# Methods for Approximating Loop Subdivision Using Tessellation Enabled GPUs

Ashish Amresh, John Femiani, and Christoph Fünfzig

Department of Engineering, College of Technology and Innovation,
Arizona State University, USA
Laboratoire Electronique, Informatique et Image (LE2I), Université de Dijon, France
{amresh,john.femiani}@asu.edu, c.fuenfzig@gmx.de

**Abstract.** Subdivision surfaces provide a powerful alternative to polygonal rendering. The availability of tessellation supported hardware presents an opportunity to develop algorithms that can render subdivision surfaces in realtime. We discuss the performance of approximating Loop Subdivision surfaces using tessellation-enabled GPUs in terms of speed and quality of rendering for these methods as well as the implementation strategy. We also propose a novel one pass unified rendering setup for all three methods. Subdivision using the Loop method supports arbitrary triangle meshes and provides for easy transition from polygonal rendering of triangles to the parametric domain. Majority of graphics software applications, especially game engines, render polygons as triangles. The objectives of this paper are to evaluate the performance of smooth rendering algorithms developed to take advantage of tessellator enabled GPUs, provide an easy transition from polygonal to parametric rendering and propose an optimal way to achieve multi-level rendering dependent on performance and visual needs of the application.

## 1 Introduction

Fast rendering of polygons has been an established process for over a decade and it has been primarily driven by the processing power of the GPU. The common graphics libraries DirectX and OpenGL expose this power and developers reconfigure their applications to deliver increased realism by implementing rendering techniques [1–3] like normal mapping, bloom lighting, shadows, and animation techniques [4, 5] like rigging for character and facial animation. In most cases the polygons are rendered as triangles and animation is performed using a skeleton, that transforms the vertices of the triangle in realtime. With the addition of a hardware tessellator unit to the graphics pipeline in Direct3D11 and OpenGL 4.0, the asset production pipeline can be simplified to a greater detail by having the content creators model both the cut-scenes and application level assets using subdivision surfaces. The tessellator unit adds two new programmable stages , the hull and domain in Direct3D11 and control and evaluation in OpenGL 4.0, to the graphics pipeline. The methods described in this paper use OpenGL 4.0 and our language reflects this usage. Rendering parametric surfaces using tessellation enabled GPUs as been discussed in detail in both Direct3D and OpenGL developer documentation. For the purposes of this paper we will be looking at various methods for approximate rendering of the Loop subdivision surface [6]. The main factors that limit the performance of the GPU is the number of control points in the input mesh and number of calculations (shader lines of code) performed in the control shader. This is mainly because the GPU is bound by its memory bandwidth and its speed is directly dependent on the number of memory fetches. Our research, presented in Section 5, indicates that algorithms that use many instructions to derive the control points in the control shader can severely limit performance even though their initial memory foot print is low. We therefore choose three methods based on these observations. The Point-Normal (PN) method [7] has 6 control points and 13 operations in the control shader, the Walton-Meek (WM) method [8] has 6 control points and 70 operations in the control shader and our proposed Gregory method has 15 control points and 0 operations in the control shader. A new method for approximating Loop subdivision surfaces with triangular Gregory patches is proposed. The methods are implemented using hardware tessellation. We render various objects using the three methods and describe the performance in terms of their visual quality and speed. We find that the speed is also dependent on the number of models in the scene invoking a particular method and therefore the methods need to be carefully chosen in order to optimize the scene. For this purpose we propose an unified rendering framework, that can combine all three methods at run time. Our goal is to provide application developers the information necessary for migrating from polygonal domain to smooth rendering of triangles. By applying the unified rendering framework, described in Section 4, developers can perform automatic level-of-detail(LOD) calculations for their meshes. The number of control points in the input mesh then becomes synonymous with the LOD level. This gives the ability to have a two step system for LOD, one computes the control points of the input mesh and the other computes the edge tessellation factors.

