Synthesizing Subdivision Meshes using Real Time Tessellation

Matthias Holländer
Telecom ParisTech - CNRS LTCI
Paris, France
hollaender@telecom-paristech.fr

Tamy Boubekeur
Telecom ParisTech - CNRS LTCI
Paris,France
tamy.boubekeur@telecom-paristech.fr

***Abstract*—We propose a new GPU method for synthesizing subdivision meshes with exact adaptive geometry in real time. Our GPU kernel builds upon precomputed tables of basis functions for subdivision surfaces and is therefore supporting all subdivision schemes, either interpolating or approximating, for triangle or quad meshes. We designed our kernel so that it can be integrated seamlessly within a standard tessellation pipeline, exploiting software or hardware (adaptive) tessellation methods. We make use of the tessellator unit as an adaptive mesher for maximum subdivision level, exploiting the linear nature of subdivision surfaces to enable arbitrary level of detail adaptivity and control the visual smoothness using Subdivision Shading by applying the same tables as for geometry. We evaluate our kernel on a variety of dynamic meshes and compare it to subdivision substitutes.**

***Index Terms*—Subdivision Surfaces;Real time Tessellation;GPU Programming**

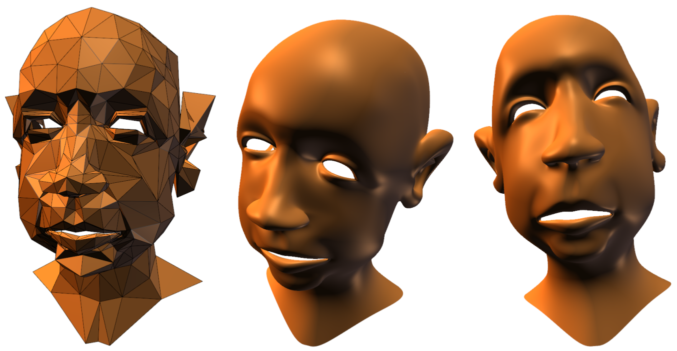


Fig. 1: *Real time adaptive GPU subdivision meshes (2 frames on the right) from a dynamic coarse mesh (left).*

I. Introduction

Real time tessellation methods allow to refine surface meshes by generating a denser set of polygons on the fly. The domain surface can be kept coarse to ease animation and manipulation at application level, while newly inserted vertices are often used to sample a smooth surface before eventually moving them according to a scalar or vector displacement map. As the input mesh may have an arbitrary topology, subdivision surfaces are frequently considered as a good representation for the underlying smooth surface to sample. However, due to the cost of exact evaluation, these surfaces are often replaced by approximations which can be evaluated at any parameter domain point in linear time, without relying on recursive schemes. Unfortunately, while a large variety of subdivision surface models exist, most of the approximation schemes are dedicated to a particular model (e.g., interpolating subdivision surfaces for triangle domain). In this work, we show how real time tessellation can be used as an adaptive mesher for **any** subdivision scheme by using precomputed basis functions tables (BFT). We propose a GPU implementation that reaches high frame rates while outputting dense dynamic subdivision meshes and integrates seamlessly into the standard tessellation pipeline, consuming only a moderate amount of GPU memory. Additionally, we suggest an adaptation of *Subdivision Shading* to such an evaluation which, although different, offers a similar smoothness control mechanism as the original method, still avoiding recursion or exact parametric evaluation.

A. Related Work

Subdivision Surfaces: A subdivision surface [1] is a smooth parametric surface of arbitrary topology defined by a base polygonal mesh (domain) and a subdivision scheme. A subdivision scheme defines a smooth surface entirely by either approximating (e.g., Catmull-Clark scheme [2] for quad meshes, Loop scheme [3] for triangle meshes or Loop-Stam scheme [4] for tri-quad meshes) or interpolating (e.g., Modified Butterfly scheme [5] for triangle meshes) the base meshes vertices. They are often defined by a collection of subdivision masks tailoring a recursive tessellation process interleaved with local filtering operators and generate a denser subdivision mesh closer to the limit (continuous) subdivision surface at each step. For many schemes, subdivided vertices can be projected to their limit position directly, therefore sampling the subdivision surface exactly. When a subdivision scheme is derived explicitly from a spline basis, an exact, recursion-free evaluation allows to sample the surface at arbitrary parameter values. Recursive or parametric evaluations are usually both too expensive for practical interactive applications. However, as any point of a subdivision mesh can be defined as a weighted combination of base mesh vertices, it is possible to precompute a table of basis functions [6]. The weights contained within this table are associated to each base vertex for each fine vertex v for which the so-defined combination provides the position of v. These weights depend on the base domain connectivity and the tessellation level and can be

