To appear in proceedings of Graphics Interface 2008.

PNG1 Triangles for Tangent Plane Continuous Surfaces on the GPU

Christoph Fünfzig[1, 2‡] Kerstin Müller[1, 2§] Dianne Hansford[1¶] Gerald Farin[1‖]

[1] PRISM Lab, Arizona State University, Tempe, USA
[2] Computer Graphics and Visualization, University Kaiserslautern, Germany

Abstract

Improving the visual appearance of coarse triangle meshes is usually done with graphics hardware with per-pixel shading techniques. Improving the appearance at silhouettes is inherently hard, as shading has only a small influence there and the geometry must be corrected. With the new geometry shader stage released with DirectX 10, the functionality to generate new primitives from an input primitive is available. Also the shader can access a restricted primitive neighborhood. In this paper, we present a curved surface patch that can deal with this restricted data available in the geometry shader. A surface patch is defined over a triangle with its vertex normals and the three edge neighbor triangles. Compared to PN triangles, which define a curved patch using just the triangle with its vertex normals, our surface patch is G^1 continuous with its three neighboring patches. The patch is obtained by blending two cubic Bézier patches for each triangle edge. In this way, our surface is especially suitable for efficient, high-quality tessellation on the GPU.

We show the construction of the surface and how to add special features such as creases. Thus, the appearance of the surface patch can be fine-tuned easily. The surface patch is easy to integrate into existing polygonal modeling and rendering environments. We give some examples using Autodesk Maya®.

Keywords: PN triangles, Bézier surfaces, Geometry shader, GPU-based tessellation

Index Terms: I.3.5 [Computing Methodologies]: Computer Graphics—Surface Representation, Spline, I.3.1 [Computing Methodologies]: Computer Graphics—Graphics Processors

1 Introduction

Graphics hardware is improving at a fast pace. In real-time rendering the visual appearance of triangle models is augmented by special per-pixel techniques using textures. With coarse triangle models it is inherently difficult to improve the geometric appearance at silhouettes. This is the case because shading has only a small influence there, and the geometry must be refined efficiently. Until recently, only the vertex shader was available to displace the vertices. As the bandwidth between CPU and GPU is restricted, the main problem is how to communicate the mesh and displacement data efficiently to the GPU. In the geometry shader released with the Direct3D 10 API, there is the possibility to generate new primitives using a restricted input mesh. There is fixed-size neighborhood information available consisting of the triangle and its three edge neighbor triangles, see Figure 1, right. In this paper, our task is to construct a G^1 continuous surface patch defined by exactly this restricted input mesh. The resulting surface should be

- reasonable smooth,
- on edges the smoothness should be taggable so that special features are available similar to creases on subdivision surfaces,
- a surface patch should be suitable and efficient for GPU rendering with the new geometry shader stage.

To meet these requirements, we construct two cubic triangular Bézier patches for each edge of a triangle. These two cubic triangular Bézier patches generate a G^1 join across their edge. To get the final G^1 surface, we blend the resulting triangular Bézier patches together such that G^1 continuity is not destroyed. We start with related work for this problem in Section 2. Section 3 describes our construction of the PNG1 surface. After a short overview of the method, we summarize some results on cubic Bézier triangles and their G^1 continuity (Section 3.1). Then we apply a G^1 construction on each side of the triangle. Section 3.2 contains also the derivation of the blending functions and the overall surface evaluation. For improved surface lighting, Section 3.3 contains two options to get normals: original normals derived from the PNG1 patches and modified normals by interpolation. Special features are presented in Section 3.4. The properties and applications (Section 3.5) of PNG1 surfaces on GPUs are explained afterwards. We show visual results using Autodesk Maya® and give additional details on the implementation (Section 4). Finally, we close with conclusions and possible further work (Section 5).

2 Related Work

For modeling of curved surfaces there exists a zoo of different surface types. Parametric surfaces such as Bézier and B-Spline surfaces are defined by a control mesh [7]. Also surfaces resulting

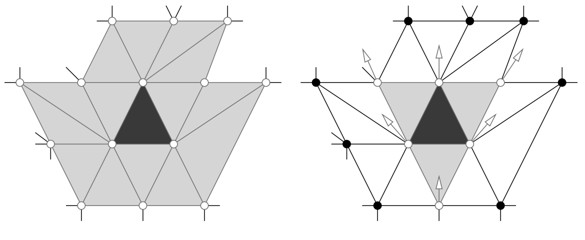


Figure 1: On the left, the required neighborhood (light gray) for the tessellation of the gray marked face using the Loop subdivision scheme. On the right, the neighborhood (light gray) of the gray marked face on the GPU (GL_TRIANGLES_ADJACENCY_EXT).

from subdivision of a base mesh are available. For a prominent example, the Loop subdivision surface [14] works with a triangle base mesh and generates an approximating, C^2 continuous surface except at the irregular points where it is only C^1. For a refinement step, the complete 1-neighborhood of a triangle is required as

‡e-mail: c.fuenfzig@gmx.de
§e-mail: kerstin.mueller@gmx.org
¶e-mail: dianne.hansford@asu.edu
‖e-mail: gerald.farin@asu.edu

shown in Figure 1, left. We define the restricted neighborhood as the light gray triangles with its vertex normals in Figure 1, right.

The problem of mesh refinement consists of two sub-problems. Firstly, the defining data for a surface patch has to be transferred to the computation unit. Secondly, it remains to tessellate the surface patch at a sufficient resolution to show all the surface details. In principle, this can be done with a fixed tessellation or with adaptive patterns resulting from vertex depth tags. The process of mesh refinement has been covered by Adaptive Refinement Patterns (ARP) in general [2], and has been specially solved for NURBS and their complex trimming [10]. These approaches choose a sufficiently fine tessellation (already stored on the GPU) and displace it within the vertex shader stage. Some adaptivity is achieved by precomputing all possible fitting tessellations corresponding to the resolutions at the edges.