## 2 Background

The idea of rendering parametric triangular surfaces in realtime dates back to the introduction of PN-triangles [7], developed to improve the surface quality of low polygonal meshes. Quadratic approximation surface (QAS) [9, 10] used the idea of rendering a PN-patch for limit Loop points and normals. With PNG1-triangles [11] this approach was further improved to satisfy $G^1$ continuity across each edge. Similarly [8] provides a method for $G^1$ continuous surface when the boundary curves are known. The idea of approximating Catmull-Clark subdivision surfaces using bi-cubic Bézier patches was introduced in [12] and then further refined in [13] for hardware tessellation using Gregory Patches. These methods are suitable for quad-dominant meshes and do not work well for triangular patches. The above research forms the primary motivation for developing a list of methods for rendering an approximate Loop subdivision surface by varying the number of control points for the input mesh. The Loop subdivision scheme is a face split based on triangular splines [14] and at every subdivision step it calculates a new vertex for each existing one and a new vertex for each edge. We can calculate the limit point and limit tangents for the Loop control mesh directly as shown in [15]. A parametric triangular Bézier patch is defined in [16] and a triangular Gregory patch is defined in [17, 18] and is a modified form of a quartic Bezier patch where the boundary curves are cubic and the interior surface is quartic.However, the inner control points of the patch are dependent on the $(u, v, w = 1-u-v)$ parameter values of the domain. This leads to the formulation of unique inner points for each parameter value in the domain. This construction was introduced by Gregory to ensure that a pair of patches meeting at a shared edge are tangent plane continuous across that edge. The patch is evaluated by the following equation:

$$\begin{aligned} T(u,v,w) = {} & u^3\boldsymbol{p}_0 + v^3\boldsymbol{p}_1 + w^3\boldsymbol{p}_2 + \\ & 3uv(u+v)(u\boldsymbol{e}_0^+ + v\boldsymbol{e}_1^-) + \\ & 3vw(v+w)(v\boldsymbol{e}_1^+ + w\boldsymbol{e}_2^-) + \\ & 3wu(w+u)(w\boldsymbol{e}_2^+ + u\boldsymbol{e}_0^-) + \\ & 12uvw(u\boldsymbol{F}_0 + v\boldsymbol{F}_1 + w\boldsymbol{F}_2) \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{F}_0 &= \frac{w\boldsymbol{f}_0^- + v\boldsymbol{f}_0^+}{v+w}, \\ \boldsymbol{F}_1 &= \frac{u\boldsymbol{f}_1^- + w\boldsymbol{f}_1^+}{w+u}, \\ \boldsymbol{F}_2 &= \frac{v\boldsymbol{f}_2^- + u\boldsymbol{f}_2^+}{u+v} \end{aligned} \tag{1}$$

Figure 1 shows the labeling of control points for the Gregory patch, and how it maps to a quartic Bézier triangle.

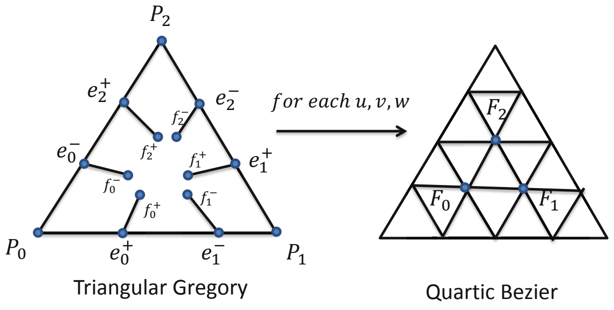


**Fig. 1.** Control points for triangular Gregory and its relation with quartic Bézier

## 3 Approximation Methods

In this section we will illustrate how to approximate the Loop subdivision surface by using the points, normals and tangents obtained in Section 2. We start by introducing methods that require less input control points and then move on to those that require more. We also discuss implementation details for the control shader and provide performance trade-offs.

