A VIEW-DEPENDENT ADAPTIVITY METRIC FOR REAL TIME MESH TESSELLATION

Tamy Boubekeur

Telecom ParisTech - CNRS LTCI, Paris, France

ABSTRACT

Real-time tessellation methods offer the ability to upsample 3D surface meshes on the fly during rendering. This upsampling relies on 3 major steps. First, it requires a tessellation kernel which can be implemented on GPU or may be already available as a hardware unit. Second, the surface model defines the positions of the newly inserted vertices - we focus on recent visually smooth models. And third the adaptive sampling tailors the spatially varying distribution of newly inserted vertices on the input surface. We study the last component and introduce a new view-dependent adaptivity metric which builds upon both intrinsic and extrinsic criteria of the input mesh. As a result, we obtain a better vertex distribution around important features in the tessellated mesh. [1]

Index Terms— Computer Graphics, Real Time Rendering, Tessellation, Adaptive Geometry, Meshes.

1. INTRODUCTION

Thanks to the native support offered by modern graphics processor units (GPU), polygonal meshes are the dominant surface representation in most interactive graphics applications (e.g., games). For the sake of real time performance, these meshes are usually (very) coarse and used in combination with a variety of textures and shading methods to depict dynamic objects using only a limited number of polygons, most often triangles. Although piecewise flat, polygonal meshes appear smooth at rendering time thanks to normal interpolation methods and normal maps. However, a large number of artifacts remain visible in feature areas – high curvature, contours, silhouettes, prominant geometric features and virtually all salient regions – stemming from the underlying piecewise linear nature of these coarse surfaces. This list is not exhaustive but one may agree on the fact that artifacts will be particularly problematic in perceptually important regions.

To cope with this problem, a new stage called *tessellation* has been recently introduced in the graphics pipeline. Prior to rendering, real time tessellation is a special case of geometry synthesis and aims at increasing the mesh resolution on-the-fly (see Fig. 1). At rendering time, each input coarse polygon is subdivided, either uniformly or adaptively, and the set of refined vertices is displaced according to a displacement map, a procedural function or an underlying high order surface. The latter case has recently motivated several research projects with the introduction of smooth geometric local upsampling techniques, ranging from close-to-perfect higher order surface approximation to cheap "geometry blurring" methods.

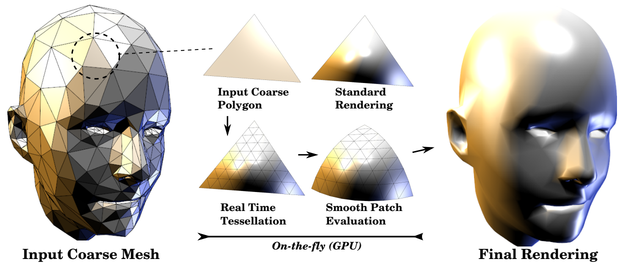

Fig. 1. **Tessellation principle**: at each frame, input coarse polygons are subdivided prior to shading to produce smoother shapes or higher resolution geometric structures.

Background. Basically, any real time tessellator unit, either natively incorporated or implemented on the GPU, is composed of three major steps: a target *geometry function*, evaluated at each fine vertex and stored in a texture or defined analytically (e.g., smooth surface patch); a *tessellation algorithm*, which efficiently generates a large number of polygonal primitives for each coarse input polygon; *a scalar map*, precomputed or computed on-the-fly, which specifies the level of tessellation required at each point of the input surface.

Contribution. Although the first two steps of the tessellation pipeline have been extensively discussed lately, the last one remains particularly unstable, as balancing a budget of polygons to sample a surface involves a large number of intrinsic and extrinsic parameters. In this paper, we restrict our work to the case of tessellation for smooth geometric upsampling entirely defined from a coarse input mesh and show how a new tessellation metric, inspired by recent work in the fields of optimization and *expressive* rendering, achieves better results than previous methods.

2. RELATED WORK

Real Time Tessellation. The introduction of *geometry shaders* (GS) in DX10 hardware did not allow genuine tessellation operators to be implemented. Although more

This work has been supported by the 3DLife European Network of Excellence and the ANR MEDIAGPU project.

