To appear in ACM Transactions on Graphics (SIGGRAPH Asia 2008 Proceedings)

Phong Tessellation

Tamy Boubekeur Marc Alexa
TU Berlin

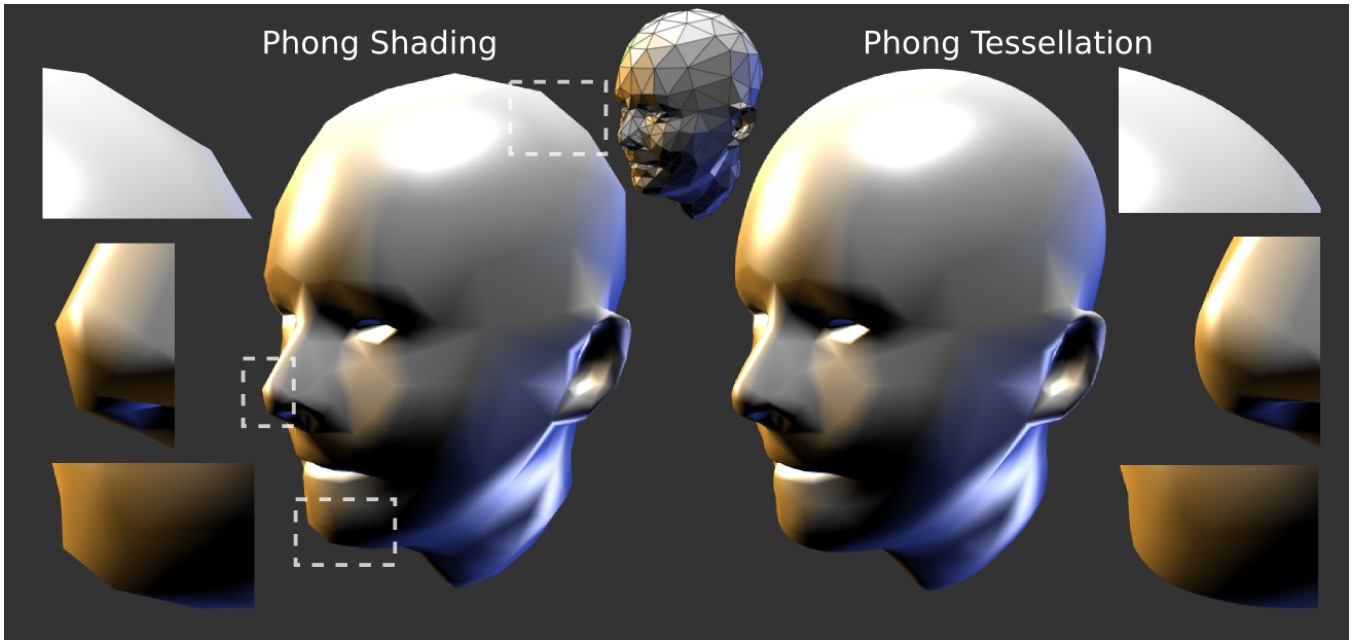


Figure 1: *Phong Tessellation completes Phong Shading.*

Abstract

Modern 3D engines used in real-time applications provide shading that hides the lack of higher order continuity inside the shapes using modulated normals, textures, and tone-mapping – artifacts remain only on interior contours and silhouettes if the surface geometry is not smooth. The basic idea in this paper is to apply a purely local refinement strategy that inflates the geometry enough to avoid these artifacts. Our technique is a geometric version of Phong normal interpolation, not applied on normals but on the vertex positions. We call this strategy Phong Tessellation.

CR Categories: I.3.3 [Computer Graphics]: Line and Curve Generation; I.3.3 [Picture/Image Generation]: Display Algorithms; I.3.1 [Hardware Architecture]: Graphics Processors;

Keywords: real-time tessellation, mesh refinement, visual continuity

1 Introduction

Interactive rendering progressively adopts effects inspired by techniques used in off-line rendering: HDR tone mapping, soft shadows, ambient occlusion, color bleeding, motion blur etc. now frequently appear in the interactive 3D engines. Since high quality smooth surfaces, such as NURBS and subdivision surfaces, are nowadays the standard geometric representation in off-line rendering, it appears natural to establish them also in real-time applications. However, the requirements of video games and other interactive applications are usually entirely visual, and a large part of the "visual smoothness" is already generated using Phong Shading and normal mapping. Thus, the remaining problem to solve is improving the geometry where visual smoothness cannot be generated based on shading: on contours and silhouettes.