### 3.1 PN-Triangles

Vlachos et al. [7] propose *curved PN triangles* for interpolating a triangle mesh by a parametric, piecewise cubic surface. This established technique generates a $C^0$-continuous surface which stays close to the triangle mesh. At first, the PN scheme places the intermediate control points $\bar{b}_{ijk}$ at the positions $(ib_{300}+jb_{030}+kb_{003})/3$, $i+j+k=3$, leaving the three corner points unchanged. Then, each $b_{ijk}$ on the border is constructed by projecting the intermediate control point $\bar{b}_{ijk}$ into the plane defined by the nearest corner point and its normal. Finally, the central control point $b_{111}$ is constructed by moving the point $\bar{b}_{111}$ halfway in the direction $m-\bar{b}_{111}$ i.e ($b_{111}=0.5*(m+\bar{b}_{111})$) where $m$ is the average of the six control points computed on the borders as described above. The construction uses only the three triangle vertices and its normals. This makes it especially suitable for a triangle rendering pipeline. By using the Loop limit points and normals derived in [15], we get a Loop approximation using PN-triangles.

### 3.2 Walton-Meek Triangles

Walton and Meek [8] proposed the WM method for achieving $G^1$ patch from the boundary curves of a triangular control net. Both the WM method and the Gregory Method, described in the next section, construct a triangular Gregory patch to evaluate the surface. The boundary curves are obtained in the same manner as PN-triangles and the inner points are calculated for each $u, v, w$, this ensures that any two neighboring triangles have a common tangent plane along their shared boundary. The main difference between the WM and Gregory method is based on where and how the calculations for the tangent plane continuity occur, the WM performs this inside the control shader while the Gregory processes this information and sends it to the control shader. We pass the limit Loop points (3) and normals (3)a total of 6 control points for this method. The WM method constructs the inner points based only on the boundary control points as the 1-ring neighborhood is not available inside the control shader. A simple explanation of the WM method is provided below, for details refer to [8]:

– Degree elevate the cubic boundary curve, formed by the limit Loop points and normals, to a quartic and get the control points of the curve.
– For each boundary curve, determine the two interior control points associated with this curve using only the data available on the boundary. Altogether, the 6 interior Gregory control points result.

– For each $u$,$v$,$w$, evaluate the patch by blending the the six Gregory points and obtaining the interior control points of the quartic patch as shown in Equation (1).

### 3.3 Gregory Triangles

As shown in Figure 1, we need to calculate 15 control points for evaluating a triangular Gregory patch. For each $u$, $v$, $w$ parameter value the boundary control points are degree elevated from cubic to quartic using the two corner and two edge points and the the inner quartic points $\boldsymbol{F}_0, \boldsymbol{F}_1, \boldsymbol{F}_2$ are calculated from the inner six Gregory control points as shown in Equation (1). So an approximation of the Loop surface is possible by calculating these 15 control points, 3 corner, 6 edge and 6 inner for the Gregory triangle.

The corner points are set as the limit loop control points and the edge points are calculated using the limit loop tangents, however we need to choose the right length for these tangents to get the edge points and we know that the derivative at the end points of a Gregory patch is $3(\boldsymbol{e}_0^+ - \boldsymbol{p}_0)$ we can solve $\boldsymbol{e}_0^+$ by $\boldsymbol{e}_0^+$ by

$$\boldsymbol{e}_0^+ = \boldsymbol{p}_0 + \frac{2}{3}\boldsymbol{t}_{0,1}\lambda \tag{2}$$

where $\boldsymbol{t}_{0,1}$ is the Loop limit tangent from $\boldsymbol{p}_0$ to $\boldsymbol{p}_1$ and $\lambda$ is chosen to be the subdominant eigenvalue of the Loop surface and is given by

$$\lambda = \frac{3}{8} + \frac{1}{4}\cos\left(\frac{2\pi}{n}\right) \tag{3}$$

We now have to construct the inner points so that the surface is tangent plane continuous across the edge. By using the method described in [13], we find that setting the traversal vector $r$ to be equal to the Loop surface cross tangents, see Figure 2, provides the best results. The traversal vector $r$ is calculated by performing two levels of Loop subdivision on the original control mesh and evaluating the cross tangents at $1/4, 1/2, 3/4$ along each edge and then linearly interpolating them to find the values at $1/3, 2/3$.

