_{011}/\|h_{011}\|, \quad h_{011} = N_2 + N_3 - v_{23}(P_3 - P_2), \\
n_{101} &= h_{101}/\|h_{101}\|, \quad h_{101} = N_3 + N_1 - v_{31}(P_1 - P_3).
\end{aligned}$$

The values of n are normalized before they are passed on to the vertex shading stage of the pipeline.

3.4 Curved Sharp Edges

So far we assumed that the data originated from a triangulation with one normal associated with each vertex. This implies that we smooth out all edges. To model sharp or crease edges, say a hemisphere capped by a disc, we would like to connect two triangulations by identifying vertices on their global boundary to form the crease. Evidently, the vertices on the crease now have two distinct normals and this must lead to cracks in the surface if only information based on one of the two flat triangles is available. Spline and generalized subdivision surfaces solve the problem by tagging the vertices, respectively searching the neighborhood and recognizing global boundary status. The argument below shows that with entirely local information, cracks can not be avoided.

Observation 3.1 (curved sharp creases) *If two patches share a vertex but not the corresponding normal, there is no strategy based only on the vertex and normal information (and respecting the vertex and normal information) local to one patch at a time to generate the same curve tangent for either patch independent of the normal information of the other.*

Proof The tangent of the curve is defined as the intersection of two planes, each defined by the vertex and one of the two normals. Since only one plane is known when constructing each patch a consistent choice is only possible by chance. ⋈

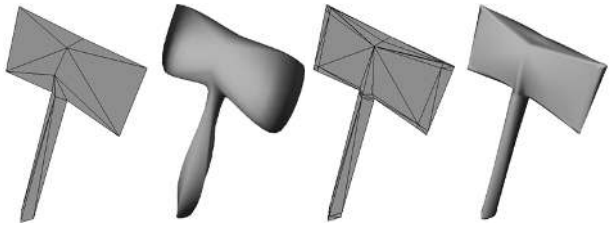

Figure 10: Sharpening a blunt axe (*left two*) by adding a seam of small triangles (*right*). Small triangles at the bottom also isolate the shaft from its end to keep it more cylindrical.

The observation implies that curved sharp creases with different normals associated with each side of the (logical) edge between two triangles will in general result in gaps in the surface.

An exception is the PN construction when the normals at two endpoints consistently imply a straight line segment as common boundary, e.g. when both normals are perpendicular to the edge. We are considering several options to also support curved sharp creases, e.g. increasing the number of indices in the input stream from three to (at most) nine for triangles with up to six crease half-edges. The additional indices point to the point-normal pair of the abutting triangle in the neighboring triangulation. This setup supports darts, i.e. edges that are sharp at one end and smooth at the other, as well as generic crease edges.

The algorithm for determining the PN triangle geometry is unchanged except that we now project onto a tangent line to obtain the tangent coefficient of an edge: let N^1 and N^2 be the two normals associated with the crease point P_1 along the edge $\overline{P_1P_2}$. Then b_{210} has to lie on the line $P_1 + t(N^1 \times N^2)$ where $\times$ is the cross product and b_{210} is the projection of $(2P_1 + P_2)/3$ onto that line:

$$b_{210} = P_1 + \frac{(P_2 - P_1) \cdot T}{3} T, \qquad T = N^1 \times N^2.$$

Currently, as illustrated in Figure 10, a software preprocessing step adds a rim of small triangles along edges intended to be sharp.

4 Hardware Performance

As Figure 2 illustrates, curved PN triangles are generated and tesselated in the pipeline stage inserted between the vertex-and-primitive-assembly stage and the vertex-shading stage. The vertex-and-primitive-assembly stage combines multiple streams of data such as 3-space position, 3-space normal, 3-space tangent vectors, texture coordinates and colors. Of these, the PN triangle construction only uses position and normal data. The other data are forwarded to the subtriangles typically by linearly interpolating the values at the vertices of the flat input triangle. Vector-valued attributes can also use quadratic interpolation. Because the input and output triangles of the curved PN triangle stage are of the same format, the impact on the design on adjacent pipeline stages is minimal.

