QAS: Real-time Quadratic Approximation of Subdivision Surfaces

Tamy Boubekeur Christophe Schlick
LaBRI - INRIA - University of Bordeaux

Abstract

We introduce QAS, an efficient quadratic approximation of subdivision surfaces which offers a very close appearance compared to the true subdivision surface but avoids recursion, providing at least one order of magnitude faster rendering. QAS uses enriched polygons, equipped with edge vertices, and replaces them on-the-fly with low degree polynomials for interpolating positions and normals. By systematically projecting the vertices of the input coarse mesh at their limit position on the subdivision surface, the visual quality of the approximation is good enough for imposing only a single subdivision step, followed by our patch fitting, which allows real-time performances for million polygons output. Additionally, the parametric nature of the approximation offers an efficient adaptive sampling for rendering and displacement mapping. Last, the hexagonal support associated to each coarse triangle is adapted to geometry processors.

1 Introduction

Subdivision surfaces are undoubtedly the most flexible smooth geometric representation. By only manipulating a carefully designed low-resolution mesh, an high-resolution smooth version is automatically generated using a set of local recursive rules applied on each input coarse polygon. However, while being intensively used in CAD and SFX industries, they have not yet gained a significant interest for interactive and real-time applications. In fact, their recursive definition imposes a non-trivial CPU overhead, difficult to hide in interactive applications.

A *subdivision scheme* [13] defines a smooth surface using a coarse mesh M^0 and a subdivision operator S, that combines various refinement rules (odd vertex, even vertex, border, crease, etc). For most subdivision schemes such as Loop [8] or Catmull-Clark [6], these rules are local, and only require the one-ring-neighborhood for subdividing each coarse polygon, quickly converging toward the limit surface. Thus, the application of the refinement rules is done recursively, generating a set of meshes $\{M^0, M^1, ..., M^n\}$ with $M^{k+1} = S(M^k)$ until the chosen depth n. The weighted combination of neighboring vertices for computing the next position of a vertex is usually illustrated with a subdivision mask. For stationary schemes, limit masks exist that directly provide the projection of a vertex on the limit surface.

With the increasing demand in realism for interactive applications, efficient rendering of subdivision surfaces has become a major research area in recent years. However, the lack of geometry generation on GPU, as well as the reduced knowledge about local neighborhood allowed in the graphics hardware pipeline, have led researchers to tackle efficient rendering of subdivision surfaces with two different approaches: *precomputed tables of basis functions* [9] which can be implemented on GPU [2], and *images-based methods* [10, 5] which convert the input coarse mesh into a set of images and recursively apply scaling and filtering operations (multi-pass rendering). Unfortunately, the former approach has a clear lack of flexibility and does not address the problem of geometry generation, while the latter is slow in general, yielding interactive performances only for models composed of a few hundreds of coarse polygons. Alternatively to true subdivision schemes, *visually smooth* refinement can be tailored efficiently by using triangular Bézier patches locally generated on triangles [12] but their empirical generation only provides poor visual quality.

Figure 1: . **Left:** *Coarse mesh (546 triangles).* **Middle** *Our real-time GPU approximation of the subdivision surface (527 FPS - depth 5 - 500k triangles).* **Right:** *True Loop subdivision performed on CPU at same depth.*

2 Approximated Subdivision

The method we propose in this paper uses limit projections for driving a local polynomial approximation of the surface, which allows a direct evaluation at arbitrary location without recursion, in the spirit of the work done by Stam [11], but efficient enough to be done in real-time. In fact, by considering both positions and normals, we produce a visually smooth rendering adapted to interactive applications using simple quadratic Bézier patches, directly at vertex/geometry shader level [1], without any mesh-to-image conversion. We call our technique "QAS" for *Quadratic Approximation of Subdivision Surfaces*.