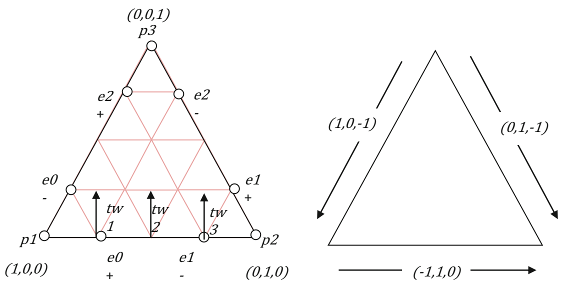


**Fig. 2.** Loop Cross tangents at 1/4, 1/2 and 1/3 shown by tw1, tw2 and tw3

The equation for the inner points is given by

$$\boldsymbol{f}_0^+ = \frac{1}{4}\left(c_1\boldsymbol{p}_0 + (4 - 2c_0 - c_1)\,\boldsymbol{e}_0^+ + 2c_0\boldsymbol{e}_1^- + r\right) \tag{4}$$

where $c_0 = \frac{2\pi}{n_0}$ and $c_1 = \frac{2\pi}{n_1}$, $n_0$ and $n_1$ are the valence at $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$

## 4 Unified Rendering Framework

We propose a unified rendering framework that involves three stages, preprocessing, CPU switching and rendering.During the preprocessing stage all the control point calculations for each triangle in the input mesh is done irrespective of the method chosen. Therefore for each triangle, 3 limit points, 3 limit normals, 6 edge points and 6 inner points, resulting in a total of 15 control points and 3 normals are calculated as shown in Figure 3. The application can then dynamically change this data at run time on a case by case basis, giving the developer an automatic one-pass algorithm to switch the methods at run time to manage quality vs. performance or connect it into existing LOD structures. As shown in Figure 4, there are three main stages for the unified rendering framework. In the first stage, control polygon from asset production packages like Maya is read by a custom patch generation tool that creates for each base mesh triangle, the 18 control points. In the second stage the application creates custom draw calls for all the three methods and performs CPU based decisions to choose the appropriate method. The last stage involves creating the effect file using the CGFX file format with the methods implemented as separate techniques. Control and Evaluation shaders need to be written and we provide CGFX files that implement the methods discussed.

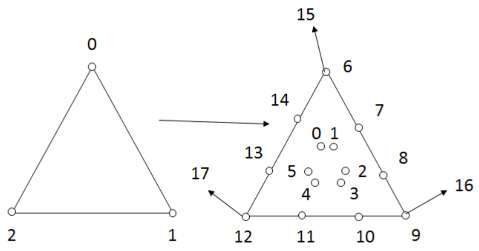


**Fig. 3.** Indexing of the 18 control points calculated for every input triangle

### 4.1 Control and Evaluation Shader

The control shader takes the input control mesh based on the method chosen, and performs tessellation calculations to determine each edge tessellation factor. Also depending on the case it can perform other calculations before sending the

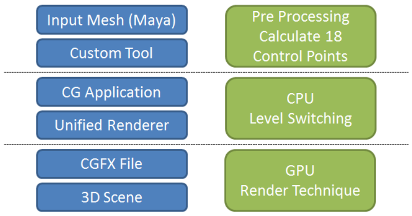


**Fig. 4.** Various stages of the unified rendering framework

output to the evaluation shader. For example, in the PN method the 10 control points for the cubic Bézier triangle and the 6 normals for quadratic interpolation are calculated in the control shader.