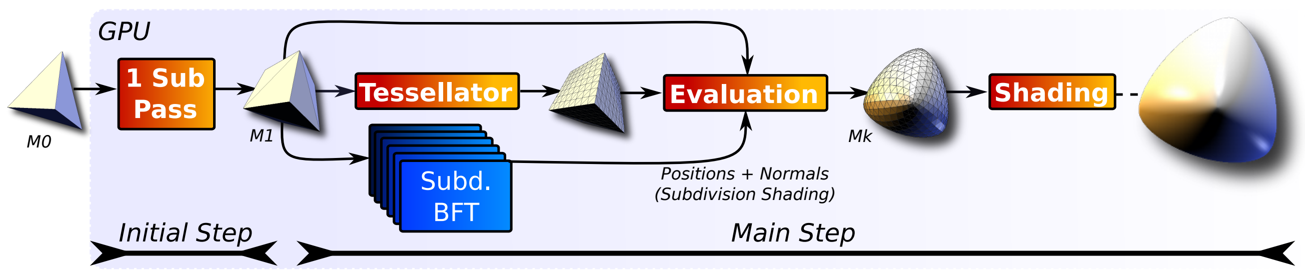


Fig. 2: *Our GPU subdivision mesh synthesis pipeline.*

generated for limit positions as well, i.e. the projection of v onto the subdivision surface, which is equivalent to an infinite number of subdivision steps.
Alternatively, subdivision surfaces can be efficiently sampled on the CPU at rational parameters by combining translation and scaling functions [7], therefore computing the basis functions on-the-fly. However, this is not easily adaptable to a GPU implementation.
Real Time Tessellation: Mesh tessellation can be performed on the fly and adaptively on the GPU. At first stages of the rendering pipeline, the input mesh is super-sampled, generating a large number of fine polygons replacing the input ones and displacing the resulting vertex set to sample a given function defined on the base surface. The entire process is repeated at each frame, enabling the application to only manage the coarse (dynamic) mesh at application level (e.g., animation, physics, interaction) while a high resolution mesh is synthesized and displayed in real time. Such tessellation methods can be implemented on any GPU equipped with vertex shading capabilities [8], [9], exploit GPU computing environments [10] and benefit from recent hardware support since DirectX 11 and OpenGL 4 graphics API versions. One noticeable property of such methods is their ability to tessellate *adaptively* the input mesh, providing watertight high resolution meshes with spatially varying density. Although subdivision surfaces may appear as a natural application for real time tessellation, recursive or parametric evaluation are often considered too slow for typical interactive applications and a number of alternative curved surface models have been developed for this purpose.

Fast Curved Surface Models: Substitutes to subdivision surfaces [11] offer visual smoothness with a close look to what subdivision surfaces produce and without the need for mapping the recursive evaluation to the GPU [12], [13]. These representations are often based on low degree spline patches [14]. For instance, a *Curved PN Triangles* [15] is a triangular Bézier patch built solely from the input triangle (vertex positions and normals). Such local schemes can only mimic high order continuity by exploiting a similar strategy to Phong Normal Interpolation [16]: a synthetic normal field is generated using interpolation over the input vertex normals, independently of the actual geometric differentials. This interpolation is quadratic [17] in the case of PN Triangles and can even make use of subdivision basis [18]. A number of subdivision surface approximations [19], [20], [21] have been proposed following a similar strategy. Simpler operators such as Phong Tessellation [22] can also offer an economic way to get rid of most of the typical visual artifacts stemming from coarse meshes. Although all of those methods produce smoother high-resolution versions of the input mesh, none of them can reproduce the high quality of a true subdivision surface.

B. Contribution

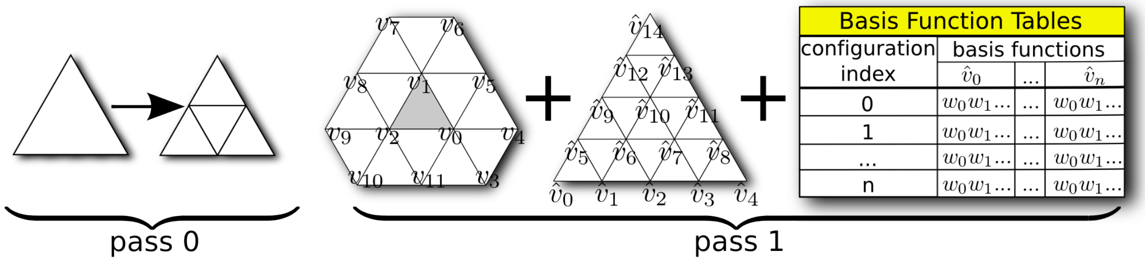
We propose an adaptive subdivision surface meshing algorithm which exploits real time GPU tessellation to produce dynamic subdivision meshes on-the-fly. Our geometry synthesis kernel is oblivious to the particular subdivision scheme in use and can therefore be combined with all classical ones (i.e., stationary local schemes with compact support). We use BFTs at *maximum level* to index an adaptive triangulation and exploit the same tables to generate a smooth normal field in a similar way as subdivision shading [18]. Our kernel runs at high frame rates for dynamic input base meshes with deep (adaptive) tessellation ratios and is fully compatible with the standard tessellation pipeline. On the contrary to subdivision substitutes, we do not aim at producing visually smooth surfaces only but rather propose to carry off computations from CPU to GPU for all applications exploiting subdivision meshes (e.g., high-end modeling packages) without the need to switch to a new representation.