Looking at Figures 1 and 2 for instance, using a high quality smooth surface representation for the faces would not make a significant difference in the interior part of the shaded images, as textures, Phong normal interpolation and lighting already provide a realistic visual complexity. However, the silhouettes suffer from the underlying polygonal representation. There are two competing goals when trying to improve the visual artifacts along contours:

- producing geometry along contours that is smooth so that the visual artifacts are avoided, while
- creating this geometry with as few operations as possible, as the area around the contour only accounts for a small part of the image and, thus, the overall visual impression, whose generation requires most of the CPU and GPU cycles.

We designed *Phong Tessellation* in this spirit, being barely more expensive than Phong Shading alone: it is based on using barycentric interpolation and orthogonal projections, and requires only the information a triangle naturally carries on the GPU, i.e. its vertex positions and vertex normals. We show that it is almost as efficient as the standard linear (flat) tessellation of triangles, while providing significantly improved contours in most cases. As shown in Fig. 1 it naturally complements Phong Shading.

Figure 2: *As shown in these captures from the game Doom 3 (id Software), normal mapping and Phong shading provide reasonable shading of the interior of objects, but the piecewise linear shape is clearly visible along contours.*

2 Previous work

Real-time tessellation has recently received a lot of interest in computer graphics. The reason is that applications usually maintain coarse approximations of 3D models for animation, interaction, physical simulation, while the GPU would be capable of rasterizing large amounts of primitives. Because the bus between CPU and GPU is still a limitation, it makes sense to send the coarse model to the GPU, where additional vertices and polygons are generated prior to projection and scan conversion.

Several techniques were developed to generate smooth surfaces on the GPU (NURBS [Guthe et al. 2005] or subdivision surfaces [Shiue et al. 2005]), however, they are considered too slow for the integration into real-time engines. Quite generally, generating smooth surfaces that depend on maintaining the topological structure of a mesh on the GPU require several rendering passes and are memory and computation intensive, which is incompatible with the requirements of realtime 3D engines.

One successful solution is derived from the seminal ideas of Gouraud [Gouraud 1971] and Phong [Phong 1975]: the *visual smoothness*, namely, realizing that in most situations exact *geometric smoothness* is not critical as long as the surface appears to be smooth as a result of the shading technique. Interpolating per-vertex normal vectors for shading computations achieves this visual smoothness, still avoiding knowledge of topological neighborhoods and processing polygons independently. Valchos et al. [Vlachos et al. 2001] built upon this principle: curved PN triangles are a purely local scheme, constructing a cubic Bézier patch according to the three positions and the three normals of the triangle. Following Van Overveld and Wyvill [van Overveld and Wyvill 1997], a quadratic patch is constructed to evaluate continuous normal fields meeting with C^0 continuity along edges and yielding a continuous shading. Boubekeur and Schlick [Boubekeur and Schlick 2007] use subdivision scheme [Loop 1987; Zorin et al. 1996] that steers a local quadratic approximation, leading to visually smoother surfaces than PN triangles, while keeping a high framerate. Most recently, Loop and Schaefer [Loop and Schaefer 2008] use low degree quad patches, again with separate normal fields, to approximate Catmull-Clark surfaces. The idea of a "visually smooth" geometric upsampling may be applied everywhere on the input mesh, or only on specific locations such as silhouettes. For instance, Dyken *et al.* [Dyken et al. 2008] use PN Triangles on silhouettes only.

All these techniques, as well as screen space methods [Max 1989], use the same basic principle: each input polygon is replaced by a polynomial patch. Our suggested operator for Phong tessellation avoids the explicit patching and is therefore simpler and more efficient while still being visually convincing in most cases.