The Evaluation shader receives the $u, v, w$ values from the tessellator and the output from the control shader and calculates the surface point and normal at that parameter set. The PN method evaluates the point using [15] and the normal by quadratic interpolation. The WM method evaluates the point by Equation (1), and the normal by quadratic interpolation. The Gregory method transforms Equation (1) into a quartic Bézier by degree elevating the boundary cubic curves and calculating $F0, F1, F2$ as described in Section 3. This allows for simultaneous calculation of the point and normal using the de Casteljau algorithm. Table 1 shows the the number of arithmetic calculations (shader lines of code) in the control and evaluation shaders for the three methods.

**Table 1.** Shader properties for the three methods

| **Method Name** | **Control Shader** | **Evaluation Shader** | **Control Points** |
|---|---|---|---|
| PN | 13 | 10 | 6 |
| WM | 70 | 10 | 6 |
| Gregory | 0 | 20 | 15 |

**Table 2.** Frame rate comparison for the three methods

| **Model Name** | **Base Mesh Tris/Verts** | **Tess Level/ Object Count** | **PN FPS** | **WM FPS** | **Gregory FPS** |
|---|---|---|---|---|---|
| Face | 102/200 | 5/1 | 1850 | 1180 | 1780 |
| Bunny | 502/1000 | 5/1 | 1297 | 822 | 1190 |
| Big Guy | 1754/2900 | 5/1 | 940 | 635 | 870 |
| Face | 102/200 | 5/50 | 479 | 379 | 266 |
| Bunny | 502/1000 | 5/50 | 63 | 52 | 32 |
| Big Guy | 1754/2900 | 5/50 | 53 | 39 | 22 |

**Table 3.** Geometric error reported by Metro for the three methods

| **Method Name** | **Max error** | **mean error** | **RMS error** |
|---|---|---|---|
| PN | 0.5618 | 0.019 | 0.030 |
| WM | 0.3913 | 0.013 | 0.022 |
| Gregory | 0.1746 | 0.009 | 0.013 |

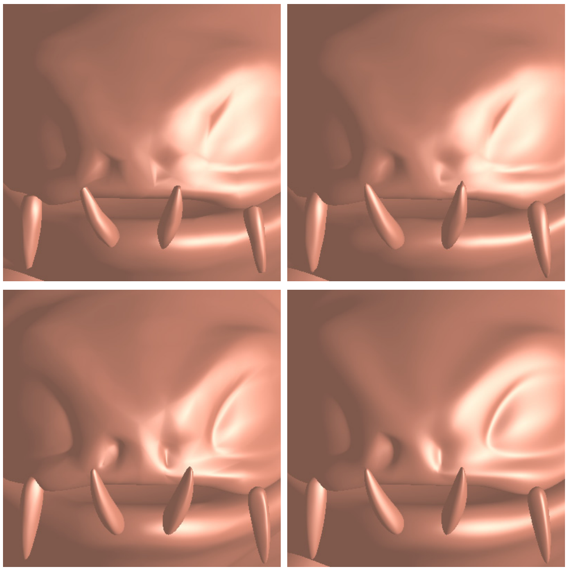


**Fig. 5.** Monster frog rendered using PN (top-left), WM (top-right), Gregory (bottom-left) and Original Loop (bottom-right) methods

