Semi-uniform, 2-Different Tessellation of Triangular Parametric Surfaces

CONFERENCE PAPER · NOVEMBER 2010

DOI: 10.1007/978-3-642-17289-2_6 · Source: DBLP

2 AUTHORS:

Ashish Amresh

Arizona State University

21 PUBLICATIONS **39** CITATIONS

SEE PROFILE

Christoph Fünfzig

31 PUBLICATIONS **132** CITATIONS

SEE PROFILE

Semi-Uniform, 2-Different Tessellation of Triangular Parametric Surfaces

Ashish Amresh, Christoph Fünfzig

School of Computing, Informatics and DS Engineering, Arizona State University, USA
Laboratoire Electronique, Informatique et Image (LE2I), Université de Dijon, France
amresh@asu.edu, c.fuenfzig@gmx.de

Abstract. With a greater number of real-time graphics applications moving over to parametric surfaces from the polygonal domain, there is an inherent need to address various rendering bottlenecks that could hamper the move. Scaling the polygon count over various hardware platforms becomes an important factor. Much control is needed over the tessellation levels, either imposed by the hardware limitations or by the application. Developers like to create applications that run on various platforms without having to switch between polygonal and parametric versions to satisfy the limitations. In this paper, we present SD-2 (Semi-uniform, 2-Different), an adaptive tessellation algorithm for triangular parametric surfaces. The algorithm produces well distributed and semi-uniformly shaped triangles as a result of the tessellation. The SD-2 pattern requires new approaches for determining the edge tessellation factors, which can be fractional and change continuously depending on view parameters. The factors are then used to steer the tessellation of the parametric surface into a collection of triangle strips in a single pass. We compare the tessellation results in terms of GPU performance and surface quality by implementing SD-2 on PN patches.

1 Introduction

Rendering of parametric surfaces has a long history [1]. Early approaches consist of direct scanline rasterization of the parametric surface [2]. Modern GPUs have the ability to transform, light, rasterize surfaces composed of primitives like triangles and more recently also to tessellate them. Tessellation in particular can save bus bandwidth for rendering complex smooth surfaces on the GPU.

The proposed triangle tessellation approaches can be distinguished either by the hardware stage they employ or by the geometric tessellation pattern. Approaches originating from triangular subdivision surfaces use a 1-to-4 split of the triangle, as seen in [3] and [4]. For greater flexibility of the refinement, factors are assigned to the edges of each triangle, which results in three different tessellation factors for an arbitrary triangle. Tessellation patterns for this general case go back to several authors after [5], for example, in [6] optimized for a hardware implementation, and in [7] for a fast CUDA implementation. Fractional tessellation factors target the discontinuity problems with integer factors during

animations, whose abrupt changes are visible in the shading and other interpolated attributes. With fractional tessellation factors and a tessellation pattern continuously depending on them, the changes are less abrupt with the view.

The interior patch is tessellated uniformly with one of the tessellation factors and the two other edges have to be connected accordingly. This can be achieved by a gap-filling strip of certain width, see Figure 1. We call this process as *stitching* the gaps.

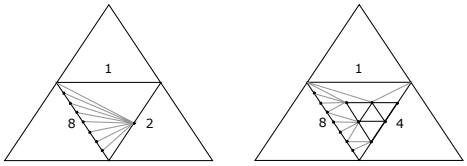

Fig. 1. Gap-filling strips connect the two border curves resulting from different tessellation factors [7].

Shading artifacts can be visible as the tessellation is strictly regular in the interior part and highly irregular in the strips. This can be mitigated by choosing not too different tessellation factors. Also the recently presented DirectX 11 graphics system [8] contains a tessellation stage with a tessellator for triangular and quadrangular domains, steered by edge and interior fractional tessellation factors. The tessellator's architecture is proprietary though.

Non-uniform, fractional tessellation [9] adds reverse projection to the evaluation of the tessellation pattern to make it non-uniform in parameter-space but more uniform in screen-space.