2.1 Principle

The very first subdivision step provides a crucial information on the target smooth surface: it indicates in which di-

0-7695-3009-5/07 $25.00 © 2007 IEEE
DOI 10.1109/PG.2007.20

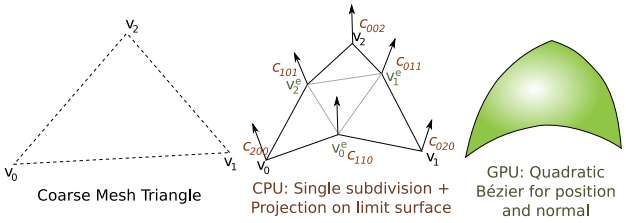


Figure 2: *QAS principle.* **Left:** *Coarse triangle T of M^0.* **Middle:** *Enriched hexagon H^T sampled on M^∞.* **Right:** *Final smooth patches $\{P_T, N_T\}$ generated on GPU.*

rection the surface will converge for all its edges. By studying the different subdivision schemes developed over the years, we can observe that the variation they produce on edges is a good indicator of their smoothness and curvature quality. This information is even more accurate with limits masks (i.e. when projecting each vertex at its limit position). We propose to use this initial guess of the first subdivision step, performed on the CPU in our implementation, to compute a local quadratic Bézier approximation on the GPU. Instead of using an empirical estimation of the Bézier coefficients for producing the *visual continuity* as done in [12], we fit two Bézier patches on the limit positions (resp. normals) provided by the single subdivision step with projection on the limit surface M^∞ (see Figure 2). With these two patches in hand, we can sample *adaptively* the piece of subdivision surface belonging to each input coarse triangles using either the *vertex shader* [3, 4] or the *geometry shader* [1].

2.2 Single Projection

The algorithm starts by applying a single subdivision step using limit masks. Each triangle T is thus split into 4 sub-triangles, with vertices on the limit surface. These sub-triangles share 6 vertices (Figure 2) and the submesh can thus be organized in an hexagonal shape $H_T = \{v_0, v_1, v_2, v_0^e, v_1^e, v_2^e\}$ with $v_i = \{p_i, n_i\}$ being the limit positions and normals (using tangent masks for instance) at this location. This structure is adapted to recent graphics hardware including a geometry shader stage, which enables to transmit triangles with edge neighbors: here we transmit edge vertices inserted by the subdivision pass instead. Note that we focus on triangle meshes, since they are ubiquitous in interactive applications. Thus, we use the Loop scheme [8] as a basis of QAS. The Modified Butterfly scheme [14] can be used when the interpolation of the coarse mesh is mandatory. We perform this step on CPU in our implementation. However, a GPU implementation can be considered.

2.3 GPU Polynomial Approximation

Once H_T is transmitted to the GPU, a shader (either vertex shader on old devices or geometry shader on recent ones) automatically fits 2 *triangular Bézier patches* to H_T: $P_T(u,v)$ for positions and $N_T(u,v)$ for normals. In other words, we produce a *procedural displacement map* and a

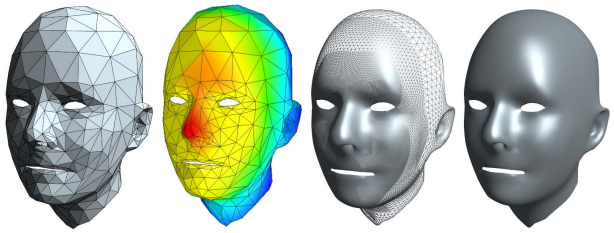


Figure 3: *Adaptive Subdivision Rendering.* **Left:** *Input coarse mesh (703 triangles).* **Mid. Left:** *View-dependent adaptive depth tag.* **Mid. Right:** *GPU Adaptive tessellation (620k tri. at 499 FPS)* **Right:** *Final QAS rendering.*

procedural normal map that approximate the variation of the limit subdivision surface. Both patches are defined by:

$$Q(u,v,w) = \sum_{i+j+k=2} b^2_{ijk}(u,v,w)c_{ijk}$$

with

$$b^2_{ijk} = \frac{2!}{i!j!k!}u^i v^j w^i \quad \text{and} \quad w = 1-u-v$$

In practice, c_{ijk} is replaced by p_{ijk} or n_{ijk} (see Figure 2). We use quadratic patches as they provide a good trade-off between curvature reproduction and computational cost.