Defining Patch Data

As mechanisms to transfer data to the GPU, vertex attributes and recently also uniform buffer objects are available. These can be combined to transfer all defining data for a surface patch. GPGPU techniques for encoding the mesh connectivity and mesh geometry into textures [21, 4] have been proposed to make larger neighborhoods available in the vertex and fragment shader stages of the GPU. The variety of schemes possible in this way is very broad but the resulting algorithms are inherently multi-pass and still require considerable preparation on the CPU after changes of the mesh connectivity. For subdivision surfaces (Loop, Butterfly), an approximation of a limit triangle by a quadratic surface patch interpolating in the triangle corner and mid-edge vertices, has been proposed in [3]. After one subdivision step on the CPU into four limit triangles using the 1-neighborhood, the 1-neighborhood is no longer necessary to do the local approximation by quadratic surface patches in a PN-like fashion [22]. Again there is a complex CPU preparation process involved. Unfortunately, this quadratic approximation destroys the C^1 continuity of the subdivision surface. A similar approach was proposed for the adaptive tessellation of Catmull-Clark surfaces [15]. The Catmull-Clark surface is approximated by bicubic surface patches defined by 16 control points. In this way, it can be evaluated on current GPUs. A variable-size hardware vertex cache has been proposed in [13] to make the 1-neighborhood available in future GPU designs.

Tessellation

Besides using precomputed tessellation patterns in a patch-by-patch rendering, the geometry shader released with the Direct3D 10 API [1] can generate new triangles or triangle-strip primitives and, even more important, offers new primitives for patch definition. The complete neighborhood of a triangle is also not available but the edge triangle neighbors can be accessed inside a new primitive consisting of six vertices (Figure 1). In [2], the current restriction to generate not more than 1024 varying floats is pointed out. With vertex coordinates and vertex color/vertex normal there are 8 floats per vertex so that 126 triangles can be emitted in one triangle strip. It is worthwhile to note that the new input primitives can be batched efficiently, i.e., large batches of these primitives can be issued in one API call. This is a considerable improvement over a patch-by-patch setup each time a surface patch is rendered.

Parametric surfaces like curved PN triangles [22] can deal with the restricted neighborhood information. A PN triangle is generated by a single triangle and its vertex normals. The construction uses a cubic Bézier patch that is only C^0 in general. To give the impression of a smooth surface, the normals are computed separately from the surface via a smart normal interpolation method. Figure 2 shows an example of PN triangles. Because PN triangles are simple to evaluate, they found their way into graphics hardware (*TruForm rendering*) [9]. Mann and Davidchuk presented a

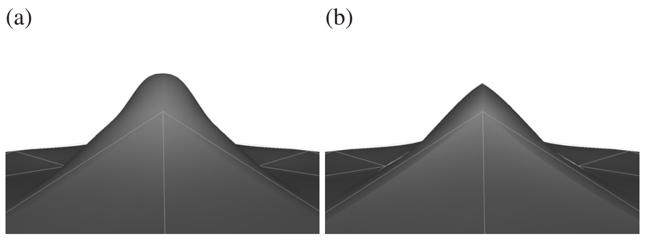


Figure 2: Comparison PNG1 (a) and PN (b). The C^0 continuity of the PN patch is clearly visible at the silhouette. The PNG1 patches present a G^1 surface with a smooth silhouette.

more sophisticated construction in [6, 17] that has G^1 continuity and uses only the restricted neighborhood. They generate a hybrid Bézier patch that blends the interior as well as the boundary control points. The construction makes use of a planar scattered data interpolation scheme proposed in [8]. Unfortunately, they do not offer special features for their surface type. Our surface patch is similar in spirit but our construction only uses a sufficient condition for cubic Bézier triangles to be C^1 continuous across an edge. By controlled violation of this condition, we can easily create special features like sharp edges.

In [16], parametric surface interpolation is surveyed, and the paper compares local and global methods. From the local methods, the one by Shirman and Sequin [20] constructs three quartic triangular Bézier surfaces per patch similarly to the method of Chiyokura and Kimura [5]. Further on, they present Nielson's side vertex method [18] with rational blending of three cubic triangular Bézier surfaces. They state that with global methods the remaining free parameters can be chosen much better compared to the heuristic choices in the local methods. For our application though, global methods are not an option due to their preparation requirements. Later, Hahmann and Bonneau [11] presented an interesting local method using the 4-split instead of the Clough-Tocher-split in the previous methods. They define four quintic patches leaving three free Bézier control points. Unfortunately, the use of the complete 1-neighboorhood and the computation of 63 control points makes this nice method unsuitable for the GPU.

3 PNG1 TRIANGLE PATCHES

In general, it is *not* possible to build a G^1 join between two neighboring cubic Bézier triangles if only the middle control point of both Bézier patches is variable. This is also the case if the control points adjacent to the corner points share the same tangent plane (see [19] for more details and an example). Therefore, we construct for each edge of the triangle, a surface based on two cubic Bézier patches that has the G^1 join to its edge-neighbor. The resulting three surfaces from the three sides are blended together conveniently to maintain the G^1 property on all three boundaries. That way, we obtain a kind of a cubic Bézier patch whereby its control points are replaced by control functions. Similar to the method in [22], to alleviate the effect of the curvature discontinuity that is not eliminated by a G^1 surface, we smooth the normals via a quadratic interpolation. The resulting surface looks smoother and gives us the desired appearance. To enhance the variety of modeling options, special features are used. Special features reduce the continuity locally on the surface to allow for, e.g., sharp edges. They were introduced for subdivision surfaces firstly but they are also available for PNG1.

3.1 Cubic Bézier Triangles

A Bézier triangle is defined by

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

where b_{ijk} are the control points of the Bézier triangle and $w = 1-u-v$ [7]. For $n = 3$ we get a cubic Bézier triangle (see Figure 3, left, and Appendix A). The normals of the surface can be computed by

$$\vec{N}(u,v,w) = D_{\vec{u}}B(u,v,w) \times D_{\vec{v}}B(u,v,w) \tag{2}$$

where

$$\vec{u} = [1,0,-1]$$
$$\vec{v} = [0,1,-1]$$

We obtain a C^1 join between two adjacent Bézier triangles, if the quadrangles $\square p_i q_i r_i q_{i+1}$, where $i = 0,\ldots,n-1$, are coplanar and all quadrangles are similar [7] (see Figure 3). We will use this condition for the construction of our PNG1 patch to ensure a C^1 join to its adjacent triangles. In this way, we achieve a surface that is at least G^1 everywhere.