3 Phong Tessellation

Consider a triangle $\mathbf{t}$ indexing three vertices $\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$, with $\mathbf{v} = \{\mathbf{p}, \mathbf{n}\}$, $\mathbf{p} \in \mathbb{R}^3$ the position and $\mathbf{n} \in \mathbb{R}^3$ the normal vector. Generally, the tessellation of $\mathbf{t}$ generates a set of new vertices lying on a surface defined over $\mathbf{t}$. Linear tessellation simply generates vertices on the plane defined by $\mathbf{t}$. Each generated vertex $\mathbf{p}(u,v)$ in the linear tessellation corresponds to a barycentric coordinate $(u,v,w), u,v \in [0,1], w = 1-u-v$ defined by

$$\mathbf{p}(u,v) = (u,v,w)(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k)^{\mathsf{T}} \tag{1}$$

Phong normal interpolation is using the same process, only normalizing the result in the end:

$$\mathbf{n}'(u,v) = (u,v,w)(\mathbf{n}_i, \mathbf{n}_j, \mathbf{n}_k)^{\mathsf{T}}, \quad \mathbf{n}(u,v) = \mathbf{n}'/\|\mathbf{n}'\| \tag{2}$$

Clearly, Phong normals are successful because they are as simple as possible. They have numerous well known defects (e.g. they cannot provide inflection points), but still they are the preferred choice because the few cases in which they produce visible artifacts are far outweighed by their computational simplicity in comparison to higher order variants. We believe that a real-time mesh refinement operator should be as efficient and as simple as Phong normal interpolation to be similarly attractive in interactive applications. The procedure we motivate and describe in the following appears to be the simplest that still provides curved geometry.

Note that around each vertex the tangent plane defined by the vertex normal is the appropriate local geometry. So, conceptually we project the triangle $\mathbf{t}$ onto the tangent plane of vertex $\mathbf{v}_i$ and perform barycentric interpolation within the tangent plane to define the geometry in the vicinity of $\mathbf{v}_i$. The geometry relative to $\mathbf{v}_j$ and $\mathbf{v}_k$ is similarly defined. The geometry defined over the whole triangle $\mathbf{t}$ is then generated by barycentric interpolation of the three barycentric interpolations within the projected triangles. Since projection commutes with barycentric interpolation, the computation of points can be simplified to the following procedure:

1. compute the linear tessellation, then
2. project the resulting point orthogonally onto the three tangent planes defined by the triangle vertices, and finally
3. compute the barycentric interpolation of these three projections.

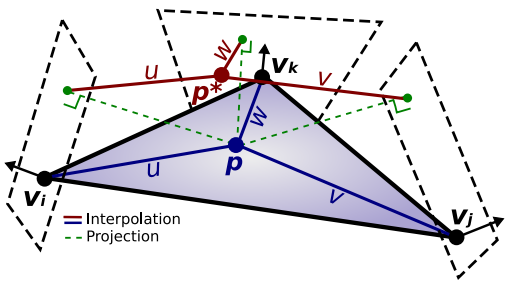


Figure 3: *Phong Tessellation principle. Instead of interpolating normals as in Phong Shading, we interpolate projection onto vertices tangent plane to define a curve geometry for each triangle.*

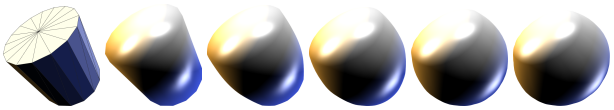


Figure 4: *Coarse mesh, followed by various Phong Tessellations with α equal to 0, 1/4, 1/2, 3/4 and 1.*

Let

$$\pi_i(\mathbf{q}) = \mathbf{q} - \left((\mathbf{q} - \mathbf{p}_i)^{\mathsf{T}} \mathbf{n}_i\right) \mathbf{n}_i$$

be the orthogonal projection of $\mathbf{q}$ onto the plane defined by $\mathbf{p}_i$ and $\mathbf{n}_i$, then Phong Tessellation is defined as:

$$\mathbf{p}^*(u,v) = (u,v,w) \begin{pmatrix} \pi_i(\mathbf{p}(u,v)) \\ \pi_j(\mathbf{p}(u,v)) \\ \pi_k(\mathbf{p}(u,v)) \end{pmatrix} \tag{3}$$

Comparing linear tessellation (Eq. 1), this operator adds three orthogonal projections and an additional barycentric interpolation.