Now, we have to define the 6 control points required by both Bézier patches, such as they interpolate the limit vertices (either positions or normals). These control points are organized as an hexagon (middle part of Figure 2): three of them correspond to the original vertices $\{v_0, v_1, v_2\}$ projected at the limit and are naturally interpolated by the triangular Bézier patches at control points $\{c_{200}, c_{020}, c_{002}\}$, while $\{c_{110}, c_{011}, c_{101}\}$ correspond to edge vertices $\{v_0^e, v_1^e, v_2^e\}$ and are **not** interpolated. So, we need to define them such as the actual geometry defined by P_T (resp. N_T) interpolates the edge positions (resp. normals). Actually, a linear collocation is possible in this case. For instance, considering the first edge vertex p_0^e, we have to solve:

$$P\left(\frac{1}{2},\frac{1}{2},0\right) = \frac{1}{4}(p_0 + p_1 + 2p_{110}) = p_0^e$$

which implies that

$$p_{110} = \frac{1}{2}(4p_0^e - p_0 - p_1)$$

Other edge coefficients are obtained in a similar fashion, and the same principle is used for computing the Bézier patch for normals. Separating the position field and normal field defined for each patch allows a local computation of the approximation (on a per-hexagon basis), without dealing with high order cross-edge continuity [12]. By interpolation, the normal field defined by N_T is guaranteed to be C^0 on edges, which produces a visually smooth shading.

2.4 Adaptive Rendering

By substituting recursive rules with Bézier patches, we can directly evaluate the surface approximation at arbitrary parameter values. So not only uniform tessellation is done

without recursion, but adaptive refinement is also made easier. This adaptivity can be performed by setting a per-vertex subdivision *depth tag*, either on CPU or GPU, using for instance a view-dependent metric (e.g. coarse triangle to camera distance) or a view-independent one (e.g. curvature approximation). While future *tessellator units* will enable direct QAS rendering, adaptive tessellation can be obtained on today's hardware with either two implementations:

- **Geometry Shader**: H_T can be directly transmitted to the GS using the DX10 pipeline [1]. A simple loop evaluates points and normals using P_T and N_T and outputs a stream of triangles. Unfortunately, this solution only holds for low subdivision depth, as the size of the GS output is hardware limited.
- **Vertex Shader**: For higher subdivision depth (3 and more), the *adaptive refinement kernel* of [4] is more efficient: H_T is transmitted to the vertex shader using uniform variables, and a pretessellated patch, called *Adaptive Refinement Pattern* (ARP), choosen according to the depth tags of T, is drawn. The ARP directly evaluates P_T and N_T at the pretessellated nodes using *barycentric interpolation* [3]. Note that the transfer cost of H_T is not a bottleneck for deep levels.

Figure 3 gives an example of an adaptive on-the-fly rendering of QAS. In the following listing, we provide a generic GPU implementation of QAS in GLSL for on-the-fly Bézier patch fitting and adaptive sampling, running on any GPU equipped with vertex shading capabilities:

```
const uniform vec3 n0, n1, n2, p0, p1, p2;
const uniform vec3 ne0, ne1, ne2, pe0, pe1, pe2;
vec3 edgeCP (vec3 e, vec3 p0, vec3 p1) {
  return (e * 4.0 - p0 - p1) * 0.5;
}
vec3 Q (float u, float v, float w,
        vec3 p0, vec3 p1, vec3 p2, vec3 e0, vec3 e1, vec3 e2) {
  vec3 n200 = p0, n020 = p1, n002 = p2;
  vec3 n110 = edgeCP (e0, p0, p1);
  vec3 n101 = edgeCP (e2, p0, p2);
  vec3 n011 = edgeCP (e1, p1, p2);
  return w * (n200*w + n110*2*u) +
         u * (n020*u + n011*2*v) +
         v * (n002*v + n101*2*w);
}
vec3 P (float u, float v, float w) {
  return Q (u, v, w, p0, p1, p2, pe0, pe2, pe2);
}
vec3 N (float u, float v, float w) {
  return Q (u, v, w, n0, n1, n2, ne0, ne2, ne2);
}
void main(void) {
  float u = gl_Vertex.x; // barycentric coordinates
  float v = gl_Vertex.y; // as position in the
  float w = 1.0 - u - v; // RP drawn
  gl_Vertex.xyz = P (u, v, w);
  gl_Normal = normalize (N (u, v, w));
  [...] // Shading
}
```