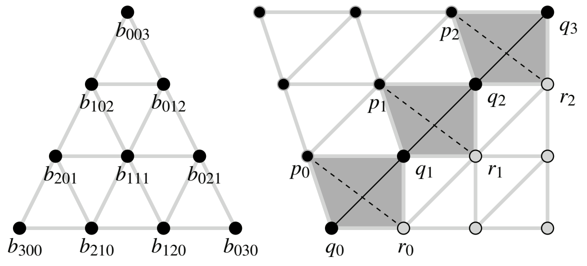


Figure 3: Left: Control points of a cubic Bézier patch. Right: C^1 join of two cubic Bézier patches.

3.2 Construction of a PNG1 Triangle Patch

We use the notation described in Figure 4 and start with building the surface corresponding to the edge $\overline{P_1P_2}$. The surfaces for the edges $\overline{P_2P_3}$ and $\overline{P_3P_1}$ can be generated similarly.

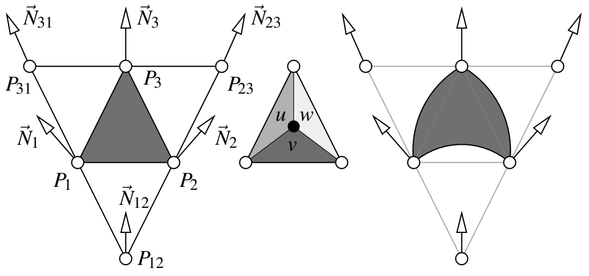


Figure 4: Notation for the input mesh for a PNG1 patch (left) and the resulting shape (gray marked, right). The parameter space and the barycentric coordinates are illustrated in the middle.

Consider the tangent plane of P_1 defined by the point P_1 and its normal $\vec{N}_1$ (see Figure 5). We project the points P_2, P_3, and P_{12} in the direction of $\vec{N}_1$ into the tangent plane of P_1. To preserve the edge length in the projection, we scale the vector $\text{proj}(P_i) - P_1$ until it has the same length as $P_i - P_1$ for $i = 2,3,12$. The result of this projection and scaling are two adjacent triangles in the tangent plane as illustrated in Figure 5, right. Subdivision of these two triangles by factor $1/3$ provides for each triangle the three red colored similar subtriangles. They form three similar quadrangles that are coplanar. The same procedure can be done with the tangent plane of P_2 defined by point P_2 and its normal $\vec{N}_2$: P_1, P_{12}, and P_3 are projected into the tangent plane of P_2 and a scaling is done to maintain the original edge length. The subdivision of the two projected triangles in the tangent plane of P_2 results also in three similar quadrangles.

For the construction of the PNG1 patch (see Figure 6, top) we need the triangle adjacent to P_1 in the tangent plane of P_1 ($\triangle P_1 b_{P1:210} b_{P1:201}$, red marked) and also the triangle adjacent to P_2 in the tangent plane of P_2 ($\triangle b_{P2:120} P_2 b_{P2:021}$, blue marked). An affine transformation of the red triangle from tangent plane P_1 into the tangent plane of P_2 provides the point $b_{P1:021}$ (see Figure 6, bottom). In an analogous manner, point $b_{P2:201}$ is computed by transferring the blue triangle from tangent plane P_2 into tangent plane P_1.

For the middle control points, we generate a plane defined by

$$\vec{e}_1 = b_{P2:120} - b_{P1:210} \tag{3}$$
$$\vec{e}_2 = (\vec{N}_T + \vec{N}_{T12}) \times \vec{e}_1 \tag{4}$$
$$\vec{N}_{12} = \vec{e}_1 \times \vec{e}_2 \tag{5}$$

where $\vec{N}_T$ is the normal of the triangle plane P_1, P_2, P_3, and $\vec{N}_{T12}$ denotes the normal of P_1, P_{12}, P_2 (see Figure 7). A transfer of the red triangle from the tangent plane of P_1 and a transfer of the blue triangle from the tangent plane of P_2 provides the points $b_{12:P1:111}$ and $b_{12:P2:111}$. Constructed in this way, the resulting red and blue triangles respectively are similar to each other. With the same procedure for the adjacent triangle $\triangle P_2 P_1 P_{12}$ we get coplanar similar quadrangles which results in the G^1 property.

Applying the described construction also to the edges $\overline{P_2P_3}$ and $\overline{P_3P_1}$, provides the points as illustrated in Figure 8, top. The red control points were created via the tangent plane of P_1, the blue ones are constructed using the tangent plane of P_2, and the green control points are generated by the tangent plane of P_3. Now we map the calculated control points to a control function (see Figure 8, bottom). The indexing of the control point $b_{Pl:ijk}$ results from its creation by tangent plane of P_l, $l = 1,2,3$ and its correspondence to control function ijk, where ijk is the standard indexing for Bézier triangles. We use control functions instead of control points to get the form of a cubic Bézier patch:

$$S(u,v,w) = \sum_{i+j+k=3} \frac{n!}{i!j!k!} u^i v^j w^k b_{ijk}(u,v,w) \tag{6}$$

where $b_{300}(u,v,w) = b_{300}$, $b_{030}(u,v,w) = b_{030}$, and $b_{003}(u,v,w) = b_{003}$ are the triangle vertices. The control functions $b_{ijk}(u,v,w)$ blend the assigned control points and deliver for each parameter (u,v,w) a suitable control point.

To specify the blending functions $b_{ijk}(u,v,w)$ we look at the side $\overline{P_1P_2}$ firstly. To get a G^1 join across $\overline{P_1P_2}$, the following conditions are sufficient on the border curve $v = 0$:

1. On the border curve, we are required to use only the control point constructed from this edge
   $b_{210}(u,v,w)\mid_{v=0} = b_{P1:210}$ and $b_{120}(u,v,w)\mid_{v=0} = b_{P2:120}$

2. Likewise, the derivatives of $b_{210}(u,v,w)$ and $b_{120}(u,v,w)$ should be zero on this edge:
   $D_{\vec{u}}b_{210}(u,v,w)\mid_{v=0} = 0, \quad D_{\vec{v}}b_{210}(u,v,w)\mid_{v=0} = 0$
   $D_{\vec{u}}b_{120}(u,v,w)\mid_{v=0} = 0, \quad D_{\vec{v}}b_{120}(u,v,w)\mid_{v=0} = 0$

3. For the remaining control points affecting the continuity on this side, we need to blend between $b_{P1:201}$ and $b_{P2:201}$, $b_{P1:021}$ and $b_{P2:021}$, $b_{12:P1:111}$ and $b_{12:P2:111}$ for u from 0 to 1, w from 1 to 0.