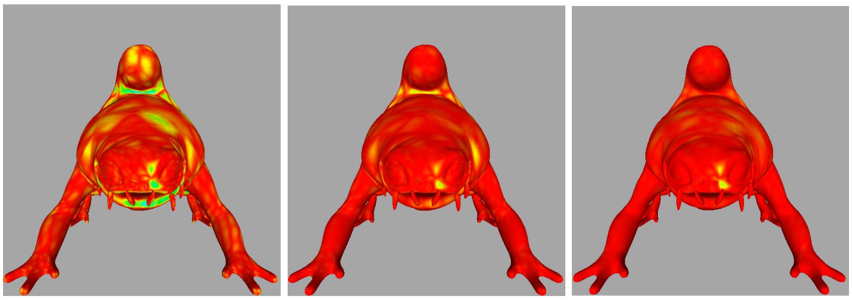


**Fig. 6.** Geometric approximation error calculated by Metro for PN, WM and Gregory methods

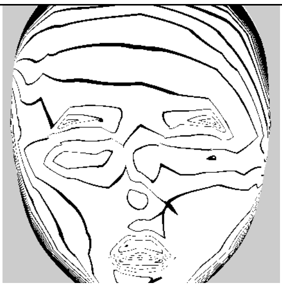
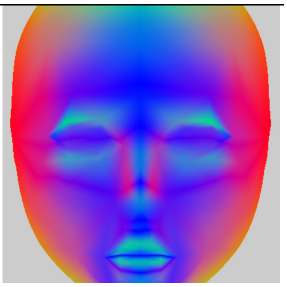
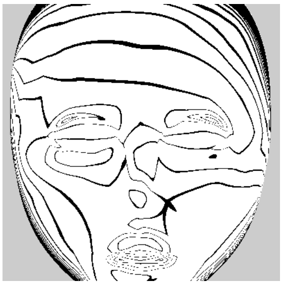
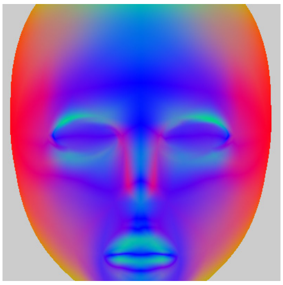
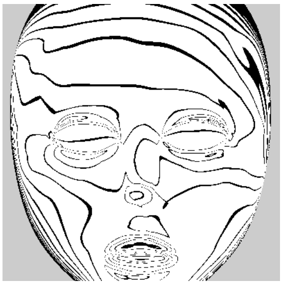
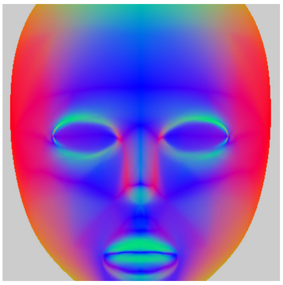
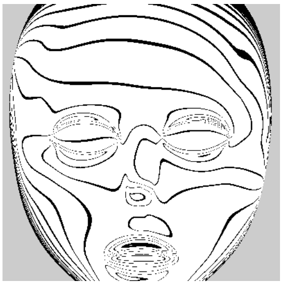
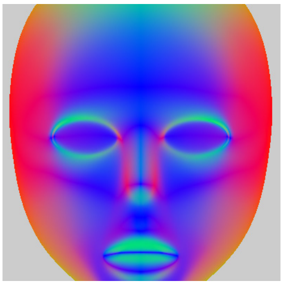
| Method | Reflection Lines | Normals |
|---|---|---|
| PN |  |  |
| WM |  |  |
| Gregory |  |  |
| Loop |  |  |

**Fig. 7.** Reflections lines (left) and normals (right) for the Face model

## 5 Results

In this section we measure the speed and quality of the surface for the three methods described in this paper. With all conditions being the same we test the frame rate at various tessellation levels. We also render the reflection lines as shown in Figure 7 for each method to test the quality of the surface. Our tests were performed on PC with Windows 7 32bit and NVIDIA Quadro5000 graphics card. Table 2, shows the frame rate comparisons for the face, bunny and big guy models at tessellation level 5 and also compares the numbers by changing the number of models drawn from 1 to 50. It is seen that PN performs best under all situations and Gregory outperforms the WM method only when few models use it. This confirms that performance depends on the number of operations in the hull shader and the number of control points that need to be stored in video memory for a base mesh triangle. Figure 5 shows the monster frog model rendered using the three methods, PN, WM, Gregory and the original Loop method. Table 3 shows the geometric error calculated by Metro [19] by comparing the triangle meshes generated by Loop subdivision to the ones generated by the three methods. Figure 6 shows this error mapped to vertex colors and we can observe that except at extreme extraordinary points (valence greater than 7) the error produced by the Gregory method is minimal. To compensate for this limitation the original control mesh would need to avoid having such points at model time.