To get an initial estimate of the impact of inserting the triangle-spawning PN triangle stage, we determined the number of vector operations for each output subtriangle vertex. The vector operations are dot products, additions of two vectors, scaling and per-component multiply of two vectors. The current implementation uses between 6.8 and 11.6 vector operations per generated vertex depending on the number of evaluations. `lod` 2 and 3 (see Figure 4) have the highest ratio of vector operations per vertex.

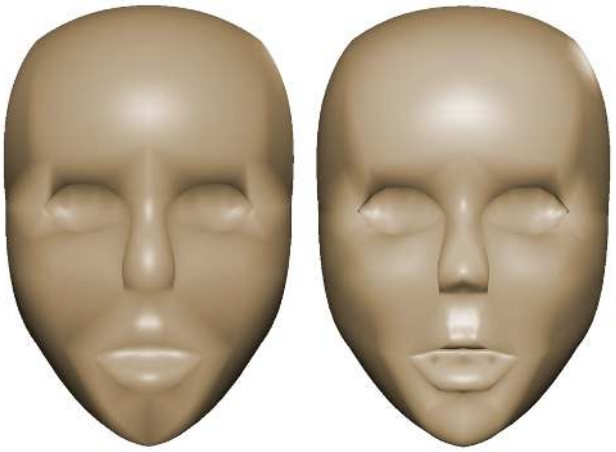

Figure 11: Mask of PN triangles with linearly varying normal (*left*) and quadratically varying normal (*right*).

To get a more detailed estimate of the impact, we need to consider fill rate and vertex shading. Fill rate is not a bottleneck since the screen area in pixels is unchanged. Vertex shading (transformation and lighting) and triangle set up, on the other hand, have to cope with a larger number of vertices. The overall rendering performance, in many applications, is, however, limited by the bandwidth needed to feed vertices to the graphics processor rather than the graphics processor cycles available for the geometry. Therefore, if the processor is sufficiently fast and the bus is busy, curved PN triangles render at the same speed as flat triangles.

Allowing a higher-order surface primitive, such as curved PN triangles, to be passed directly to hardware is a form of *geometry compression* that reduces bus traffic into the graphics chip. PN triangles, like other curved surface primitives, can act as a level of detail mechanism. Since the input to the PN triangle stage is the lowest level of detail, a flat triangle, its storage costs and bandwidth usage are minimal; and we can obtain higher levels of detail on the fly. If, instead, a developer were to store models for `lod` 0-4 (Figure 4) she would need approximately 70 times the amount of memory.

Key-frame interpolation is another instance where PN triangles substantially enhance performance. Given two models with identical topology, either the CPU or the graphics processor must generate interpolated values between the vertex positions of the two models. An application can reduce the number of vertices it has to store by applying the PN triangle algorithm after key-frame interpolation.

In summary, PN triangles can be viewed as a method for compressing geometry. Both bus bandwidth and memory usage can be substantially reduced by employing coarser input and PN triangles.

5 Visual Results

Figure 11 highlights the difference in visual appearance between curved PN triangles with linear and with quadratic normal. At times, the PN surface rendering is even preferable to the exact rendering of higher quality surfaces. Figure 12 places next to each other a 20-triangle cone refined with Butterfly interpolating subdivision [4] (which converges to an almost everywhere C^2 surface) and the geometrically non-smooth surface of curved PN triangles. Surfaces generated with modified Butterfly [18] and interpolatory $\sqrt{3}$ subdivision ([9], Figure 10) exhibit the same extraneous oscillations as Butterfly interpolating subdivision. Our experiments with

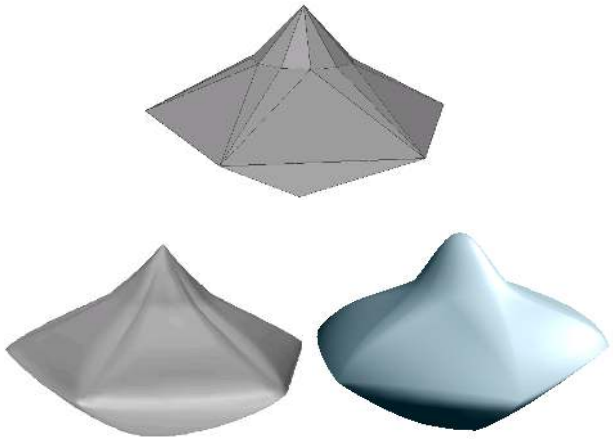

Figure 12: Input 'diamond model' (*top*), Butterfly interpolatory subdivision [4], (*left*), and the curved PN triangle surface (*right*). (Compare also with modified Butterfly and interpolatory $\sqrt{3}$ subdivision shown in Figure 10 of [9]).

characters from existing games demonstrate an appreciable increase in visual quality already for `lod` 1 or 2; the models rendered on the right in Figure 13 use `lod`=6.