Notations: We denote $\mathbf{M}^0$ as the base mesh and $\mathcal{S}$ as a given local and compact subdivision operator. $\mathbf{M}^k = \mathcal{S}^k(\mathbf{M}^0)$ is the subdivision mesh after k steps of subdivision applied on $\mathbf{M}^0$. Any mesh $\mathbf{M}^i$ is composed of a vertex set $\mathbf{V}^i = \{v_j^i\}$ and a face set $\mathbf{F}^i = \{f_j^i\}$.

II. GPU Subdivision Kernel

A. Overview

The basic idea is to use the tessellation unit (either GPU emulated or hardware supported) as a real time *adaptive mesher* for a *maximum resolution* BFT. As usual with BFTs, several tables have to be generated, one for each input face connectivity configuration. Following the idea of Bolz and


pass 0
pass 1
v_7 v_6 v_8 v_1 v_5 v_9 v_2 v_0 v_4 v_{10} v_{11} v_3
$\hat{v}_{14}$ $\hat{v}_{12}$ $\hat{v}_{13}$ $\hat{v}_9$ $\hat{v}_{10}$ $\hat{v}_{11}$ $\hat{v}_5$ $\hat{v}_6$ $\hat{v}_7$ $\hat{v}_8$ $\hat{v}_0$ $\hat{v}_1$ $\hat{v}_2$ $\hat{v}_3$ $\hat{v}_4$

Basis Function Tables

configuration index	basis functions		
	$\hat{v}_0$	...	$\hat{v}_n$
0	$w_0 w_1 \ldots$	...	$w_0 w_1 \ldots$
1	$w_0 w_1 \ldots$	...	$w_0 w_1 \ldots$
...	$w_0 w_1 \ldots$	...	$w_0 w_1 \ldots$
n	$w_0 w_1 \ldots$	...	$w_0 w_1 \ldots$

Fig. 3: *An input triangle t^0 is subdivided once on the GPU resulting in four triangles t_i^1. Each triangle t_i^1 is tessellated (pass 1 middle) – for instance by instancing a refinement pattern or by using the DX11 tessellator – and related basis functions are queried. Finally, for each created vertex $\hat{v}_i$ the linear combination given by the weights w_i and the vertices v_i is computed.*

Schröder [6], we do not generate all the intermediate BFTs and directly precompute a maximum table, for instance at level 5 which corresponds to a tessellation pattern of 32x32 faces. The weights stored by these maximum BFTs correspond to limit projections: all vertices evaluated using these tables will lay on the (limit) subdivision surface. Therefore, there is no difference between a subdivision vertex at level 3 or 5 for instance and alternative adaptive triangulations can exploit the same set of maximum level tables, whatever the desired number of vertices and triangles. Our algorithm is described in Figure 2 and makes use of 3 major components:

- An initial GPU subdivision pass is applied on $\mathbf{M}^0$ to isolate extraordinary vertices (to reduce the number of tables required) and to initialize a normal field on $\mathbf{M}^1$ for subsequent *Subdivision Shading*.
- Uniform or adaptive tessellation is performed on $\mathbf{M}^1$, refining it directly to level k. This step can exploit recent hardware tessellation units [23], [24] or GPU implementations [8], [9], [10]
- A set of basis function tables $\{B^i\}$ are precomputed and stored on the GPU. At rendering time, vertex positions and normals of $\mathbf{M}^k$ are computed using linear combinations of vertices of $\mathbf{M}^1$, with weights stored in $\{B^i\}$. The spherical averages required to interpolate points on the Gauss sphere are approximated using a single normalization [18].

Basis function tables speedup the evaluation of points on the limit subdivision surface in contrast to complicated and slow direct evaluation [25] or GPU recursive emulations [12], [13]. They can be obtained by generating *configuration meshes* which consist of a single triangle and its one-ring neighbourhood as depicted in Figure 3 (lower left). All vertices are placed within the xz-plane except one which is moved to $z = 1$. This configuration mesh is subdivided with a given scheme and the basis functions are given in the z-values. This procedure is repeated for all vertices of the configuration mesh.