## 6 Conclusion

In this paper we have presented three methods for approximating Loop subdivision surfaces using tessellation enabled hardware. Developers can unify the asset production pipeline and provide automatic switching between these methods at runtime. The maximum number of control points required per patch is 18, the PN and WM methods use a subset of 6 while the Gregory uses a subset of 15. Our observation leads to the following conclusions. The Gregory method is best suited for characters and fluid assets that incorporate complex animations as seen in game characters, the WM method is best suited for dynamic objects with simple animations used in weapons, vehicles and breakable objects, while the PN method works best for static objects like trees, environments and terrain. This observation stems from the information in Table 2 based on the performance of the methods at single and multiple instances as well as at lower and higher tessellations. The unified rendering setup, shown for triangles, can similarly be applied for quads using the Catmull-Clark subdivision method. In future work, we plan to improve the unified rendering setup to include meshes consisting of arbitrary polygons and not just triangles.

## References

1. Luebke, D., Humphreys, G.: How gpus work. Computer 40, 96–100 (2007)
2. Kautz, J.: Hardware lighting and shading: A survey. In: Computer Graphics Forum, vol. 23, pp. 85–112. Wiley Online Library (2004)
3. Akenine-Moller, T., Haines, E.: Real-time rendering. AK (2002)
4. Kry, P., James, D., Pai, D.: Eigenskin: real time large deformation character skinning in hardware. In: Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 153–159. ACM (2002)

5. Baran, I., Popović, J.: Automatic rigging and animation of 3d characters. In: ACM SIGGRAPH Papers, p. 72. ACM (2007)
6. Loop, C.: Smooth Subdivision Surfaces Based on Triangles. Master's Thesis, University of Utah 1, 1–74 (1987)
7. Vlachos, A., Peters, J., Boyd, C., Mitchell, J.: Curved PN triangles. In: Proceedings of the 2001 Symposium on Interactive 3D Graphics, pp. 159–166. ACM (2001)
8. Walton, D.J., Meek, D.S.: A triangular G1 patch from boundary curves. Computer-Aided Design 28, 113–123 (1996)
9. Boubekeur, T., Schlick, C.: Approximation of subdivision surfaces for interactive applications. In: ACM Siggraph Sketch Program (2007)
10. Boubekeur, T., Schlick, C.: Qas: Real-time quadratic approximation of subdivision surfaces (2007)
11. Fünfzig, C., Müller, K., Hansford, D., Farin, G.: PNG1 triangles for tangent plane continuous surfaces on the GPU. In: GI 2008: Proceedings of Graphics Interface 2008, pp. 219–226. Canadian Information Processing Society, Toronto (2008)
12. Loop, C., Schaefer, S.: Approximating Catmull-Clark subdivision surfaces with bicubic patches. ACM Transactions on Graphics (TOG) 27, 1–11 (2008)
13. Loop, C., Schaefer, S., Ni, T., Castaño, I.: Approximating subdivision surfaces with gregory patches for hardware tessellation. ACM Transactions on Graphics (TOG) 28, 1–9 (2009)
14. Seidel, H.: Polar forms and triangular B-spline surfaces. Computing in Euclidean Geometry, 235–286 (1992)
15. Li, G., Ren, C., Zhang, J., Ma, W.: Approximation of Loop Subdivision Surfaces for Fast Rendering. IEEE Transactions on Visualization and Computer Graphics (2010)
16. Farin, G.: Curves and Surfaces for Computer-Aided Geometric Design — A Practical Guide, 5th edn. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, 499 pages. Morgan Kaufmann Publishers, Academic Press (2002)
17. Gregory, J.A.: Smooth interpolation without twist constraints. Computer Aided Geometric Design, 71–87 (1974)
18. Chiyokura, H., Takamura, T., Konno, K., Harada, T.: G1 surface interpolation over irregular meshes with rational curves. NURBS for Curve and Surface Design, 15–34 (1990)
19. Cignoni, P., Rocchini, C., Scopigno, R.: Metro: measuring error on simplified surfaces. In: Computer Graphics Forum, vol. 17, pp. 167–174. Wiley Online Library (1998)