6 Conclusions and Future Work

Curved PN triangles aim at providing a smoother silhouette and more organic shape for triangle-based models at a minimal cost for model preparation, application program modification, and rendering performance. Curved PN triangles can be viewed as a triangle multiplier in that each PN triangle takes on input a flat triangle and outputs many subtriangles in the same format. Sending a coarser triangulation to be rendered can therefore be viewed as a form of geometry compression that reduces bandwidth requirements and allows more geometry to be delivered to graphics memory, because the subtriangles ultimately rendered exist only on the chip. The underlying architecture is a first step towards supporting much neglected triangle-based surfaces at the hardware level.

Acknowledgments

We would like to thank our colleages at ATI Research, especially Steve Morein, Clay Taylor, Vineet Goel and Andy Gruber, as well as Charles Loop from Microsoft Research and Xiaobin Wu at the University of Florida for their input. The conference committee, especially Carlo Sequin, provided a variety of useful suggestions to help make this a stronger paper. We would also like to thank id Software for the models used in Figure 13, and Sam Howell of ATI Research provided some of the other models and illustrations.

References

[1] H. Chiyokura and F. Kimura. Design of solids with free-form surfaces. In *SIGGRAPH 83 Conference Proceedings*, volume 17(3), pages 289–298, July 1983.

[2] R. Clough and J. Tocher. Finite element stiffness matrices for analysis of plates in blending. In *Proceedings of Conference on Matrix Methods in Structural Analysis*, 1965.

[3] P. de Casteljau. Outillages méthodes calcul. Technical report, A. Citroen, Paris, 1959.

[4] Nira Dyn, David Levin, and John A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, April 1990.

[5] Gerald Farin. Smooth interpolation to scattered 3D data. In Robert E. Barnhill and Wolfgang Boehm, editors, *Surfaces in Computer-Aided Geometric Design*, pages 43–63. North-Holland, 1983.

[6] Gerald Farin. Triangular Bernstein–Bézier patches. *Computer Aided Geometric Design*, 3(2):83–127, 1986.

[7] J. A. Gregory. *Smooth interpolation without twist constraints*, pages 71–88. Academic Press, 1974.

[8] L. Kobbelt. $\sqrt{3}$ subdivision. *SIGGRAPH 2000 Conference Proceedings*, pages 103–112, 2000. New Orleans.

[9] U. Labsik and G. Greiner. Interpolatory sqrt(3)-subdivision. In M. Gross and F. R. A. Hopgood, editors, *Computer Graphics Forum (Eurographics 2000)*, volume 19(3), 2000.

[10] Charles T. Loop. Smooth subdivision surfaces based on triangles, 1987. Master's Thesis, Department of Mathematics, University of Utah.

[11] G. Nielson. *A transfinite, visually continuous, triangular interpolant*, pages 235–246. SIAM, 1987.

[12] C. W. A. M. Van Overveld and B. Wyvill. Phong normal interpolation revisited. *ACM Trans. on Graphics*, 16(4):397–419, October 1997.

[13] J. Peters. Smoothing polyhedra made easy. *ACM Transactions on Graphics*, 14(2):161–169, April 1995.

[14] M. J. D. Powell and M. A. Sabin. Piecewise quadratic approximations on triangles. *ACM Transactions on Mathematical Software*, 3(4):316–325, December 1977.

[15] L. A. Shirman and C. H. Séquin. Local surface interpolation with Bézier patches: Errata and improvements. *Computer Aided Geometric Design*, 8(3):217–222, August 1991.

[16] Leon A. Shirman and Carlo H. Sequin. Local surface interpolation with bezier patches. *Computer Aided Geometric Design*, 4(4):279–295, December 1987.

[17] K van Overveld and B. Wyvill. An algorithm for polygon subdivision based on vertex normals. 1997. http://www.win.tue.nl/cs/tt/kees/graphics.references.