A shape factor α can be used to interpolate between linear (flat) and Phong Tessellation, controlling the amount of curvature produced over a triangle. In fact, we use the following definition of inserted vertices at (u,v) for Phong tessellation:

$$\mathbf{p}^*{}_\alpha(u,v) = (1-\alpha)\mathbf{p}(u,v) + \alpha(u,v,w) \begin{pmatrix} \pi_i(\mathbf{p}(u,v)) \\ \pi_j(\mathbf{p}(u,v)) \\ \pi_k(\mathbf{p}(u,v)) \end{pmatrix} \tag{4}$$

Fig. 4 shows the influence of this shape parameter on the final geometry. In our experiments, we fix $\alpha = 3/4$ globally as this value provides convincing results in most of the situations. Nevertheless, α can also be set on a per-vertex basis and interpolated over triangles.

4 Properties

The important computational properties of Phong Tessellation are that (1) only the information associated to a triangle on the GPU is used and (2) the evaluation has no side effects: note that all other techniques for evaluating patches on the GPU we are aware of [Vlachos et al. 2001; Boubekeur and Schlick 2007; Loop and Schaefer 2008] require either two rendering passes (one for setting up the patch by writing the necessary patch information into the memory of the GPU and another one for evaluating this information) or a full patch construction for each refined vertex. Our construction can be evaluated in a single pass, because vertex positions and normals are part of the information for triangles.

The geometry generated by our definition is a quadratic patch, as can be seen immediately by writing out the definition:

$$\begin{aligned}\mathbf{p}^*(u,v) =& u^2\mathbf{p}_i + v^2\mathbf{p}_j + w^2\mathbf{p}_k + uv\left(\pi_i(\mathbf{p}_j) + \pi_j(\mathbf{p}_i)\right) + \\ & vw\left(\pi_j(\mathbf{p}_k) + \pi_k(\mathbf{p}_j)\right) + wu\left(\pi_k(\mathbf{p}_i) + \pi_i(\mathbf{p}_k)\right)\end{aligned}$$

The quadratic patch directly shows that the surface will be curved, as desired, as long as the vertex normals are different from each other (see Fig. 5). For identical normals the surface is identical to the flat triangle, similar to Phong normals, which are constant for constant input. Note that a quadratic patch cannot provide an inflection point. We feel a higher order patch is not warranted in practical applications, again similar to higher order normal interpolation widely considered unnecessary in practice.

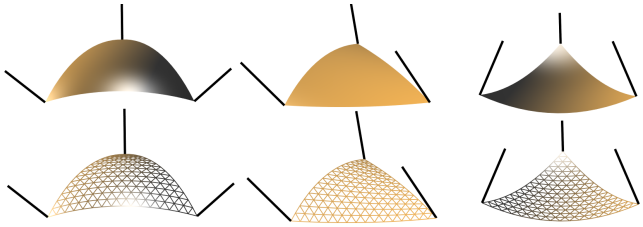


Figure 5: *Convex, saddle and concave normals configuration.*

While the patches are curved, the collection of quadratic patches forms a C^0 continuous surface: along each edge, barycentric interpolation leads to using only the positions and normals from the vertices incident on the edge, and that information is identical for two neighboring triangles. Across edges or vertices the tangents are not continuous. However, combining Phong Tessellation with Phong normal interpolation for shading results in *visual smoothness* where Phong normals provide smoothly varying shading across mesh edges and Phong vertices form curved silhouettes and contours (see Fig. 6).

Phong Tessellation interpolates the input mesh vertices and does not exhibit the shrinking effects of approximating refinement schemes (e.g. subdivision). This allows reusing all existing meshes present in real-time applications, without redesigning them so that their limit surface reaches the desired shape. The visual results compared to other techniques that interpolate the mesh vertices such as PN triangles or modified Butterfly subdivision are equally convincing (see Fig. 9), while being significantly more efficient. Of course, Phong tessellation, such as curved PN triangles, might generate contours with visible tangent discontinuities, while subdivision surface will always provide at least C^1 contour curves.

The quality of the curved geometry defined by Phong tessellation depends on the normals supplied with vertices. In our tests, we use a classical angle-weighted combination of incident faces normals. We observe similarly good but slightly different results using other weights. Note also that singularities such as sharp edges could be controlled using scalar tags [Boubekeur and Schlick 2005b].