In a recent work, [10] keeps a uniform tessellation pattern and snaps missing vertices on the border curve to one of the existing vertices at the smaller tessellation factor. This is a systematic way to implement gap-filling but it is restricted to power-of-two tessellation factors and 1-to-4 splits of the parameter domain. Figure 2 gives an illustration of the uniform tessellations with snapped points in black. For largely different tessellation factors, the resulting pattern can be quite irregular.

In this work, we restrict the edge tessellation factors in such a way that only two different tessellation factors can occur in each triangle. For such cases, a tessellation pattern can be used, which is much simpler and more regular than in the three different case (Section 3). The tessellation code can output the resulting triangles as triangle strips, which makes this approach also suitable for an inside hardware implementation. We will demonstrate this advantage by implementing the pattern in a geometry shader (Section 3.3). In Section 3.2, we analyze which assignments are possible with two different factors and show that many important cases are contained, i.e., factors based on the distance to the camera eye plane and to the silhouette plane. We show visual results and

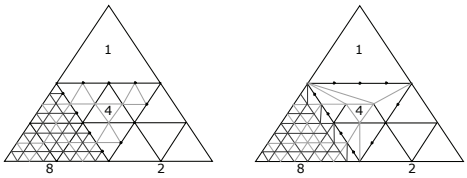

Fig. 2. Semi-uniform tessellation by snapping edges of larger tessellation factor (black points) to the nearest point of the smaller tessellation factor [10].

GPU metrics obtained with our implementation in Section 4. Finally, we give conclusions and future work in Section 5.

2 Locally Defined Triangular Parametric Surfaces

Vlachos et al. [11] 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 and thus avoids self-interference.

A parametric triangular patch in Bézier form is defined by

$$p(u,v,w) = \sum_{i+j+k=n} \frac{n!}{i!j!k!} u^i v^j w^k b_{ijk} \tag{1}$$

where b_{ijk} are the control points of the Bézier triangle and $(u, v, w = 1-u-v)$ are the barycentric coordinates with respect to the triangle [12]. Setting $n = 3$ gives a cubic Bézier triangle as used in the following construction.

At first, the PN scheme places the intermediate control points $\overline{b}_{ijk}$ 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 $\overline{b}_{ijk}$ into the plane defined by the nearest corner point and its normal.

Finally, the central control point b_{111} is constructed moving the point $\overline{b}_{111}$ halfway in the direction $m - \overline{b}_{111}$ where m is the average of the six control points computed on the borders as described above. The construction uses only data local to each triangle: the three triangle vertices and its normals. This makes it especially suitable for a triangle rendering pipeline.

3 Semi-Uniform, 2-Different Tessellation

Semi-uniform 2-Different (SD-2) is our proposed pattern for adaptive tessellation where the tessellation factors on the three edges of a triangle are either all same or only two different values occur. As parametric triangular surfaces are composed of patches, it is necessary to do a tessellation of patches. In order to

change the tessellation based on various criteria, the computation of suitable tessellation factors and an adaptive tessellation pattern for them is necessary. In the following sections, we describe both these components.

3.1 Edge Tessellation Factors based on Vertex/Edge Criteria

For adaptive tessellation of a triangular parametric, normally tessellation factors are computed per vertex or per edge. If computed per vertex then they are propagated to tessellation factors per edge. Given the three edge tessellation factors (f_u, f_v, f_w), the subdivisions of the three border curves into line segments are given by $p(0, i/f_u, (f_u - i)/f_u)$, $i = 0 \dots \lfloor f_u \rfloor$, $p(j/f_v, 0, (f_v - j)/f_v)$, $j = 0 \dots \lfloor f_v \rfloor$, and $p(k/f_w, (f_w - k)/f_w, 0)$, $k = 0 \dots \lfloor f_w \rfloor$. A gap-free connection to the mesh neighbors is guaranteed by a tessellation that incorporates these conforming border curves.