Property 1 and 3 follow directly from the construction of these points. Property 2 results from the surface derivatives on the side $v = 0$. See Appendix B for the surface derivatives $D_{\vec{u}}S(u,v,w)$, $D_{\vec{v}}S(u,v,w)$ in terms of the derivatives of the control point functions. Analoguous conditions apply to the edges $\overline{P_2P_3}$ and $\overline{P_3P_1}$.

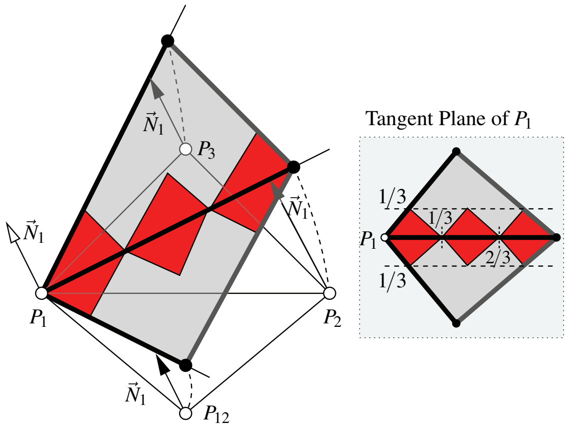


Figure 5: Left: Normal projection of P_2, P_3, P_{12} onto the tangent plane of P_1. The length of the vectors proj$(P_2)-P_1$, proj$(P_3)-P_1$, proj$(P_{12})-P_1$ is scaled to equal the original length P_2-P_1, P_3-P_1, $P_{12}-P_1$. Right: A subdivision of the two projected triangles by factor $1/3$ provides for each triangle the three red colored similar subtriangles.

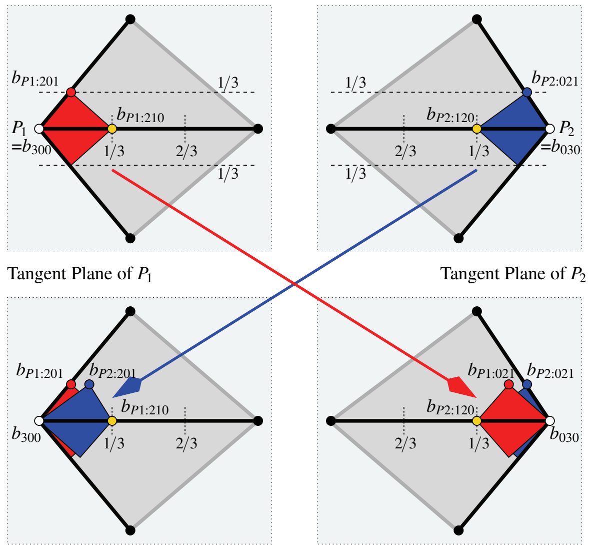


Figure 6: Affine transformation of the two red triangles from tangent plane P_1 into the tangent plane of P_2. Likewise, make an affine transformation of the two blue triangles from tangent plane P_2 into the tangent plane of P_1. The resulting red and blue quadrangles respectively are coplanar and similar.

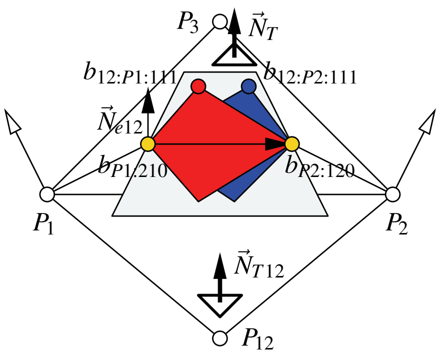
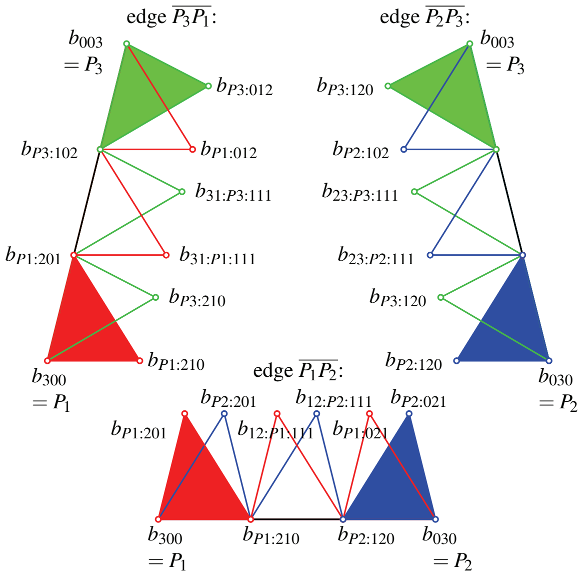


Figure 7: Construction of the middle plane by the two adjacent triangle normals and $b_{P2:120}-b_{P1:210}$. The red and blue triangles from the tangent planes of P_1 and P_2 are transferred into the middle plane to compute the points $b_{12:P1:111}$, $b_{12:P2:111}$.



Grouping of the control points:

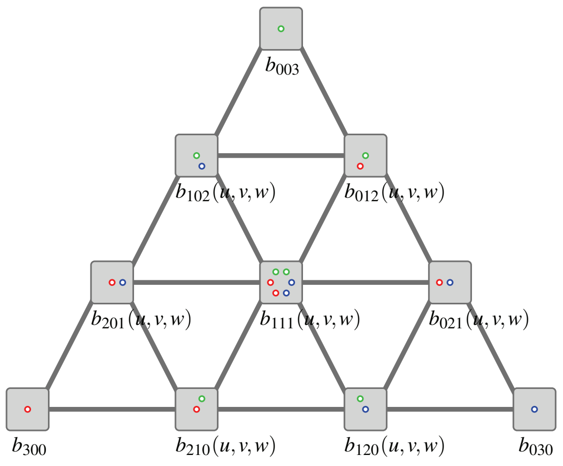


Figure 8: Top: Construction of a G^1 patch for each side provides the points illustrated. Bottom: These points are schematically grouped together. A control function blends the points.