[1] *Author's draft. Final version published in IEEE International Conference of Image Processing (ICIP) 2010 Proceedings.*

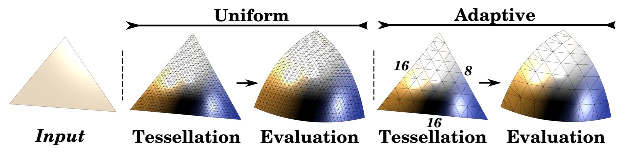

Fig. 2. **Adaptive Tessellation**: spatially varying mesh upsampling decided on a per-primitive basis. Here the tessellation ratio is specified per-edge.

flexible than a tessellator unit, the geometry shader is limited to a rather low upsampling ratio, which limits strongly its application to tessellation. On the contrary, *instanced tessellation* [1][2][3][4] offers a mechanism to quickly tessellate a mesh up to several orders of magnitude denser surfaces at high framerate. The basic idea is to precompute tessellation patterns in a parametric space and to store them in GPU memory. At rendering time, these patterns are drawn *instead* of the coarse polygons, which become transformation parameters. A number of methods have also been proposed to cast tessellation as a generic data upsampling problem, addressing it either using multi-pass texture [5] processing or CUDA [6]. Last, DirectX11 GPUs [7] provide full hardware support for tessellation, introducing a dedicated adaptive tessellator unit as well as 2 new programmable stages: the *hull* shader, defining the geometry function as a set of control points and the *domain* shader, evaluating the tessellated primitives attributes (e.g., position).

Curved Surfaces for Real Time Applications. One of the main applications of real time tessellation is *smooth geometric upsampling*, which covers all techniques aiming at increasing the input mesh resolution while smoothing it. As a result, each input polygon becomes a curved patch. A number of these techniques use a separate normal field computed independently and generalize Phong normal interpolation [8]. Vlachos et al. introduced Curved PN Triangles [9] as cubic Bézier patches constructed from vertex positions and normals of a single triangle. Loop and Schaefer [10] as well as Boubekeur and Schlick [11] proposed to approximate subdivision surfaces using low degree polynomial patches. This enables high framerates as well as better visual quality compared to PN triangles. Last, simpler local operators, such as the Phong Tessellation [12] help hidding the lack of high order continuity of meshes in critical areas (contours, silhouettes).

Adaptivity Metrics on Meshes. The entire tessellation process is conducted by an importance metric defining a spatially varying tessellation ratio. In order to ease adaptive *crack-free* tessellation, this ratio is usually specified on a per-edge or per-vertex basis. In the following, we will not make any strong distinction between both cases and consider edge ratios as the average of edges vertex ratios. Note however that all computations can be performed per-edge directly. In any case, the tessellation unit gathers the requested tessellation level for the current primitive and generates the corresponding amount of vertices (see Fig. 2). Depending on the application, the tessellation ratio may be computed according to various criteria, which can be classified into two families: (**i**) *Extrinsic*: in every point of the coarse input mesh, the view position and direction influence the tessellation ratio. *Proximity* (Fig. 3b) and *silhouetteness* (Fig. 3c) [13][2] [12] are examples of such view-dependent metrics. (**ii**) *Intrinsic*: these criteria can be computed independently of any external information. For static objects, precomputation is an option here. Curvatures (mean, gaussian, max) are examples of such view-independent metrics (Fig. 3d).

Overall, the adaptivity metric is a critical component, because it tailors the distribution of polygons and therefore the geometric computation performed to improve the shape quality. Intuitively, one may want to dedicate most of the GPU's horsepower to *visually important* regions. In this work we exploit recent work in mesh optimization and expressive rendering to tailor a new adaptivity metric achieving this goal.

3. A VIEW-DEPENDENT ADAPTIVITY METRIC

We propose a new adaptive tessellation metric which takes into account both intrinsic and extrinsic criteria. We target smooth geometric upsampling methods which allows us to safely consider the input coarse mesh as a plausible guess of the final high resolution mesh and to evaluate geometric quantities on it.

Visual Importance. Our basic observation is that the amount of tessellation should be related to what we perceive from a shape: this notion of *shape perception* has actually be exploited in other fields, from which we took inspiration.

First, in the context of *surface optimization*, Lee et al. [14] showed that *saliency* is an effective indicator to simplify meshes while preserving important areas. Their method is based on multi-scale curvature analysis to determine visually *salient* features to which purely intrinsic methods, e.g., QEM simplification [15], may be blind. Second, conveying shapes using simple lines while preserving important features has led Judd et al. [16] to design a new line drawing method extracting the so-called *Apparent Ridges*. These lines depict geometry in a view dependent fashion, capturing curves corresponding to maxima of surface curvature in image space.