An assignment of edge tessellation factors can be based on criteria like vertex distance to the camera eye plane, edge silhouette property, and/or curvature approximations using normal cones [4].

3.2 Edge Tessellation Factors for the 2-Different Case

For our following tessellation pattern, we need that in the edge tessellation factor triple (f_u^*, f_v^*, f_w^*) only two different values $f_u^* = f_v^*$ and f_w^* occur.

Approximation of an arbitrary tessellation factor assignment (f_u, f_v, f_w), $f_u \neq f_v$, $f_v \neq f_w$, $f_u \neq f_w$ by a 2-different one (f_u^*, f_v^*, f_w^*) is a non-local problem. Therefore, we avoid the general case and guarantee that the tessellation factor calculation never produces 3-different factors for the edges of a triangle. Then SD-2 can be used for a faster and simpler tessellation.

Let $d : V(M) \to \mathbb{R}$ be a *level function* on the vertices of the mesh, which means d is always monotone increasing on shortest paths $\{v_0 = x \in I, \dots, v_l = y\}$ from a vertex $x \in I$ to a vertex y: $d(v_{i-1}) \leq d(v_i)$, $i = 1, \dots, l$, where $I = \{x \in V(M) : \forall y \ d(x) \leq d(y)\}$ is the set of minimum elements. Given a level function d, it is easy to derive a tessellation factor assignment f^*, which is only 2-different, as follows $f^*(\{s, e\}) := g(\min\{d(s), d(e)\})$ with a normally monotone, scalar function g. Note that the level function is only used to impose an order on the mesh vertices, which is easy to compute based on the vertex coordinates.

We give examples of tessellation factor assignments below, which are constructed with the help of a level function as described.

Distance from the camera eye plane. The smallest distance d_p to a plane, for example, the camera eye plane, naturally is a level function, as defined above. A semi-uniform edge tessellation factor assignment (f_u^*, f_v^*, f_w^*) for a triangle then is $f_{\text{edge}}^* := g(\min\{d(s_{\text{edge}}), d(e_{\text{edge}})\})$ with a linear function $g_1(d) := f_{\max} \frac{d_{\max} - d}{d_{\max} - d_{\min}} + f_{\min} \frac{d - d_{\min}}{d_{\max} - d_{\min}}$ or a quadratic function $g_2(d) := \frac{f_{\max} - f_{\min}}{(d_{\min} - d_{\max})^2}(d - d_{\max})^2 + f_{\min}$ mapping the scene's depth range $[d_{\min}, d_{\max}]$ to decreasing tessellation factors in the range $[f_{\min}, f_{\max}]$.

Silhouette refinement. Silhouette classification is usually done based on a classification of the vertices into front-facing ($n^t(e-v) \geq 0$) and back-facing ($n^t(e-v) < 0$) using the vertex coordinates v, vertex normal n and the camera eye point e. The distance function d_{silh} to the silhouette plane is a level function as defined above. But the function $n^t(e-v)$ is easier to compute by just using the vertices and vertex normals of the mesh. It is not a level function though, but an edge is crossed by the silhouette plane in case the two incident vertices are differently classified. Each triangle can have exactly 0 or 2 such edges. An edge tessellation factor assignment with just two values, a for an edge not crossed by the silhouette, and b for an edge crossed by the silhouette, can be used to refine the silhouette line and present full geometric detail at the silhouette.

The edge tessellation factor assignments based on a level function can be directly computed inside a GPU shader. We show examples of this in Section 4. On the contrary, the edge tessellation factors obtained by a curvature approximation in the patch vertices can not be made 2-different for arbitrary meshes easily. Computing an approximation is possible though on the CPU.