Unfortunately, the number of basis function tables increases quickly for a mesh with arbitrary connectivity. However, applying a single subdivision step isolates extraordinary vertices and produces faces at level 1 with one extraordinary vertex at most. This mechanically diminishes the combinatorics for precomputation and the number of BFTs to store for a bounded valence (up to 18 in our experiments). We implement this initial subdivision step on the GPU and use $\mathbf{M}^1$ in all subsequent GPU steps. Consequently, the entire algorithm is executed on the GPU and the input coarse mesh can be provided either from the main (CPU) application (e.g., high end modeling packages such as Maya or 3DS Max) or from the GPU itself (e.g., GPU skinning). After this step all faces will have at most one extraordinary vertex and are ready for further tessellation and evaluation using basis function tables.
For the sake of simplicity, the discussion will be limited to the Loop subdivision because triangles are ubiquitous for real time rendering and the subdivision scheme produces a surface that is at least C^1-continuous everywhere. However, our approach is not specific to this scheme and can be used with other schemes as well (e.g., Catmull-Clark, Butterfly).

Preprocessing: To prepare the input mesh for our pipeline and to reduce the computational effort during runtime, several preprocessing steps are applied at initialization time. First, we load pregenerated basis function tables as in [6]. We store them as floating point textures which allow random access from within a shader. Afterwards, we allocate the space required to store $\mathbf{M}^1$ on the GPU which will be filled using the initial GPU subdivision step. At this step we impose $\mathbf{M}^1$ to be a uniform tessellation of $\mathbf{M}^0$ to avoid any memory manipulation at runtime and directly specify its connectivity. The advantage is that $\mathbf{M}^0$ can now be animated on the GPU without memory layout modification as long as its connectivity does not change (i.e., $\mathbf{F}^1$ is static). Note that the GPU memory cost for a given object is small and independent of its actual final resolution and that the BFTs can be shared by all objects in the scene (see Fig. 3).

B. Initial GPU Subdivision Step

A single step of uniform subdivision multiplies the number of triangles by a small fixed ratio (low data amplification) but may have to handle a large spectrum of connectivity configurations. Again, each vertex $v_i^1 \in \mathbf{V}^1$ is a linear combination

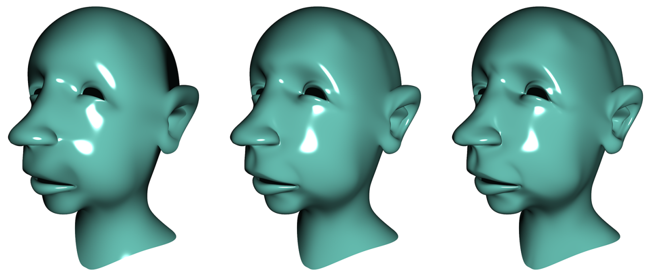


Fig. 4: *M^5 with the subdivision shading procedure starting at level 0, 1 and 2.*

of the vertices of $\mathbf{V}^0$ and a large part of the evaluation cost can be cached by applying a similar strategy as BFTs but specialized to this single step: we record for each vertex of $\mathbf{V}^1$ a *computational entry* consisting of weights and indices of its parent vertices in $\mathbf{V}^0$ and update $\mathbf{V}^1$ each time $\mathbf{V}^0$ undergoes a deformation.

Normal vectors of $\mathbf{V}^1$ are computed using a similar approach. This time, each *computational entry* for a surface normal of $\mathbf{M}^1$ has to keep track of the vertices which contribute to its calculation, namely its one-ring neighborhood. These vertex indices are encoded into the data structure ordered consistently in counterclockwise order together with a flag indicating whether the normal is located on a boundary. For the latter case the related triangle fan is treated as *open* and a different set of tables will be used in the main pass.

```
struct BFTPatch {
    uint m_uiPoints[k];
    uint m_uiIndexBFT;
    uint m_uiOneRingSize;
};
```

The values stored in `m_uiPoints` index control points of the patch, i.e., vertices of the patch and its one-ring neighbourhood. `m_uiIndexBFT` is an index into the basis function table, indicating the set of basis functions to use for the particular patch connectivity. This value is induced by the related face on $\mathbf{M}^1$, its location (interior, bordering, etc.) and the valence of the unique (possibly) extraordinary vertex. `m_uiOneRingSize` is used as a break condition while looping over the one-ring neighbourhood in the main GPU Step.

Note that the initial subdivision step could directly apply Subdivision Shading [18] using the coarse mesh' normals and only a single texture. However, we found the visual result to be too smooth sometimes resulting in a loss of visual quality. Similarly, one could delay the evaluation of geometric normals to a deeper level for starting Subdivision Shading at the expense of a lower frame rate. Remind that tangent masks can be used to compute geometric normals, which leads to the usual defects around extraordinary vertices [18].

GPU Implementation: First, we need the attributes of $\mathbf{M}^0$ as random shader-accessible resources (*textures* for OpenGL and *buffers* for DirectX). We convert the mesh using GPU *stream output* in a preliminary pass, where stream tokens should be understood as *vertex positions*, *vertex normals* etc. GPU deformation can also be applied during this stage.