Mesh Saliency and Apparent Ridges exploit view- dependent curvature, taking into account both differential properties (intrinsic) and viewer position (extrinsic) to reduce the shape to a minimal set of important features. Therefore, since such metrics have proven to be effective for defining the important surface features conveying the shape, they can also be used for our purpose. We seek for surface locations that would benefit the most from tessellation, i.e. an extra amount of computation and geometric data. So, we propose to use view-dependent curvature to detect them and consequently tailor dynamic tessellation adaptivity.

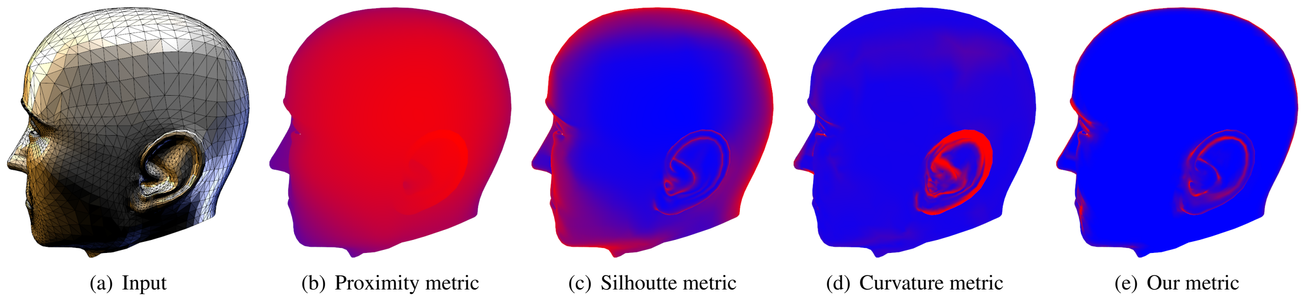

Fig. 3. **Adaptive Tessellation Metrics**: the resulting upsampling ratio is depicted from blue (low) to red (high). (a) Input mesh to tessellate. (b) Proximity is orthogonal to geometric importance. (c) Although silhouette lines contain important features, a large part of the adaptivity is wasted in irrelevant areas. (d) Surface curvature is computed independently of the point of view and cannot detect important low curvature regions observed from a tangential point of view. (e) View-dependent curvature captures important regions stemming from curvature, view-dependent feature or both.

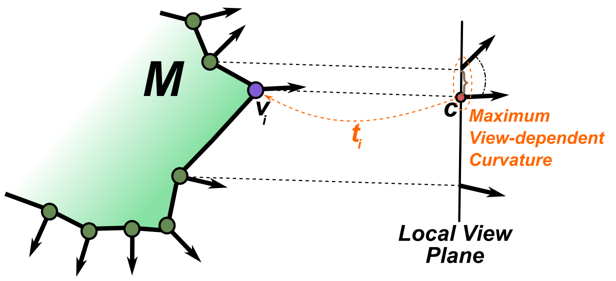

Fig. 4. **View-Dependent Tessellation Metric**: neighboring normal information is projected onto the image plane to compute the maximum variation and deduce a tessellation ratio.

Metric. Intuitively, we want to increase the mesh resolution in areas where the surface variation is faster. This notion of "variation" has to be analyzed from the current point of view, as a given region of the surface may cover a different screen area according to the view position. Thus, we track the curvature variation in image space by projecting each vertex together with its one-ring neighborhood onto the image plane and measure the normal variation on this plane. The requested local tessellation ratio is defined proportionally to this value.

More formally, we denote the input coarse mesh $\mathbf{M} = \{\mathbf{V}, \mathbf{F}\}$ with $\mathbf{V} = \{\mathbf{v}_i\}$ the set of coarse vertices, $\mathbf{v}_i = \{\mathbf{p}_i, \mathbf{n}_i\}$ (position and normal vector at $\mathbf{v}_i$) and $\mathbf{F}$ the list of polygons indexed over $\mathbf{V}$. We define the tessellation ratio t_i at each vertex $\mathbf{v}_i$ as the maximum normal variation in the image plane:

$$t_i = \arg\max_j \{w_{ij} \cdot \langle \mathbf{n}_i \cdot \mathbf{n}_j \rangle\}$$

with $\{\mathbf{n}_j\}$ the set of normal vectors of the one-ring neighborhood of $\mathbf{v}_i$ and $\langle\cdot\rangle$ the dot product (see Fig. 4).