We designed the blending functions to fulfill all the properties 1–3 for all edges of the triangle:

$$b_{201}(u,v,w) = \frac{1}{u^2+(1-u)^2}\left((1-u)^2 b_{P1:201} + u^2 b_{P2:201}\right)$$
$$b_{102}(u,v,w) = \frac{1}{u^2+(1-u)^2}\left((1-u)^2 b_{P3:102} + u^2 b_{P2:102}\right)$$
$$b_{012}(u,v,w) = \frac{1}{w^2+(1-w)^2}\left((1-w)^2 b_{P3:012} + w^2 b_{P1:012}\right)$$
$$b_{021}(u,v,w) = \frac{1}{w^2+(1-w)^2}\left((1-w)^2 b_{P2:021} + w^2 b_{P1:021}\right)$$
$$b_{120}(u,v,w) = \frac{1}{v^2+(1-v)^2}\left((1-v)^2 b_{P2:120} + v^2 b_{P3:120}\right)$$
$$b_{210}(u,v,w) = \frac{1}{v^2+(1-v)^2}\left((1-v)^2 b_{P1:210} + v^2 b_{P3:210}\right)$$

The inner function $b_{111}(u,v,w)$ must deal with six control points, two from each side, so Nielson's blending function [18] occurs also in our blending function for the inner control point:

$$\begin{aligned} b_{111}(u,v,w) = {} & \\ & \frac{uw}{uv+uw+vw}\left(\frac{(1-u)wb_{12:P1:111} + u(1-w)b_{12:P2:111}}{w+u-2uw}\right) + \\ & \frac{uv}{uv+uw+vw}\left(\frac{(1-v)ub_{23:P2:111} + (1-u)vb_{23:P3:111}}{u+v-2uv}\right) + \\ & \frac{vw}{uv+uw+vw}\left(\frac{(1-w)vb_{31:P3:111} + (1-v)wb_{31:P1:111}}{w+v-2vw}\right) \end{aligned}$$

3.3 PNG1 Normals

The normals of our surface can be computed directly (see also Appendix B) by

$$\vec{N}(u,v,w) = D_{\vec{u}}S(u,v,w) \times D_{\vec{v}}S(u,v,w) \qquad (7)$$

However, we deal with a G^1 surface and therefore, it happens that we do not get the desired smooth appearance, visible especially in reflections. Thus, for a kind of normal smoothing, we use a method similar to the one presented in [22] by Vlachos et al.

We get the smoothed normals via a quadratic Bézier triangle patch

$$\begin{aligned} N_S(u,v,w) &= \sum_{i+j+k=2} \frac{2!}{i!j!k!} u^i v^j w^k n_{ijk} \\ &= u^2 n_{020} + v^2 n_{002} + w^2 n_{200} + \\ &\quad 2wu\, n_{110} + 2uv\, n_{011} + 2wv\, n_{101} \end{aligned}$$

where (see Figure 4 for notation) $n_{020} = \vec{N}_2 = \vec{N}(1,0,0)$, $n_{002} = \vec{N}_3 = \vec{N}(0,1,0)$, $n_{200} = \vec{N}_1 = \vec{N}(0,0,1)$, and

$$n_{110} = 2\vec{N}(\frac{1}{2},0,\frac{1}{2}) - \frac{1}{2}(\vec{N}_1+\vec{N}_2)$$
$$n_{011} = 2\vec{N}(\frac{1}{2},\frac{1}{2},0) - \frac{1}{2}(\vec{N}_2+\vec{N}_3)$$
$$n_{101} = 2\vec{N}(0,\frac{1}{2},\frac{1}{2}) - \frac{1}{2}(\vec{N}_3+\vec{N}_1)$$

In this way, we obtain the original surface normals in the middle of the boundary curves as well as on the corners. With these normals, the surface appearance gives us the impression that it is smoother because curvature discontinuities are alleviated.

3.4 Special Features for PNG1 Triangle Patch

In addition to smooth surfaces, sharp edges are desired for advanced modeling applications. Loop and Catmull Clark subdivision surfaces offer nice kinds of special features (e.g., [12]) which should also be available for our PNG1 surface similarly.

From a user's point of view, sharp edges are obtained by setting sharpness values κ on designated edges of the base mesh. To generate a sharp edge with PNG1 patches, we use the same construction process as described in Section 3.2 with a minor change: If we have a sharp edge, e.g., on the side $\overline{P_1P_2}$, the regular computation of $\vec{e}_2$ (see Equation 4) is substituted by

$$\vec{e}_2 = \vec{N}_T \times \vec{e}_1$$

To emphasize the sharp edge, the points $b_{P2:201}$, $b_{12:P1:111}$, $b_{12:P2:111}$, $b_{P1:021}$ can be moved by $\kappa\vec{N}_T$. Models with special features are presented in Figures 10 and 12, right.

3.5 Properties and Applications for PNG1 Patches

From the construction of the PNG1 patch, it follows that the surface is at least G^1 at the border and G^2 in the interior of the patch. The surface patches are reasonably smooth even when a silhouette is observed, which is an additional advantage over PN triangles. Special features as known from Loop subdivision surfaces can be implemented for PNG1 patches. The appearance of the sharp edges for PNG1 differ from the ones of Loop because we deal with an interpolating method instead of an approximating one.

The G^1 surface above has been designed for the restricted neighborhood directly available, for example, in the geometry shader stage of modern GPUs. The application only needs to keep track of the edge neighbors on triangles, which is of constant size per triangle and can be maintained easily. This enables vertex attribute and index data to be processed in large batches similar to standard triangle models. The geometry shader can then carry out the evaluation of the parametric surface with formulas given in Appendix A. Using a uniform tessellation, we step through the barycentric parameter domain in rows of constant w and generate a triangle strip per row. Each emitted vertex can have a lighted color computed in the geometry shader or a normal for per-pixel lighting in the fragment shader. In each case, this requires 8 float varyings per vertex. Using l points per triangle side, this requires l triangle strips of lengths $(2l+1),\ldots,3$, defining $(2l-1),\ldots,3,1$ triangles. So a vertex sequence of length $(2l+1)+\ldots+3 = (l+1)l+l = l^2+2l$ defines $(2l-1)+\ldots+3+1 = (l+1)l-l = l^2$ triangles.

4 Results

We have implemented the PNG1 patches as a *MPxHwShaderNode* within Autodesk Maya®. The Polygons part of Autodesk Maya® is a classic polygonal modeller, and lots of low-level and high-level functions are available for surface creation. Without sharpness tags for creases, the PNG1 triangles are defined by ordinary triangle models so that it is not necessary to touch the underlying database. Even sharpness tags can be integrated easily, either directly by a new vertex-face attribute or by using a component of the vertex-face color.