5 Performance

The generation of a single vertex in the tessellation represents only few lines of shader code. Note that the computation is based on barycentric interpolation and three orthogonal projections while a straightforward evaluation of the quadratic form would either need more operations or two rendering passes. We have observed our variant to be 40% faster on average.

Since tessellator units are still in development (announced for DX 11), we emulate them using either the geometry shader unit (at low tessellation rates) or a generic GPU refinement kernel in the spirit of Boubekeur and Schlick [Boubekeur and Schlick 2005a]. In

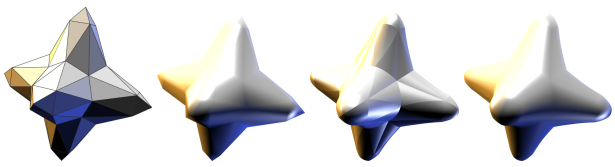
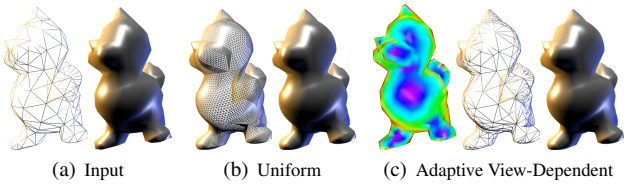


Figure 6: *Phong rendering. From left to right: the mesh, the standard Phong Shading, Phong Tessellation with analytic normals and the Phong Tessellation with Phong normals.*



(a) Input (b) Uniform (c) Adaptive View-Dependent

Figure 7: *Adaptive view-dependent Phong Tessellation for real-time geometric upsampling on silhouettes and contours only.*

each case, we perform adaptive mesh refinement [Boubekeur and Schlick 2008] by supplying an additional *depth* value with each vertex corresponding to the desired tessellation ratio in its vicinity.

As shown on Fig. 7, the main influence of Phong tessellation appears on contours and silhouettes, so we perform an adaptive refinement in these locations, considering the angle between the vertex normal and the view vector and relying on Phong Shading to produce the necessary visual smoothness inside the shape.

In our experiments, we use a simple measure to compute a refinement depth smoothly growing around contours:

$$d_i = \left(1 - \left\| \mathbf{n}_i^{\mathsf{T}} \frac{\mathbf{c} - \mathbf{p}_i}{||\mathbf{c} - \mathbf{p}_i||} \right\| \right) m$$

with $\mathbf{c}$ the position of the camera and m the maximum refinement depth. This captures silhouettes and interior contours, and addressing them with geometry synthesis makes our solution far easier and more efficient than clipping techniques [Sander et al. 2000]. To avoid strong popping artifacts, a progressive transition between coarse and refined polygons can be optionally performed by mapping the refinement depth d_i onto α.

Fig. 8 shows the framerate of Phong Tessellation compared to linear tessellation (producing flat geometry on triangles). We have measured the framerate on a GeForce 8800 GTX, 768 Mb, in practical rendering conditions. We observe an overhead of about 10% when using Phong Tessellation in the geometry processing workload (shading disabled). However, in a realistic rendering environment – with shading enabled, full image synthesis at 1600x1200, 3 light sources and dynamic deformations of the geometry (yellow and green bars in Fig. 8) – this cost becomes proportionally negligible. The different pictures presented in this paper illustrate the visual improvement.

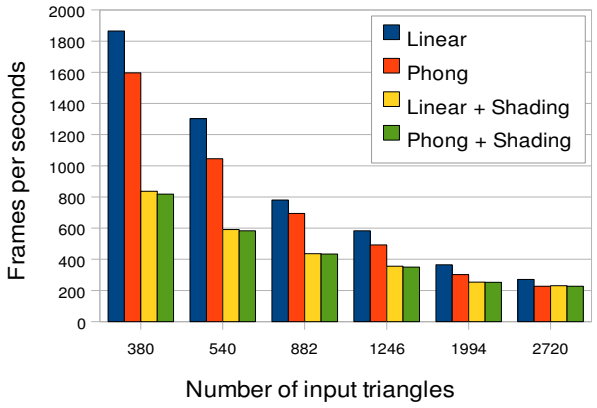


Figure 8: *Framerate comparison between linear (flat) and Phong tessellation, with and without shading (max depth 64x64).*