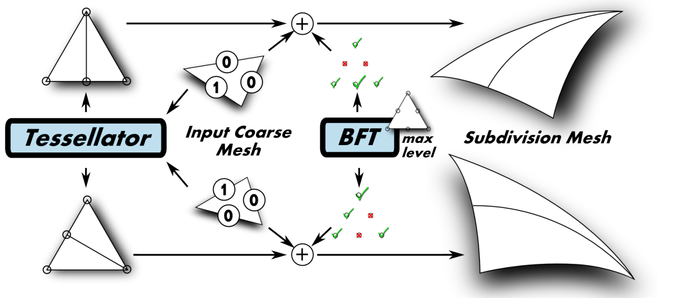


Fig. 5: *The real time tessellation GPU unit as an adaptive subdivision mesher.*

In a second pass, we initiate n instances of a single point, where n is equal to the number of vertices of $\mathbf{M}^1$. At shader level, each instance is identified by its ID and used to lookup the corresponding *computational entry* from GPU memory (e.g., textures). Therefore, each vertex of $\mathbf{M}^1$ can be computed by evaluating the linear combination as encoded in the entry. The result is streamed into a random access shader resource.

In the following pass, the surface normals of $\mathbf{M}^1$ are computed by again creating n instances of a single point. This time, normal computational entries are queried and the result is streamed to complete $\mathbf{M}^1$ in GPU memory.

Any additional vertex attributes can be treated with a similar strategy using *Multiple Render Targets* (MRT) and the calculation entries from either resource (e.g., texture coordinates can use the same weights as positions [26]). MRT also allow to pack these 2 last passes into a single one when using subdivision *gradient* masks. We however kept this 2-pass configuration for enabling alternative normal computation.

Note that hardware instancing of a single point is very fast and can be seen as a thread creation for each vertex (position, normal etc.). The initial GPU step does not, overall, represent the bulk of the computation.

C. Main GPU Subdivision Step

Positions and normals computed at the aforementioned step as well as the basis function tables are used as random access shader resources in the main step. For each face/patch $\mathbf{f}_i^1 \in \mathbf{F}^1$, tessellation is performed at desired level, outputting a dense set of parameter values. These values are 2D coordinates of points laying on $\mathbf{f}_i^1$ and the tessellator ensures that the corresponding fine triangulation is generated. We now need to evaluate the 3D position of the vertex from this 2D parameter $\{u, v\}$ and the values computed at the initial step (see Fig. 3). We first map $\{u, v\}$ onto an integer subindex: as maximum

BFTs correspond to uniform tessellation at maximum level, this subindex is trivially computed as the 1D parameter of $\{u, v\}$ on the tessellation's space filling curve. The evaluation of a surface point given by the refinement vertex $\hat{\mathbf{v}}_i$ is a linear combination of the vertices $\mathbf{v}_i^1$ of $\mathbf{M}^1$ and the basis functions as weights.

$$f(u,v) = \sum_i B^i(u,v)\mathbf{v}_i^1 \tag{1}$$

The basis functions for each surface point are given by the connectivity index of the patch and the subindex. The evaluation of equation 1 in the shader is very fast as it boils down to a simple loop in the following shader code:

```
main() {
    uint  uiPatchIdx  = GetPatchIndex(gl_InstanceID);
    BFTPatch p = LoadPatch(uiPatchIdx);
    vec3 vPatchPos[] = LoadPatchPos(p);
    vec3 vPatchNormal[] = LoadPatchNorm(p);
    float  fBFT[]=LoadBFT(p.m_uiIndexBFT, gl_Vertex.x);
    vec3 vPos = vec3(0.f, 0.f, 0.f);
    vec3 vNormal = vec3(0.f, 0.f, 0.f);
    for ( int  i=0; i<p.m_uiOneRingSize; ++i ) {
        float  fWeight = fBFT[i];
        vPos += fWeight * vPatchPos[i];
        vNormal += fWeight * vPatchNormal[i];
    }
    gl_Position  = gl_ModelviewProjection * vPos;
    vNormalOut = gl_NormalMatrix * vNormal;
    //  ...
}
```

The surface normals are computed similarly to the vertex positions by applying subdivision shading [18]. Basically the same weights are used for vertex-positions and vertex-normals, but the combination is performed on the Gauss sphere. As for the original technique, this interpolation is usually well approximated using a single exponential map interpolation, i.e., euclidean interpolation followed by a normalization. Of course, tangent BFTs or an extra pass allow to evaluate the geometric normals, but it usually leads to lower shading quality [18].