For assessing the surface shape, we rendered some simple polyhedral shapes with Phong shading in comparison to the PN patches. Figure 9 and Figure 11 (left) show the PNG1 and the PN surface, Figure 9 presents also the triangle model with its normals. The PN patches usually look good in the inside due to surface lighting generated by a smooth normal field but at the silhouettes, kinks caused by the C^0 continuous surface are clearly visible and cannot be covered up with normal interpolation.

We have used the interpolated normals described in Section 3.3. They can largely improve the surface lighting in comparison to the original normals, so that the G^1 continuous surface gets lighted with

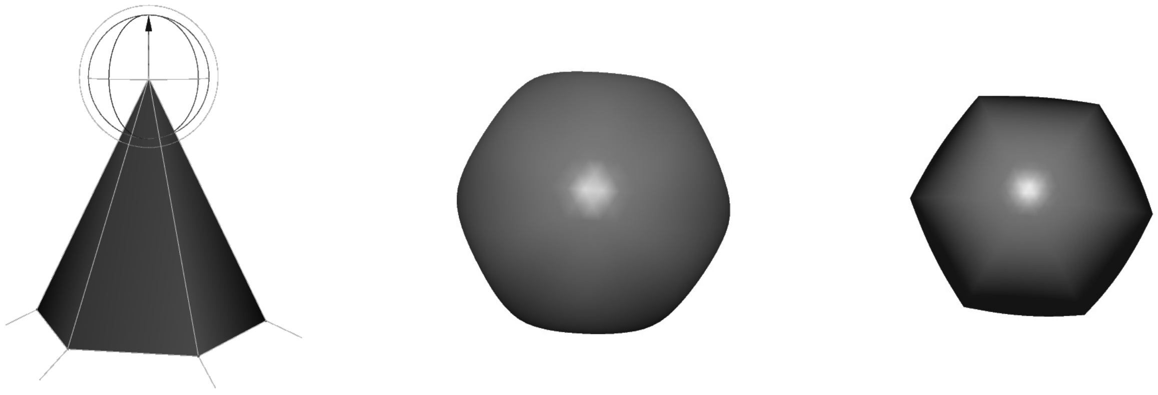


Figure 9: Cone of valence 6 from above. Triangle model with symmetrical normals (left), PNG1 patches (middle) and PN patches (right).

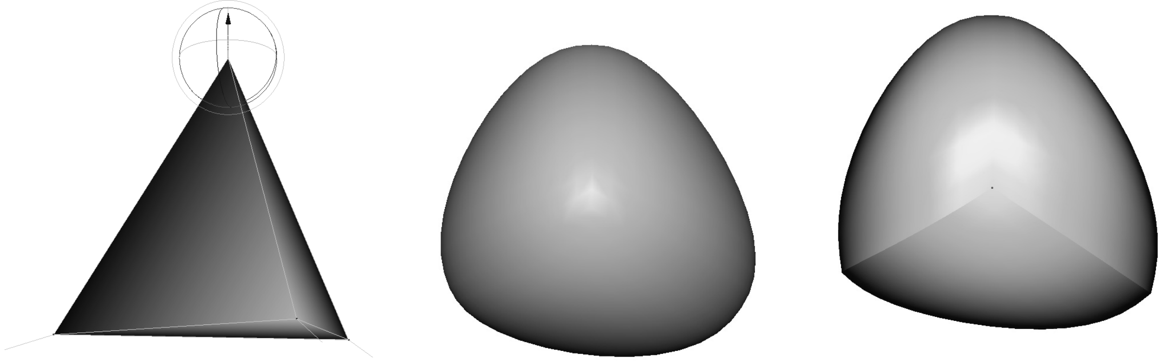


Figure 10: Smooth tetrahedron and tetrahedron with special features. Triangle model with centroid normals (left), PNG1 patches (middle), and PNG1 patches with creased bottom triangle of sharpness value $\kappa = -0.1$ (right).

an enhanced smooth normal field, which interpolates the exact normals at the triangle corners and edge mid-points.

On the tetrahedron (Figure 10) and the spaceship model (Figure 12, right), we show the effect of sharp edges on the elsewhere G^1 continuous surface.

More complex models are presented in Figure 11. The head model was previously used in the paper on PN triangles [22]. It can be seen that the shape of the PNG1 patches is more curved and smoother especially at the silhouettes. The appearance away from silhouettes is good with both due to the interpolated normals.

Figure 13 shows the Loop surface on a bunny base mesh in comparison to the PNG1 surface. For the PNG1 surface, we used the base mesh of same topology but with points/normals computed by Loop limit rules. Using these normals, both schemes generate nearly the same shape. Small differences occur in the ear and eye regions. Using normals averaged from the incident face normals, shows some more differences, especially at the ear and the tail.

We want to report on the performance of the uniform tessellation in a geometry shader writing out the vertex and normal coordinates as vertex attributes required for per-pixel lighting. The uniform tessellation can generate 100 triangles with a vertex sequence of length $l^2 + 2l = 120 < 1024/8 = 128$ in $l = 10$ triangle strips per patch. This bound results from 1024 varying floats available on a NVidia Geforce G8800 GTX used by 8 varying floats per vertex. With this resolution, we achieve the following framerates for the PN head model (200 triangles): 214 frames per second (fps) using the PN shader, and 93 fps using the PNG1 shader. With depth $l = 9$, we achieve 258 fps using the PN shader, 112.5 fps using the PNG1 shader and with depth $l = 8$, 316.5 fps using the PN shader, 138.5 fps using the PNG1 shader. For a model of typical size 1084 triangles, we observe 42 fps using the PN shader, and 14 fps using the PNG1 shader. With depth $l = 9$, we achieve 50 fps using the PN shader, 16.5 fps using the PNG1 shader and with depth $l = 8$, 61.8 fps using the PN shader, 20 fps using the PNG1 shader. Note that OpenGL vertex arrays were used for issueing the vertex and index data in these benchmarks.

5 Conclusion and Further Work

In this paper, we present an enhancement of PN patches by generating at least G^1 continuous surfaces from triangle models and their normals, called PNG1 patches. PNG1 patches are based on cubic Bézier triangles and use rational blending functions. This new curved surface significantly improves the appearance of silhouettes on shaded surface tessellations. Compared to subdivision surfaces like Loop surfaces, the new surface just uses the triangle's edge neighbors in its construction, which is of constant size, six points and six normals. In terms of surface quality such a scheme is inferior to the Loop scheme but generates nice G^1 surfaces. The appearance of shading can be further and separately improved by normal interpolation of the corner and mid-edge normals.