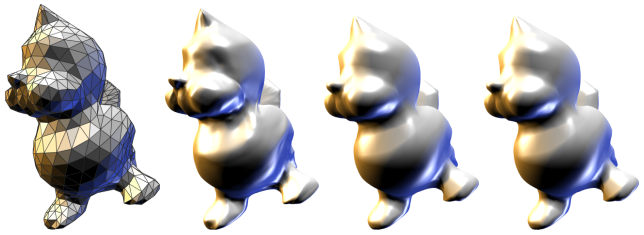


Figure 9: *Comparison: Modified Butterfly subdivision (middle-left), PN Triangles (middle-right) and Phong tessellation (right).*

6 Conclusion

Normal mapping and all the current shading techniques already provide light reflections appearing to come form curved geometry inside the shapes – but still exhibit the piecewise linear geometry on silhouettes and contours. We believe that in many realtime scenarios actually generating smooth surfaces on either the CPU or approximations of them on the GPU is wasteful. Instead, the simple procedure of *Phong Tessellation*, which requires only orthogonal projections and linear interpolations based on vertex positions and normals, can be used to generate curved contours. Combined with Phong Shading to generate continuously varying shading, it provides enough visual smoothness for convincing renderings in most situations – and that with minimal extra cost.

Phong tessellation can be implemented transparantly into the rendering pipeline, i.e. users need not adapt their rendering engine to use it. It automatically works with any geometry based on meshes, including the case of dynamically updated vertex positions or normals. It can be easily used into the upcoming tesselator units.

References

BOUBEKEUR, T., AND SCHLICK, C. 2005. Generic mesh refinement on GPU. In *Proceedings of ACM SIGGRAPH/Eurographics Graphics Hardware*, 99–104.

BOUBEKEUR, T., AND SCHLICK, C. 2005. Scalar tagged PN triangles. In *Proceedings of Eurographics 2005 (short papers)*, 17–20.

BOUBEKEUR, T., AND SCHLICK, C. 2007. QAS: Real-time quadratic approximation of subdivision surfaces. In *Proceedings of Pacific Graphics 2007*, 453–456.

BOUBEKEUR, T., AND SCHLICK, C. 2008. A flexible kernel for adaptive mesh refinement on GPU. *Computer Graphics Forum 27*, 1, 102–114.

DYKEN, C., REIMERS, M., AND SELAND, J. 2008. Real-time GPU silhouette refinement using adaptively blended bezier patches. *Comp. Graph. Forum 27*, 1, 1–12.

GOURAUD, H. 1971. Continuous shading of curved surfaces. *IEEE Transactions on Computers 20*, 6, 623–628.

GUTHE, M., BALZS, ., AND KLEIN, R. 2005. GPU-based trimming and tessellation of nurbs and t-spline surfaces. In *Proceedings of ACM SIGGRAPH*, 1016–1023.

LOOP, C., AND SCHAEFER, S. 2008. Approximating catmull-clark subdivision surfaces with bicubic patches. *ACM Transaction on Graphics 27*, 1, 1–8.

LOOP, C. 1987. *Smooth subdivisions surfaces based on triangles*. Master's thesis, University of Utah.

MAX, N. 1989. Smooth appearance for polygonal surfaces. *The Visual Computer 5*, 3 (Mai), 160–173.

PHONG, B. T. 1975. Illumination for computer generated pictures. *Comm. ACM 18*, 6 (June), 311–317.

SANDER, P., GU, X., GORTLER, S., HOPPE, H., AND SNYDER, J. 2000. Silhouette clipping. In *Proceedings of ACM SIGGRAPH*, 327–334.

SHIUE, L.-J., JONES, I., AND PETERS, J. 2005. A realtime GPU subdivision kernel. In *Proceedings of ACM SIGGRAPH*, 1010–1015.

VAN OVERVELD, C. W. A. M., AND WYVILL, B. 1997. Phong normal interpolation revisited. *ACM Transaction on Graphics 16*, 4, 397–419.

VLACHOS, A., PETERS, J., BOYD, C., AND MITCHELL, J. 2001. Curved PN triangles. *Proceedings of ACM Symposium on Interactive 3D*, 159–166.

ZORIN, D., SCHROEDER, P., AND SWELDENS, W. 1996. Interpolating subdivision for meshes with arbitrary topology. In *Proceedings of ACM SIGGRAPH*, 189–192.