3.3 Adaptive Tessellation Pattern with 2-Different Factors

In case of triangles with only two different edge tessellation factors $f_u = f_v$ and f_w, it is possible to tessellate in an especially simple way. Our tessellation pattern is composed of a bundle of parallel lines $w = i/f_u := w_i$, $i = 0, \ldots, \lfloor f_u \rfloor$, which are intersected by a second bundle of radial lines from the tip vertex $(0, 0, 1)$ to $((f_w - j)/f_w, j/f_w, 0)$, $j = 0, \ldots, \lfloor f_w \rfloor$. The intersections with the parallel line i are in the points $((1 - w_i)(f_w - j)/f_w, (1 - w_i)j/f_w, w_i)$, $j = 0, \ldots, \lfloor f_w \rfloor$. This pattern is very flexible as it works also with fractional factors $f_u = f_v$ and f_w. In the fractional case, the remainders $(f_u - \lfloor f_u \rfloor)/f_u$ and $(f_w - \lfloor f_w \rfloor)/f_w$ can be added as additional segments. In case the subdivision is symmetric to the mid-edge, it can be generated in arbitrary direction. We achieve this by shrinking the first segment and augmenting the last segment by the half fractional remainder $0.5(1/f_u - (f_u - \lfloor f_u \rfloor)/f_u)$ and $0.5(1/f_w - (f_w - \lfloor f_w \rfloor)/f_w)$ respectively. Otherwise, it has to be generated in a unique direction, for example from the nearest to the farthest vertex, which complicates things a lot. Concerning the distribution of lines, it is also possible to place them non-uniformly by a reverse projection according to [9]: $u' = \frac{u/z_1}{(1-u-v)/z_0 + u/z_1 + v/z_2}$, $w' = \frac{w/z_0}{w/z_0 + u/z_1 + (1-u-w)/z_2}$ where z_0, z_1, z_2 are the vertex depths of the triangle.

Figure 3 shows an example of the construction for fractional values $f_u = f_v = 2.5$, $f_w = 2.5$, and for integer factors $f_u = f_v = 3$, $f_w = 3$ and $f_u = f_v = 3$, $f_w = 5$.

It is possible to output all triangles between two adjacent radial lines or two adjacent parallel lines as a triangle strip, which reduces vertex repetition considerably and is beneficial on some hardware architectures. This property becomes a great advantage at the silhouettes where the triangles have edge tessellation factors ($f_u = f_v \gg f_w$) or ($f_u = f_v \ll f_w$) and the tessellation can be emitted with a minimum number of strips. For edge tessellation factors ($f_u =$

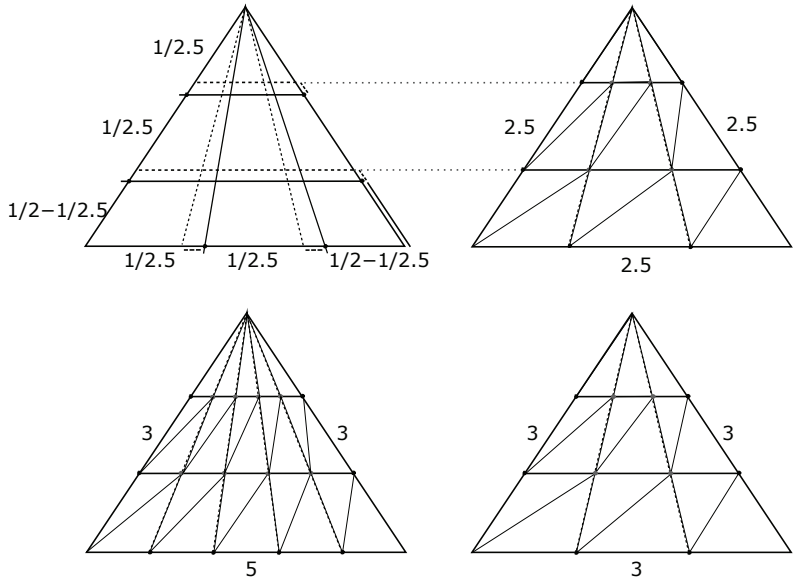

Fig. 3. Tessellation pattern for 2-different factors, which is composed of a bundle of parallel lines, intersected by a second bundle of radial lines towards the tip vertex. In the top row, for a fractional example: $f_u = f_v = 2.5$, $f_w = 2.5$; in the bottom row, for integer examples: $f_u = f_v = 3$, $f_w = 3$ and $f_u = f_v = 3$, $f_w = 5$.