Adaptive Refinement: As mentioned in [6] the tables of basis functions can be subsampled thus allowing for adaptive refinement as well. Since we consider limit surface points at any level, all basis functions of level n can be seen as a subset of basis functions of level $n+1$. The tessellation unit offers adaptive level of tessellation within a triangle when specifying different tessellation ratios for the edges of an input face. Therefore, our approach is trivially made adaptive since our algorithm does not rely on the particular connectivity of $\mathbf{M}^k$: a triangle with different tessellation ratios (i.e., fine adaptive triangulation) makes use of the same BFT (see Fig. 5). Although it is not possible to sample the subdivision surface at arbitrary parameter domain with our method (e.g., for Loop, fine vertices must all be located at dyadic split positions), we can still offer linear geomorph transition, i.e. with all vertices laying on a virtual mesh subdivided at maximum level.

III. Results and Performances

The performance of our algorithm was tested on a GeForce GTX 295, 1.8 GB graphics memory and an Intel Core i7 2.67 GHz using OpenGL under Windows (see Table I). We measured frame rates of our algorithm in combination with the Adaptive GPU Refinement Kernel [9] as a tessellator emulator. Although this implementation does not reflect the performances obtained with recent genuine hardware tessellation units, it helps to understand how the overall workload for exact subdivision mesh generation compares to fast approximations and subdivision substitutes. Table I gives the frame rate for several models at various subdivision levels and compares it to a variety of substitutes.

Surprisingly, while the presented subdivision pipeline is, of course, more expensive than local refinement schemes such as Phong Tessellation [22] or PN Triangles [15], it still succeeds at offering real time performances and significantly higher surface quality (see Fig. 7). For instance, it is only slightly more expensive than subdivision surface approximation schemes such as the QAS model [19] which also relies on an initial subdivision step (that we implemented on the GPU in our framework). We also measure flat tessellation (no fine vertex displacement) to better quantify the cost of the subdivision evaluation. Overall, the performance of our simple approach shows that subdivision surfaces can be created on the fly on the GPU with interactive performance and without resorting to substitutes. Of course, our approach relies on subdivision meshes and arbitrary parameter evaluation is bounded to linear interpolation on the fine triangles.

Applications such as games might preferably use subdivision substitutes as only visual smoothness and high frame rates are critical. However, for all applications which are using standard subdivision surfaces with recursive evaluation bounded to a limited number of subdivision steps (e.g., high end computer graphics packages for SFX and Animation), our experiments show that the CPU computational workload can be significantly decreased by offloading the entire process to our

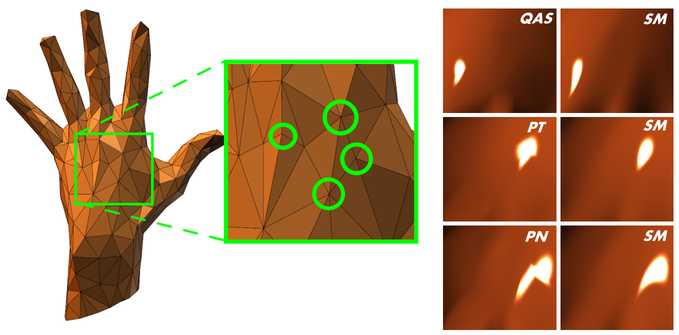


Fig. 6: *Surface quality around extraordinary vertices: comparison between quadratic approximation (QAS), Phong Tessellation (PT), PN-Triangles (PN) an our subdivision method (SM).*