As vertex normal vectors are supplied in any graphics engine, we approximate the view-dependent curvature by computing the divergence of the normal field once projected on the view image plane: the normal term captures the curvature at $\mathbf{v}_i$ in the direction $\overrightarrow{\mathbf{v}_i\mathbf{v}_j}$. Note that we only approximate the surface maximum curvature direction, as we do not evaluate the normal divergence "in-between" one-ring vertices.

The extrinsic influence (i.e. view position) is captured by the weighting term w_{ij}:

$$w_{ij} = \frac{1}{||\mathbf{c} - \pi^{\mathbf{c},\mathbf{u_c}}(\mathbf{p_j})||}$$

with $\mathbf{c}$ the viewer position, $\mathbf{u_c} = (\mathbf{p}_i - \mathbf{c})/||\mathbf{p}_i - \mathbf{c}||$ the (local) view direction and π the orthogonal projection on the so-defined plane $\{\mathbf{c}, \mathbf{u}\}$:

$$\pi^{\mathbf{c},\mathbf{u}}(\mathbf{p}) = \mathbf{p} - \langle(\mathbf{p} - \mathbf{c}) \cdot \mathbf{u}\rangle\mathbf{u}$$

For the sake of genericity, we apply a first order normalization to t_i mapping it onto the range $[0, 1]$, with 0 corresponding to no tessellation and 1 to the maximum tessellation allowed:

$$t_i := \begin{cases} 1 & \text{if } \eta(t_i) > 1 \\ 0 & \text{if } \eta(t_i) < 0, \\ \eta(t_i) & \text{otherwise} \end{cases} \quad \text{with } \eta(x) = \frac{\sigma + x - \mu}{2\sigma}$$

μ being the mean and σ the standard deviation of $\{t_i\}$. Last, we suppose a subsequent mapping function from this value to the adaptivity level range of the tessellator unit (which may still be real for geomorph units).

Figure 3 compares our metric to existing ones. Overall, this metric is able to express both intrinsic geometric features as well as extrinsic view dependent ones. Additionally, it better concentrates high values on features lines, avoiding slow adaptivity variation which often result in a waste of fine vertices. Last, as it only relies on orthogonal projections and surface differentials and it is time-coherent.

4. IMPLEMENTATION AND RESULTS

We have tested our metric using the Adaptive Refinement Kernel [3]. We used several smooth geometric upsampling

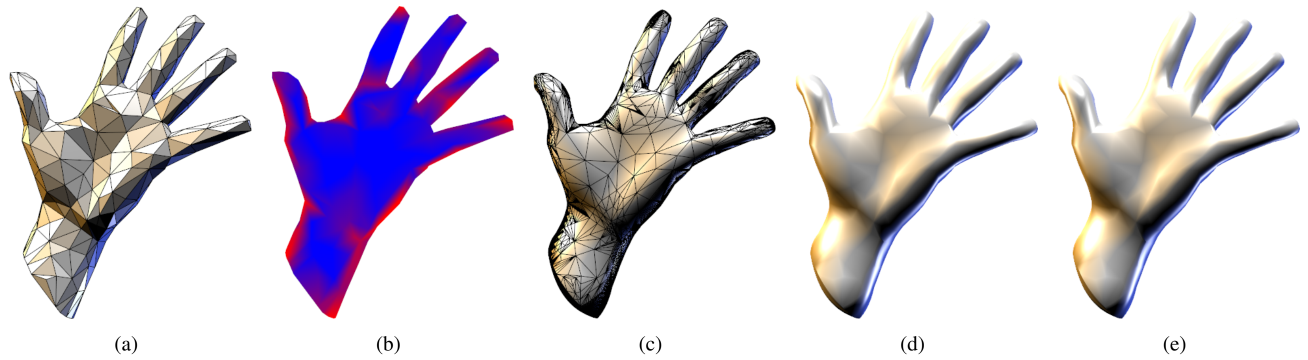

Fig. 5. **Result**. (a) Input (500 tri.). (b) Our metric. (c) The adaptive tessellation produced according to (b). (d) The final rendering using smooth upsamling with adaptive tessellation (5k tri.). (e) With uniform full resolution tessellation (125k tri.).