f_v, f_w), we give the pseudo code for barycentric coordinates $(u, v, w = 1-u-v)$ on triangle strips along radial lines in Figure 4.

```
// reorder control points/normals (b000/n000, b030/n030, b003/n003)
// so that fu=fv on u,v isolines
void calcUVValues(float diff, float same) //fu=fv=same, fw=diff
{
float line;  // counter that traverses along the diff edge
float line1; // counter that traverses along the same edge

float incS = 0.5*(1.0/same - (same - floor(same))/same); // remainder for same fractional
float incD = 0.5*(1.0/diff - (diff - floor(diff))/diff); // remainder for diff fractional

vec2  diff1, diff2; // pair of u,v values on the diff edge
float u, v;
float par; // keeps track of the current location on the same edge

for (line=0; line < diff; ++line)
{
diff1.u = (line -incD)/diff;
if (diff1.u < 0.0)
          diff1.u = 0.0;
diff1.v = 1.0 - diff1.u;
diff2.u = (line+1.0 -incD)/diff;
if (diff2.u > 1.0)
   diff2.u = 1.0;
diff2.v = 1.0 - diff2.u;

u = 0.0;
v = 0.0;
//Use PN evaluation code to calculate the point and the normal
//In GLSL use EmitVertex()
for (line1=0; line1<same; ++line1)
{
/* evaluate the point on the second radial line */
par = (line1+1.0 -incS)/same;
if (par > 1.0)
par = 1.0;
v = diff2.y*par;
u = diff2.x*par;
//Use PN evaluation code to calculate the point and the normal
//In GLSL use EmitVertex()

/* evaluate the point on the first radial linee */
v = diff1.y*par;
u = diff1.x*par;
//Use PN evaluation code to calculate the point and the normal
//In GLSL use EmitVertex()
}
//Finish the triangle strip
//In GLSL call EndPrimitive()
}
}
```

Fig. 4. Pseudo code used for generating the barycentric coordinates in SD-2 tessellation

4 Results

The method can use any triangular parametric surface, however we have chosen the PN triangles scheme. We compare SD-2 with uniform tessellation of the PN patches as well as the stitching pattern methods described in [5], [6] and [7]. We compare frame rate and number of primitives generated on the GPU. For demonstration purposes, we have implemented all methods as geometry shaders. The SD-2 method clearly outperforms the other two and it can be clearly seen that there is a significant boost in frame rate for SD-2 by switching over to triangle strips. See Tables 1 and 2 for the concrete values on a PC with Windows Vista, 32bit, and NVIDIA 9800 GTX graphics. For surface interrogation, we

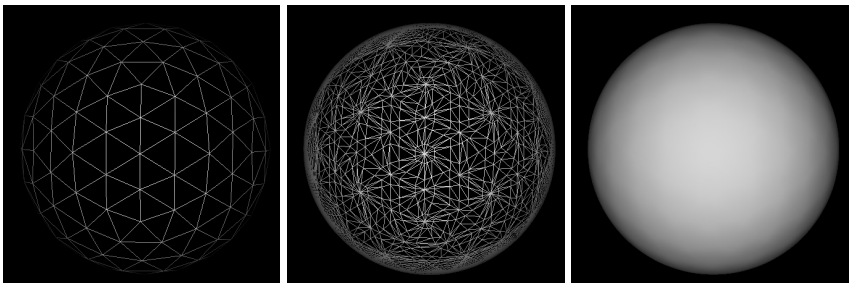

Fig. 5. Results of SD-2 refinement for improving the silhouette.