3 Results

We have implemented QAS on an AMD Athlon 3500, with 2GB of memory and an nVidia Geforce 8800 GTX, using C++, OpenGL and GLSL. While being geometrically only C^0, the resulting surface has an appearance almost indistinguishable from the equivalent true subdivision surface (see Figure 1). This is due to the combined fitting of positions and normals, which ensures both a smooth shading and curved silhouettes. Considering performances, QAS outperforms existing solutions [2, 10] for three reasons: we only perform a single true subdivision pass on CPU, we use a single rendering pass on GPU whatever the depth (i.e., constant processing cost per-vertex) and there is no geometry-to-texture conversion. Note also that the mesh is always synthesized on-the-fly, either using Geometry Shaders or Refinement Patterns, without storing the topology of the high resolution mesh. As a result we obtain real-time performance (more than 120 FPS) for objects composed of several thousands of coarse polygons, subdivided at depth 5 (more than $2.5M$ tessellated triangles). Performances degrade linearly with the number of triangles created and transmitted at CPU level. As a limitation, note that the higher is the vertex valence, the less accurate becomes QAS. However, this can be prevented by performing remeshing. Last, the direct adaptive rendering allowed by QAS, combined with its low CPU overhead makes this approximation particularly suitable for high quality interactive applications, as it offers much better results than purely empirical smoothing methods. Figure 5 gives additional examples of QAS rendering: we can observe that high framerates can be reached even with deep refinement levels, since our pure parametric evaluation does not access texture memory.

Property	Shiue's kernel	QAS
CPU Preprocess	2 passes (1x16)	1 pass (1x4)
GPU Input process	2-ring unfold	none
Rendering pass	depth $\times$ num. of tri.	1
GPU Workload	FS	VS/GS
Reproduction	Exact	Approximate
Adaptivity	Difficult	Trivial
Polygons Type	All	Pre-tessellated
Schemes	All	Dyadic
FPS (4k, depth 5)	Interactive	Real-Time

Table 1: *Comparison of QAS with Shiue's kernel [10] for the subdivision of a dynamic mesh.*

Comparison: We compare QAS to the GPU kernel of Shiue et al. [10] as it is one of the best solutions so far. Table 1 states advantages and weaknesses of our approach compared to their. One interesting property of QAS is its single pass vertex/geometry shading principle: thus, recent graphics hardware with unified architecture will automatically allocate additional shader units for vertex shading to obtain optimal balance between vertex and fragment processing, avoiding the usual conversion required by fragment-based processing of geometry. The local nature of QAS makes it also easily comparable to Curved PN Triangles [12]. Formally, the two approaches differ in the computation of Bézier control points: an empirical estimation based on tangent plane for PN Triangles, and a true limit subdivision surface interpolation in our case. As a result, we obtain

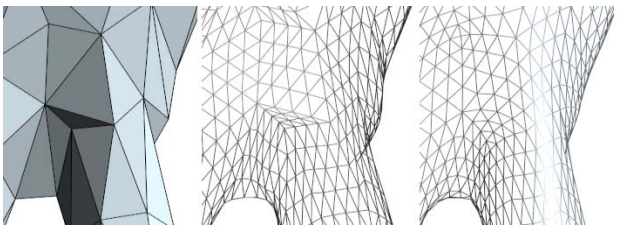


Figure 4: *Comparison with Curved PN Triangle Smoothing.* **Left:** *Coarse Mesh.* **Middle:** *Curved PN Triangles (cubic patches).* **Right:** *QAS refinement.*

a far better quality since limit projection may create larger, smoother and more consistent variation of the geometry that the simple normal-based approach (see Figure 4). This also allows us to simply use a quadratic polynomial instead of a cubic one.

4 Conclusion

We have proposed a simple and visually convincing approximation of subdivision surfaces by combining a single limit subdivision pass on CPU with a quadratic Bézier patch fitting on GPU. QAS is easy to implement, avoids recursion and reaches real-time performances for several thousands of input coarse polygons per-frame, outputting millions of tessellated triangles. Our method is generic in the sense that arbitrary depth and arbitrary vertex valence can be handled and adaptively subdivided. This approximation imposes less CPU workload, less graphics bus bandwidth and is more efficient than exact GPU subdivision kernels, while providing better visual results than empirical smoothing [12] and lower memory footprint than table-based methods. While CAD applications may benefit from more precise and more costly approximation techniques such as the recent work of Loop and Schaefer [7], QAS represents a solid choice for interactive applications, such as video games and virtual reality software, and can also be considered for special effects, as a large upsampling can be done adaptively on-the-fly for high resolution displacement mapping. As future work, we plane to perform the limit projection at GPU level still preserving a single pass rendering.