methods, such as the Curved PN Triangle, Loop Subdivision Surface Approximation and Phong Tessellation. The entire system is implemented in C++, using OpenGL and GLSL with a GeForce GTX 280 graphics device. The input coarse mesh is stored on GPU using 2 textures, one for per-vertex data (position, normals, etc) and the other for the connnectivity information (i.e. triangles). The tessellation ratio is evaluated in a preliminary rendering pass and gathered using a framebuffer object. A second pass is used to normalize these values. Note that a CUDA implementation is straightfoward and we believe this will be as easy to implement on DX11 hardware tessellator units. Alternatively, as input meshes are by definition coarse, we did not observe any bottleneck when computing these values on the CPU. From a qualitative point of view, we performed empirical tests on a variety of coarse models ranging from 50 to 10000 polygons, all upsampled with maximum level 8 (1-for-65k tessellation). As a result, our metric automatically balances between local intrinsic curvature and view dependent feature lines without explicit weighting control. We refer the reader to Cole et al. [17] who assess how view dependent curvature faithfully depicts important geometric features in line drawing. From a quantitative point of view and compared to other metrics, we observed a reduction of the requested number of tessellated polygons ranging from 15 to 75% for similar final rendering quality. Although not as impressive, the framerate improvement was between 5 and 30 percent compared to other metrics, and between 2 and 5 times higher when compared to conservative uniform tessellation (see Fig. 5). In all cases, a higher maximum tessellation ratio makes these values bigger. Note however that performance here is relevant only for the chosen tessellation technique and subject to drastic changes with hardware tessellation.

5. CONCLUSION

Our view-dependent adaptivity metric offers a fair balance between extrinsic and intrinsic properties. It is easy to implement and offers a graceful upsampling distribution around visually important features, as long as the evaluated smoother geometry is correctly approximated by the coarse one, which is usually the case. As future work, we plane to pursue the evaluation of this metric, validating it with a perceptual study.

6. REFERENCES

[1] Tamy Boubekeur and Christophe Schlick, "Generic mesh refinement on GPU," in *Proceedings of ACM SIGGRAPH/Eurographics Graphics Hardware*, 2005, pp. 99–104.

[2] C. Dyken, M. Reimers, and J. Seland, "Real-time GPU silhouette refinement using adaptively blended bezier patches," *Comp. Graph. Forum*, vol. 27, no. 1, pp. 1–12, 2008.

[3] Tamy Boubekeur and Christophe Schlick, "A flexible kernel for adaptive mesh refinement on GPU," *Computer Graphics Forum*, vol. 27, no. 1, pp. 102–114, 2008.

[4] Christopher Dyken, Martin Reimers, and Johan Seland, "Semi-uniform adaptive patch tessellation," *Computer Graphics Forum*, vol. 28, no. 8, 2009.

[5] Le-Jeng Shiue, Ian Jones, and Jorg Peters, "A real-time GPU subdivision kernel," in *SIGGRAPH*, 2005, pp. 1010–1015.

[6] Michael Schwarz and Marc Stamminger, "Fast gpu-based adaptive tessellation with cuda," *Computer Graphics Forum*, vol. 28, no. 2, pp. 365–374, 2009.

[7] A. Klein, "Introduction to the direct3d 11 graphics pipeline.," http://www.microsoft.com/downloads/.

[8] Bui Tuong Phong, *Illumination of Computer-Generated Images*, Ph.D. thesis, University of Utah, 1973.

[9] Alex Vlachos, Jorg Peters, Chas Boyd, and Jason Mitchell, "Curved PN triangles," in *Proceedings of ACM Symposium on Interactive 3D*, 2001, pp. 159–166.

[10] Charle Loop and Scott Schaefer, "Approximating catmull-clark subdivision surfaces with bicubic patches," *ACM Transaction on Graphics*, vol. 27, no. 1, pp. 1–8, 2008.

[11] Tamy Boubekeur and Christophe Schlick, "QAS: Real-time quadratic approximation of subdivision surfaces," in *Proceedings of Pacific Graphics 2007*, November 2007, pp. 453–456.

[12] Tamy Boubekeur and Marc Alexa, "Phong tessellation," *ACM Trans. on Graph. (Proc. SIGGRAPH Asia 2009)*, vol. 27, 2008.

[13] P. Sander, X. Gu, S. Gortler, H. Hoppe, and J. Snyder, "Silhouette clipping," in *ACM SIGGRAPH*, 2000, pp. 327–334.

[14] C. H. Lee, A. Varshney, and David Jacobs, "Mesh saliency," in *ACM SIGGRAPH*, 2005, pp. 659–666.

[15] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *SIGGRAPH*, 1997, pp. 209–216.

[16] Tilke Judd, Frédo Durand, and Edward H. Adelson, "Apparent ridges for line drawing," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 19, 2007.

[17] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz, "Where do people draw lines?," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 88:1–88:11, Aug. 2008.