The PNG1 patches are suitable for direct rendering in the geometry shader stage of modern GPUs and its restricted triangle neigh-

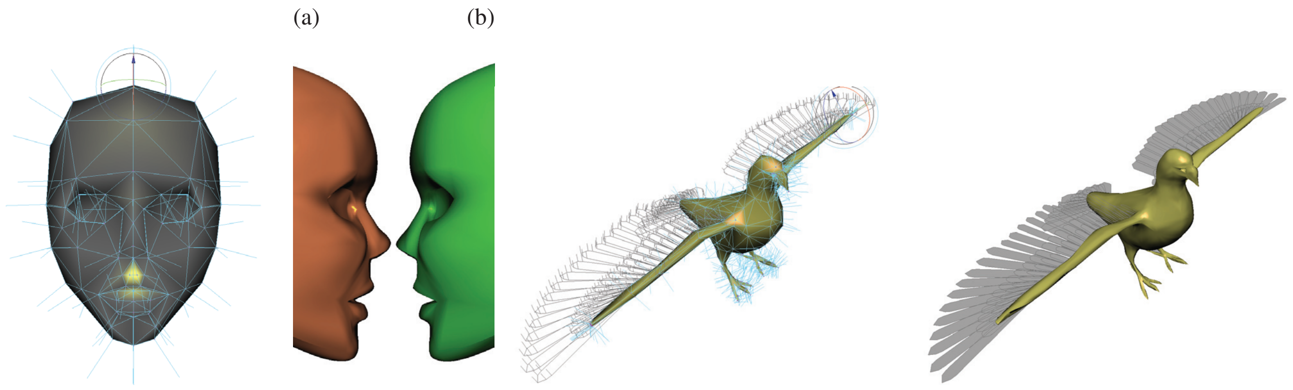


Figure 11: Comparison PNG1 (a), PN (b) for the head model (200 triangles), originally used in the PN paper [22]. Further PNG1 example of a Dove model (832 triangles).

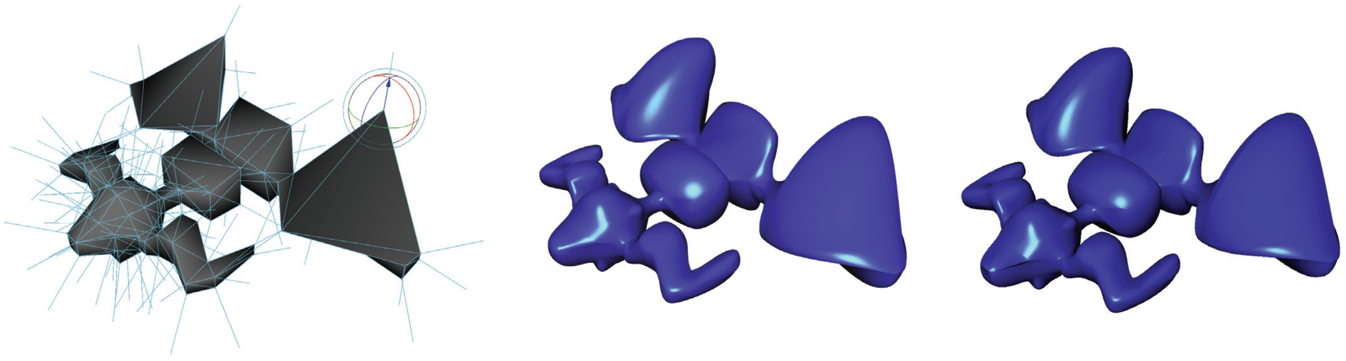


Figure 12: PNG1 Examples. Spaceship model (460 triangles, left), PNG1 model (middle), and PNG1 model with creases of value $\kappa = -0.2$ (right).

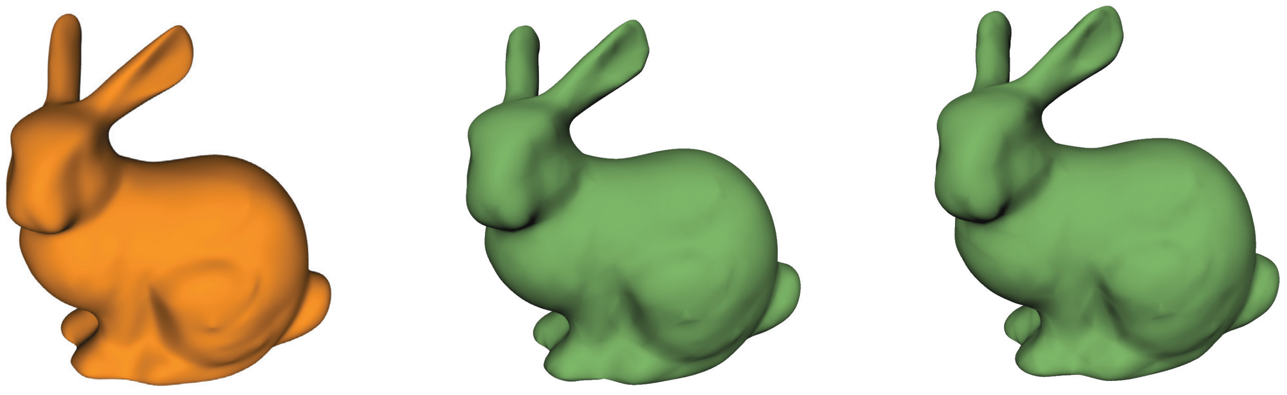


Figure 13: Loop surface on bunny mesh (910 triangles, left), PNG1 surface on base mesh of same topology and Loop limit points/normals (middle), and with Loop limit points but normals computed by averaging incident face normals (right).

borhood available with GL_TRIANGLES_ADJACENCY_EXT. They do not require any precomputation step on the CPU and are easy to implement. The geometry shader with drivers for current hardware is restricted to 1024 varying floats. We expect that this will improve with upcoming drivers and the next hardware update cycle. For fine tessellations, it is also possible to use the tessellation approach proposed in [2] using vertex buffer objects with predefined tessellation patterns and surface evaluation in the vertex shader.