References

[1] D. Blythe. The direct3d 10 system. *ACM Siggraph*, 2006.

[2] J. Bolz and P. Schroder. Evaluation of subdivision surfaces on programmable graphics hardware, 2003.

[3] T. Boubekeur and C. Schlick. Generic mesh refinement on gpu. *SIGGRAPH/Eurographics Graphics Hardware*, 2005.

[4] T. Boubekeur and C. Schlick. *GPU Gems 3*, chapter Generic Adaptive Mesh Refinement. nVidia, 2007.

[5] M. Bunnell. *GPU Gems 2*, chapter Adaptive Tesselation of Subdivision Surfaces w/ Displacement Mapping. nVidia, 2005.

[6] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological surfaces. *CAD*, 10(6), 1978.

[7] C.Loop and S. Schaefer. Approximating catmull-clark subdivision surfaces with bicubic patches. Technical report, Microsoft Research MSR-TR-2007-44, 2007.

[8] C. Loop. Smooth subdivisions surfaces based on triangles. Master's thesis, University of Utah, 1987.

[9] K. Pulli and M. Segal. Fast rendering of subdivision surfaces. *Eurographics Workshop on Rendering*, 1996.

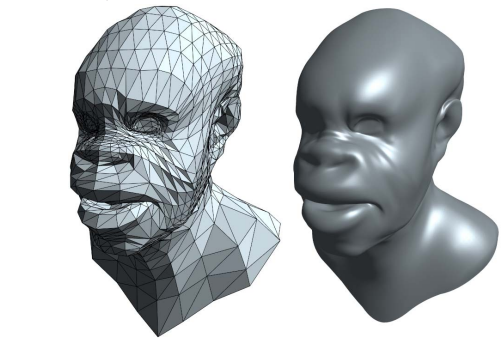
[10] L.-J. Shiue, I. Jones, and J. Peters. A realtime gpu subdivision kernel. *ACM Siggraph*, 2005.

[11] J. Stam. Evaluation of loop subdivision surfaces. ACM SIGGRAPH Course Notes, 1999.

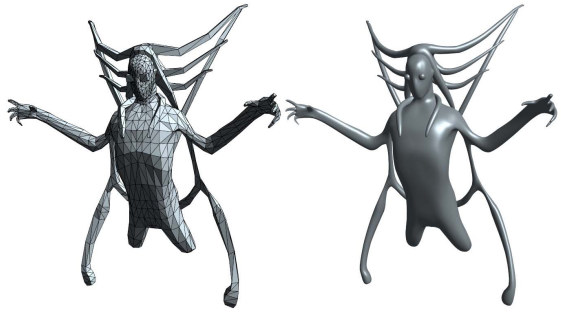
[12] A. Vlachos, J. Peters, C. Boyd, and J. Mitchell. Curved PN triangles. *ACM I3D*, 2001.

[13] D. Zorin and P. Schroder. Subdivision for modeling and animation. *ACM SIGGRAPH Courses Notes*, 2000.

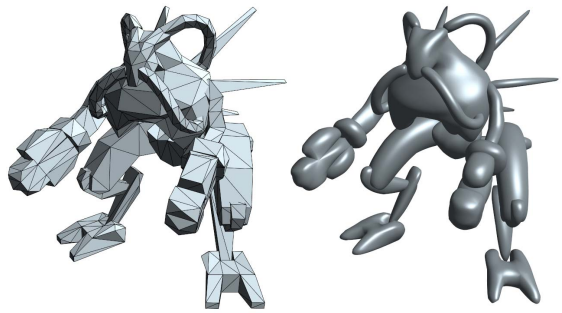
[14] D. Zorin, P. Schroeder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. *ACM SIGGRAPH*, 1996.



(a) 2720 coarse tri., depth 5 (2.7M tri.) - 113FPS



(b) 2516 coarse tri., depth 5 (2.5M tri.) - 128FPS



(c) 1246 coarse tri., adaptive depth 6 (2.6M tri.) - 132FPS

Figure 5: *Additional examples of real-time approximation of subdivision surfaces.* **Left:** *On CPU* **dynamic** *coarse mesh.* **Right:** *Realtime geometry synthesis on GPU produced by QAS.*