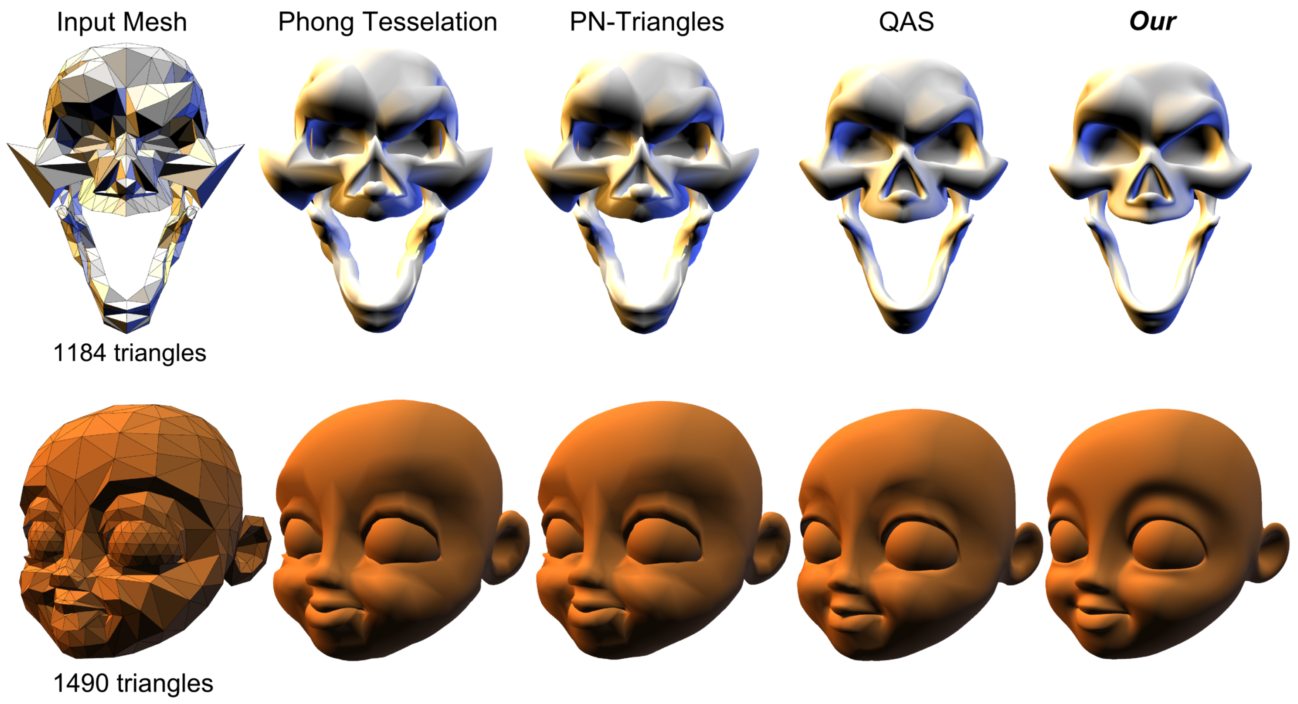


Fig. 7: GPU Refinement*: subdivision substitutes versus our GPU subdivision meshes ($\boldsymbol{M}^5$).*

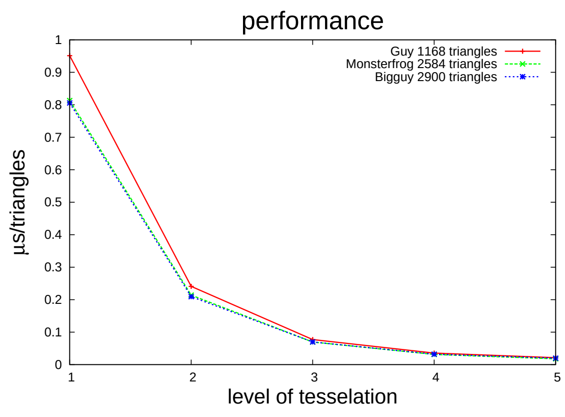


Fig. 8: *Per-triangle rendering cost as a function of the tessellation level at which it is synthesized.*

Model	Head	Guy	Frog	Big Guy
#Input tri.	524	1168	2584	2900
Tessellation Level 2 (4x4 split)				
#Output tri.	8384	18688	41344	46400
Frame rate (in fps)				
SM	641	495	334	291
Tessellation Level 4 (16x16 split)				
#Output tri.	134k	299k	661k	742k
Frame rate (in fps)				
SM	312	176	85	76
Tessellation Level 5 (32x32 split)				
#Output tri.	536k	1196k	2646k	2969k
Frame rate (in fps)				
SM	103	50	24	21
QAS	80	53	44	23
PN-Tri.	109	69	41	35
PT	120	77	48	40
Flat	137	85	52	45

TABLE I: *Performance measure.*

two step GPU algorithm, in particular regarding the per-fine triangle cost for deep subdivision level (see Fig. 8). The final result is shaded thanks to Subdivision Shading normals and is, regardless of the valence, visually smoother (see Fig. 6) than linear [16] or quadratic [17], [15], [19] normal interpolation.

Future work: Although the memory consumption is not as critical as for early GPU implementations [27] of basis function tables (about 9 MB for tables of level 5 for interior and boundary cases up to valence 18), we think that the computation time could be improved by quantizing the tables to reduce cache usage. This quantization should ensure some symmetry to guarantee crack-free subdivision meshes and for entropy. We plan to experiment with several cache optimization strategies and face reordering based on vertex valence rather than spatial proximity.

IV. Conclusion

We have shown that a simple combination of basis function tables and GPU tessellation enables dynamic subdivision meshing at high frame rates with dense output. Our approach makes adaptive GPU subdivision trivial, can be implemented on any programmable GPU using existing tessellation kernels and is ready to exploit new hardware tessellation units. Moreover, it is compatible with all subdivision schemes, does not require any particular scheme-specific setup and can provide smoother shading using Subdivision Shading. We believe that our method can be useful to developers using subdivision surfaces who want to exploit the latest GPU generations with tessellation capabilities without switching to other surface models such as subdivision substitutes.

V. Acknowledgements

This work has been partially funded by the French National Research Agency (ANR *MediaGPU* project) and the *3DLife* European Network of Excellence.

