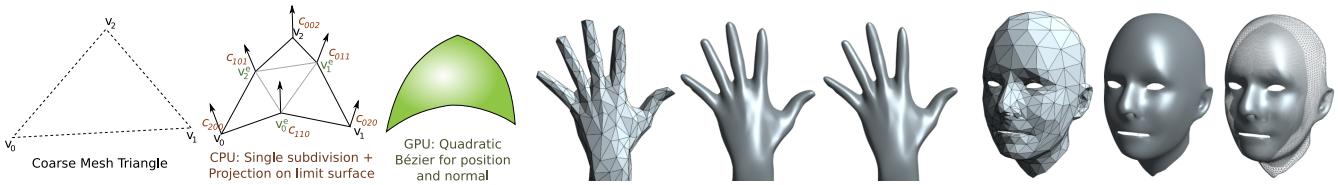
Approximation of Subdivision Surfaces for Interactive Applications

Tamy Boubekeur, Christophe Schlick

Approximation of Subdivision Surfaces for Interactive Applications

Tamy Boubekeur* Christophe Schlick†
LaBRI - INRIA - University of Bordeaux



(a) Coarse triangle, enriched hexagon and final smooth patch (b) 546 to 500k Triangles - 527 FPS (c) 703 to 620k Triangles - 499 FPS

Figure 1: *(a) Approximation principle. (b) Coarse mesh, approximated subdivision and true subdivision (depth 5). (c) Adaptive rendering.*

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 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. We propose a new efficient approximation of subdivision surfaces which offers a very close appearance compared to the true subdivision surface while being at least one order of magnitude faster than true subdivision rendering. Our technique 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 input 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 on the CPU, allowing real-time performances even for million polygons output. Additionally, the parametric nature of the approximation allows an efficient adaptive sampling for both adaptive rendering and displacement mapping.

2 Approximated Subdivision

The very first subdivision step provides a crucial information, particularly when using *limit rules*: it indicates in which direction the surface will converge for all its edges. We propose to use this initial guess of the first subdivision step performed on the CPU to compute a local quadratic Bezier approximation on the GPU. Instead of using an empirical estimation [Vlachos et al. 2001] of the Bezier coefficients, we fit two Bezier patches on the limit positions (resp. normals) provided by the single subdivision step with projection on the limit surface.

CPU Support The algorithm starts by applying a single subdivision step using limit masks. Each triangle T is thus split in 4 sub-triangles, with vertices on the limit surface. These sub-triangles share 6 vertices (Figure 1(a)) and the sub-mesh can thus be organized in an hexagonal shape $H = \{v_0, v_1, v_2, v_0^e, v_1^e, v_2^e\}$ with $v_i = \{p_i, n_i\}$ being the limit positions and normals at this location. This structure is adapted to recent hardware including a geometry shader stage, which allows to transmit edge neighbors with primitives: here we transmit edge vertices inserted by the subdivision pass instead. Note that we use the Loop scheme for producing approximating subdivision. Alternatively, the Modified Butterfly scheme can be used for interpolating the input vertices.

GPU Polynomial Approximation Once transmitted to the GPU, a shader (either vertex shader on old devices or geometry shader on recent ones) automatically fits 2 Bézier patches to H: $P(u,v)$ for positions and $N(u,v)$ for normals. Both patches are defined by:

$$Q(u,v,w) = \sum_{i+j+k=2} b^2_{ijk}(u,v,w)c_{ijk} \text{ with } b^2_{ijk} = \frac{2!}{i!\,j!\,k!}u^i v^j w^i \text{ and } w = 1-u-v$$

In practice, c_{ijk} is replaced by p_{ijk} or n_{ijk} (see Figure 1(a)). We use quadratic patches as they provide a good trade-off between curvature reproduction and computation cost. Such patches require 6 coefficients, organized as an hexagon. Three of them correspond to the original vertices $\{v_0, v_1, v_2\}$ projected at the limit and are naturally interpolated by Bezier patches, while the three others 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 P (resp. N) 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:

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

Other edge coefficients are obtained by symmetry, and the same principle is used for computing the Bézier patch for normals.

Adaptive Rendering Substituing patches to recursive rules allows direct evaluation at arbitrary parameter value. 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 depth, either on CPU or GPU, using for instance a simple camera distance or curvature metric. Then, adaptive tessellation can be performed by directly using the geometry shader for low depth and using the adaptive refinement kernel of [Boubekeur and Schlick 2007] for higher up-sampling ratio.

3 Results

While being geometrically only C^0, the resulting surface has an appearance almost indistinguishable from the true subdivision surface. This is due to the separate normal fitting, which ensures both a smooth shading and reproduces the surface inflexions sampled by edge limits normals. Considering performances, our approach outperforms existing solutions [Shiue et al. 2005] for three reasons: we only perform a single subdivision pass on CPU, we use a single rendering pass on GPU whatever the depth and there is no geometry-to-texture conversion such as [Shiue et al. 2005]. Note also that the mesh is synthesized on the fly, without storing the topology of the high resolution mesh. As a result we obtain real-time performance (> 100 FPS) for objects composed of thousand coarse polygons, subdivided at depth 5 ($2M$ tessellated triangles). Performances degrades linearly with the number of triangles created and transmitted at CPU level. As a limitation, note that the higher the vertex valence is, the less accurate becomes our fast approximation. This can be prevented by remeshing. Last, the direct adaptive rendering allowed by our method, combined with its low CPU overhead makes this approximation particularly suitable for interactive applications, with much better results than purely empirical smoothing methods.

References

BOUBEKEUR, T., AND SCHLICK, C. 2007. *GPU Gems 3*. NVidia, ch. Generic Adaptive Mesh Refinement.

SHIUE, L.-J., JONES, I., AND PETERS, J. 2005. A realtime gpu subdivision kernel. *ACM SIGGRAPH*.

VLACHOS, A., PETERS, J., BOYD, C., AND MITCHELL, J. 2001. Curved PN triangles. *ACM I3D*.

*e-mail: Tamy.Boubekeur@labri.fr
†e-mail: Christophe.Schlick@labri.fr