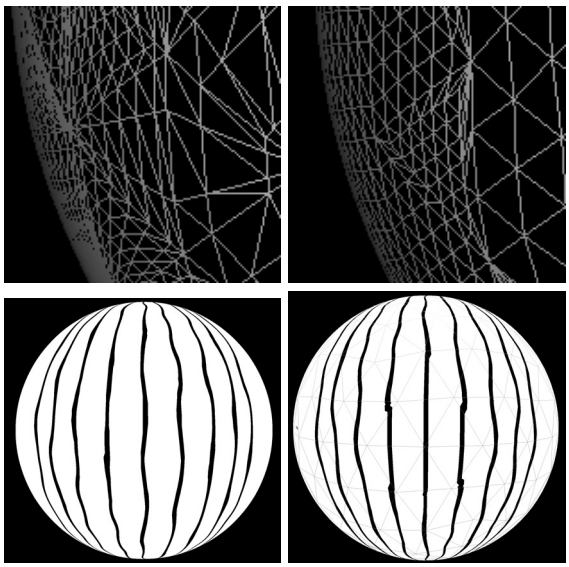

Fig. 6. Comparison of SD-2 and stitching methods using reflection lines.

render a series of reflection lines on the final surface and look at the smoothness of these lines. In general, smoother reflection lines indicate better surface quality. Reflection lines are much smoother for SD-2 in the adaptive region compared to the stitching method, see Figure 6.

5 Conclusions

In this paper, we have described, SD-2, a tessellation pattern for fractional edge tessellation factors with only two different values per triangle. Under continuous changes of the tessellation factors, the pattern fulfills all the requirements on the continuity of tessellation changes. This is especially important for the sampling of geometry and applied texture/displacement maps during animations. The

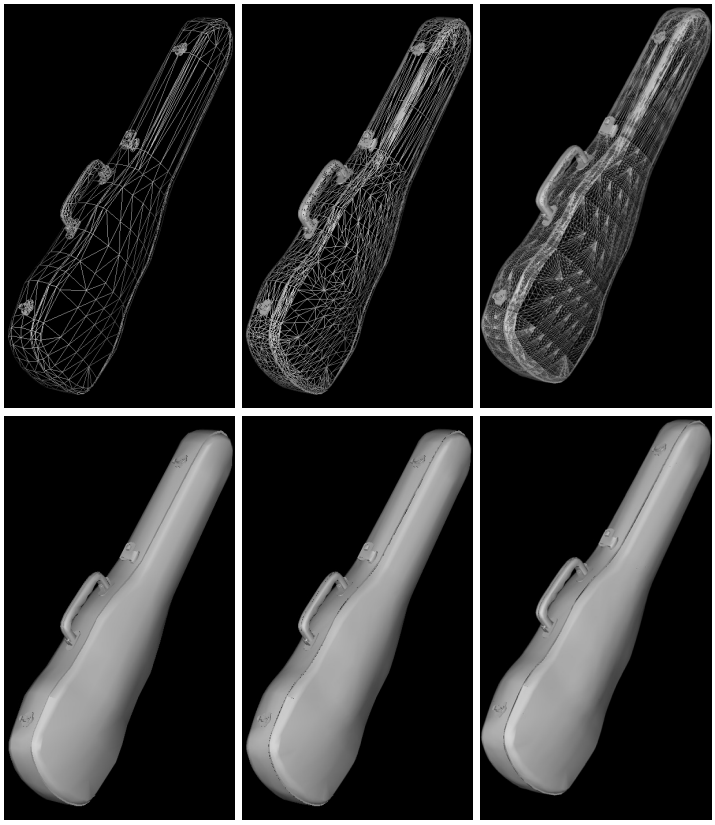

Fig. 7. Results of SD-2 refinement for continuous LOD based on the distance to the camera eye plane. From left to right, the original mesh, and adaptive tessellations generated by a linear level function with $f_{\max} = 7$, $f_{\min} = 2$, $d_{\max} = 1$ and $d_{\min} = 0$.