References

[1] D. Zorin, P. Schröder, A. Levin, L. Kobbelt, W. Sweldens, and T. DeRose, "Course Notes Subdivision for Modeling and Animation," in *ACM SIGGRAPH*, 2000.

[2] E. Catmull and J. Clark, "Recursively generated B-spline surfaces on arbitrary topological meshes," *Computer Aided Design*, vol. 10, no. 6, pp. 350–355, 1978.

[3] C. Loop, "Smooth subdivision surfaces based on triangles," *Master's thesis, University of Utah, Department of Mathematics*, 1987.

[4] J. Stam and C. Loop, "Quad/triangle subdivision," in *Computer Graphics Forum*, vol. 22. Blackwell Publishing, Inc, 2003, pp. 79–85.

[5] N. Dyn, D. Levine, and J. Gregory, "A butterfly subdivision scheme for surface interpolation with tension control," *ACM transactions on Graphics (TOG)*, vol. 9, no. 2, pp. 160–169, 1990.

[6] J. Bolz and P. Schröder, "Rapid evaluation of Catmull-Clark subdivision surfaces," in *ACM Web3D*, 2002, pp. 11–17.

[7] S. Schaefer and J. Warren, "Exact evaluation of non-polynomial subdivision schemes at rational parameter values," *Computer Graphics and Applications, Pacific Conference on*, vol. 0, pp. 321–330, 2007.

[8] T. Boubekeur and C. Schlick, "Generic mesh refinement on GPU," in *ACM SIGGRAPH/EUROGRAPHICS Graphics hardware*, 2005, pp. 99–104.

[9] ——, "A flexible kernel for adaptive mesh refinement on GPU," in *Computer Graphics Forum*, vol. 27, 2008, pp. 102–113.

[10] M. Schwarz and M. Stamminger, "Fast gpu-based adaptive tessellation with cuda," *Computer Graphics Forum*, vol. 28, no. 2, 2009.

[11] T. Ni, I. Casta no, J. Peters, J. Mitchell, P. Schneider, and V. Verma, "Efficient substitutes for subdivision surfaces," in *ACM SIGGRAPH Courses*, 2009, pp. 1–107.

[12] L. Shiue, I. Jones, and J. Peters, "A realtime gpu subdivision kernel," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1010–1015, 2005.

[13] X. H. Kun Zhou, W. Xu, B. Guo, and H.-Y. Shum, "Direct manipulation of subdivision surfaces on gpus," *ACM Transactions on Graphics (SIGGRAPH 2007)*, 2007.

[14] Y. I. Yeo, T. Ni, A. Myles, V. Goel, and J. Peters, "Parallel smoothing of quad meshes," *Vis. Comput.*, vol. 25, no. 8, pp. 757–769, 2009.

[15] A. Vlachos, J. Peters, C. Boyd, and J. Mitchell, "Curved PN triangles," in *Proceedings of the 2001 symposium on Interactive 3D graphics*. ACM New York, NY, USA, 2001, pp. 159–166.

[16] B. T. Phong, "Illumination of computer-generated images," Ph.D. dissertation, University of Utah, 1973.

[17] C. Van Overveld and B. Wyvill, "Phong normal interpolation revisited," *ACM ToG*, vol. 16, no. 4, pp. 397–419, 1997.

[18] M. Alexa and T. Boubekeur, "Subdivision shading," *ACM ToG (Proc. SIGGRAPH Asia)*, vol. 27, no. 5, pp. 142:1–142:4, 2008.

[19] T. Boubekeur and C. Schlick, "QAS: Real-time quadratic approximation of subdivision surfaces," in *Pacific Graphics*, 2007, pp. 453–456.

[20] C. Loop and S. Schaefer, "Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches," *ACM Trans. Graph.*, vol. 27, no. 1, pp. 1–11, 2008.

[21] C. Loop, S. Schaefer, T. Ni, and I. Castaño, "Approximating subdivision surfaces with Gregory patches for hardware tessellation," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 1–9, 2009.

[22] T. Boubekeur and M. Alexa, "Phong tessellation," *ACM ToG (Proc. SIGGRAPH Asia)*, vol. 27, no. 5, pp. 1–5, 2008.

[23] M. Segal and K. Akeley, "The openglm graphics system: A specification (version 4.0, core profile)," The Khronos Group Inc., Tech. Rep., March 2010.

[24] S. Drone, M. Lee, and M. Oneppo, "Direct 3d tessellation," http://www.microsoft.com/downloads/, 2008.

[25] J. Stam, "Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values," in *SIGGRAPH*, 1998, pp. 395–404.

[26] T. DeRose, M. Kass, and T. Truong, "Subdivision surfaces in character animation," in *SIGGRAPH*, vol. 98, 1998, pp. 85–94.

[27] J. Bolz and P. Schröder, "Evaluation of subdivision surfaces on programmable graphics hardware," , 2003.