[18] Denis Zorin, Peter Schroeder, and Wim Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *SIGGRAPH 96 Conference Proceedings*, pages 189–192, 1996.

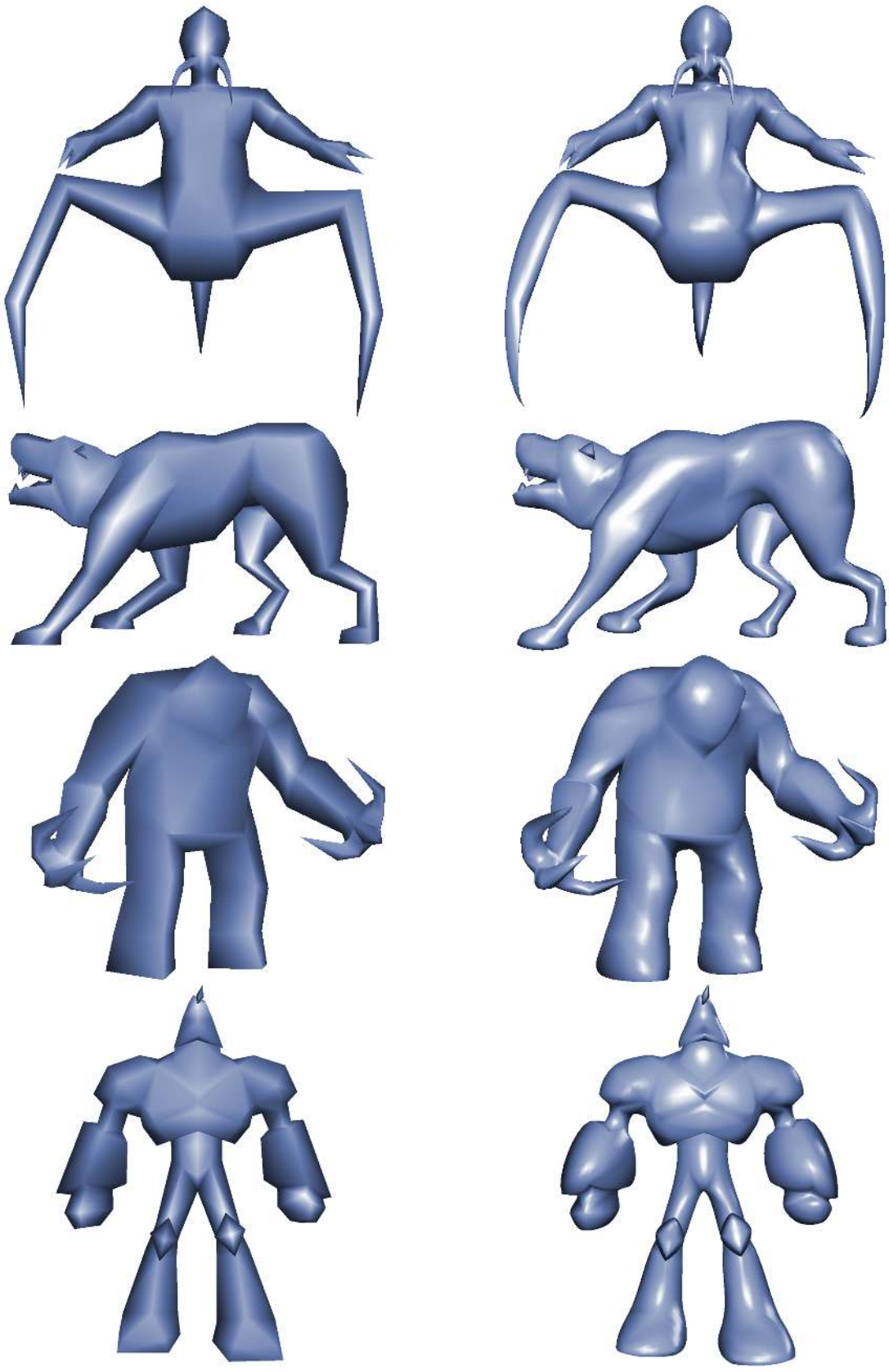

Figure 13: A family of game characters (*left*) and their PN triangle counterparts (*right*). The models were provided by id Software. None of these characters were authored with PN triangles in mind.