Table 1. Performance for silhouette refinement with max tessellation factor 7 and triangle strips.

Model Name	**Base Mesh Triangles**	**Uniform PN $f = 7$ FPS/Primitives**	**PN SD-2 FPS/Primitives**	**PN Stitch FPS/Primitives**
Sphere	320	121.0/15680	153.0/12952	122.0/12184
Violin Case	2120	19.5/103880	24.5/83932	19.9/75446
Cow	5804	7.0/284396	9.5/222586	7.4/200383

scheme is especially simple to implement, and it is suitable for triangle output in the form of triangle strips.

In terms of adaptivity, it can cover the most important cases, where the edge tessellation factors are derived from a level function on the mesh vertices. Then, the edges can be directed and 2-different edge tessellation factors can be assigned based on the minimum (or maximum) vertex on each edge. We

Table 2. Performance for silhouette refinement with max tessellation factor 7 and not using triangle strips.

Model Name	Base Mesh Triangles	Uniform PN $f = 7$ FPS/Primitives	PN SD-2 FPS/Primitives	PN Stitch FPS/Primitives
Sphere	320	60.0/15680	65.0/12952	65.0/12184
Violin Case	2120	7.5/103880	16.0/83932	16.0/75446
Cow	5804	3.2/284396	4.0/222586	4.0/200383

have shown results in terms of speed and quality with an implementation of the pattern and the edge tessellation factor assignment in the geometry shader of the GPU. This shows that the approach is also suitable for a future hardware implementation.

References

1. Sfarti, A., Barsky, B.A., Kosloff, T., Pasztor, E., Kozlowski, A., Roman, E., Perelman, A.: Direct real time tessellation of parametric spline surfaces. In: 3IA Conference. (2006) Invited Lecture http://3ia.teiath.gr/3ia_previous_conferences_cds/2006.
2. Schweitzer, D., Cobb, E.S.: Scanline rendering of parametric surfaces. SIGGRAPH Comput. Graph. **16** (1982) 265–271
3. Bóo, M., Amor, M., Doggett, M., Hirche, J., Strasser, W.: Hardware support for adaptive subdivision surface rendering. In: HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, New York, NY, USA, ACM (2001) 33–40
4. Settgast, V., Müller, K., Fünfzig, C., Fellner, D.: Adaptive Tesselation of Subdivision Surfaces. Computers & Graphics **28** (2004) 73–78
5. Moreton, H.: Watertight tessellation using forward differencing. In: HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, New York, NY, USA, ACM (2001) 25–32
6. Chung, K., Kim, L.: Adaptive Tessellation of PN Triangle with Modified Bresenham Algorithm. In: SOC Design Conference. (2003) 102–113
7. Schwarz, M., Stamminger, M.: Fast GPU-based Adaptive Tessellation with CUDA. Comput. Graph. Forum **28** (2009) 365–374
8. Gee, K.: Introduction to the Direct3D 11 graphics pipeline. In: nvision '08: The World of Visual Computing, Microsoft Corporation (2008) 1–55
9. Munkberg, J., Hasselgren, J., Akenine-Möller, T.: Non-uniform fractional tessellation. In: GH '08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association (2008) 41–45
10. Dyken, C., Reimers, M., Seland, J.: Semi-uniform adaptive patch tessellation. Computer Graphics Forum **28** (2009) 2255–2263
11. Vlachos, A., Peters, J., Boyd, C., Mitchell, J.L.: Curved PN triangles. In: I3D 2001: Proceedings of the 2001 Symposium on Interactive 3D graphics, New York, NY, USA, ACM Press (2001) 159–166
12. Farin, G.: Curves and Surfaces for Computer-Aided Geometric Design — A Practical Guide. 5^{th} edn. Morgan Kaufmann Publishers (Academic Press) (2002) 499 pages.