As further work, we want to allow arbitrary sharp edges independent of the input mesh. This makes feature creation with PNG1 triangles even more flexible as the triangle mesh does not need to be changed.

Acknowledgements

We would like to thank the German Research Council (DFG) for the support of one author by a postdoc scholarship. We thank Jörg Peters for providing us with the head model originally used in the PN triangles paper. Further thanks go to Mo Li for the flying sequence, we used in the movie, and for help with some Autodesk Maya® questions. We would like to thank Volker Settgast for the animated knight model and Torsten Techmann for providing us with the bunny model with Loop limit points/normals. Last but not least, we thank the artist Maurizio (*http://www.abyssalplains-music.info*) for the permission to use his track *Orange River* in our movie.

References

[1] D. Blythe. The Direct3D 10 system. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 724–734, New York, NY, USA, 2006. ACM Press.

[2] T. Boubekeur and C. Schlick. A Flexible Kernel for Adaptive Mesh Refinement on GPU. *Computer Graphics Forum*, 2007.

[3] T. Boubekeur and C. Schlick. QAS: Real-time Quadratic Approximation of Subdivision Surfaces. In *Proceedings of Pacific Graphics 2007*, November 2007.

[4] M. Bunnell. Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping. In *GPU Gems II*. Addison-Wesley, 2005.

[5] H. Chiyokura and F. Kimura. Design of Solids with free-form Surfaces. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 289–298, New York, NY, USA, 1983. ACM Press.

[6] M. Davidchuk and S. Mann. A Parameteric Hybrid Triangular Bézier Patch. In M. Daehlen, T. Lyche, and L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces 2*, pages 335–342, 1998.

[7] G. Farin. *Curves and Surfaces for CAGD*. Morgan Kaufmann Publishers, 2002.

[8] T. Foley and K. Opitz. Hybrid cubic Bézier Triangle Patches. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design 2*, pages 275–286. Academic Press, 1992.

[9] D. Ginsburg and A. Vlachos. ATI TruForm Rendering. http://ati.amd.com/developer/truform_faq.html, 2001.

[10] M. Guthe, A. Balázs, and R. Klein. GPU-based Trimming and Tessellation of NURBS and T-Spline Surfaces. *ACM Trans. Graph.*, 24(3):1016–1023, 2005.

[11] S. Hahmann and G.-P. Bonneau. Triangular G1 interpolation by 4-splitting domain triangles. *Computer Aided Geometric Design*, 17(8):731–757, 2000.

[12] H. Hoppe, T. DeRose, T. Duchamp, H. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise Smooth Surface Reconstruction. In *Proceedings of SIGGRAPH 1994*, pages 295–302, 1994.

[13] M. Kazakov. Catmull-Clark Subdivision for Geometry Shaders. In *AFRIGRAPH '07: Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 77–84, New York, NY, USA, 2007. ACM.

[14] C. Loop. Smooth Subdivision Surfaces Based on Triangles. Master's thesis, University of Utah, 1987.

[15] C. Loop and S. Schaefer. Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches. Technical Report MSR-TR-2007-44, Microsoft Research, Apr 2007.

[16] M. Lounsbery, S. Mann, and T. DeRose. Parametric Surface Interpolation. *IEEE Comput. Graph. Appl.*, 12(5):45–52, 1992.

[17] S. Mann. An Improved Parametric Side–Vertex Triangle Mesh Interpolant. In *Graphics Interface*, pages 35–42, 1998.

[18] G. M. Nielson. A Transfinite, Visually Continuous, Triangular Interpolant. In *Geometric Modeling: Algorithms and New Trends*, pages 235–245. SIAM, 1987.

[19] B. R. Piper. Visually Smooth Interpolation with Triangular Bézier Patches. In *Geometric Modeling: Algorithms and New Trends*, pages 221–233. SIAM, 1987.

[20] L. Shirman and C. Sequin. Local Surface Interpolation with Bézier Patches. *CADG*, 4:279–295, 1987.

[21] L.-J. Shiue, I. Jones, and J. Peters. A Realtime GPU Subdivision Kernel. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1010–1015, New York, NY, USA, 2005. ACM Press.

[22] A. Vlachos, J. Peters, C. Boyd, and J. L. Mitchell. Curved PN Triangles. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 159–166. ACM Press, 2001.

A Point and derivative evaluation for cubic Bézier Triangles

$$\begin{aligned}B(u,v,w) = {} & u^3 b_{030} + v^3 b_{003} + w^3 b_{300} + 3u^2 w b_{120} + \\ & 3u^2 v b_{021} + 3v^2 w b_{102} + 3v^2 u b_{012} + \\ & 3w^2 u b_{210} + 3w^2 v b_{201} + 6wuv b_{111}\end{aligned}$$

$$\begin{aligned}D_{\vec{u}}B(u,v,w) = {} & 3u^2 b_{030} - 3w^2 b_{300} + (6uw - 3u^2) b_{120} + \\ & 6uv b_{021} - 3v^2 b_{102} + 3v^2 b_{012} + \\ & (3w^2 - 6wu) b_{210} - 6wv b_{201} + \\ & (6wv - 6uv) b_{111} \\ D_{\vec{v}}B(u,v,w) = {} & 3v^2 b_{003} - 3w^2 b_{300} - 3u^2 b_{120} + \\ & 3u^2 b_{021} + (6vw - 3v^2) b_{102} + 6vu b_{012} \\ & - 6wu b_{210} + (3w^2 - 6wv) b_{201} + \\ & (6wu - 6uv) b_{111}\end{aligned}$$

B Point and derivative evaluation for PNG1 patches

$$\begin{aligned}S(u,v,w) = {} & \\ & u^3 b_{030}(u,v,w) + v^3 b_{003}(u,v,w) + w^3 b_{300}(u,v,w) + \\ & 3u^2 w b_{120}(u,v,w) + 3u^2 v b_{021}(u,v,w) + 3v^2 w b_{102}(u,v,w) + \\ & 3v^2 u b_{012}(u,v,w) + 3w^2 u b_{210}(u,v,w) + 3w^2 v b_{201}(u,v,w) + \\ & 6wuv b_{111}(u,v,w)\end{aligned}$$

$$\begin{aligned}D_{\vec{u}}S(u,v,w) = {} & \\ & -3\, b_{300} w^2 + 3 b_{030} u^2 + 3\left(D_{\vec{u}} b_{210}\left(u,v,w\right)\right) w^2 u \\ & -6 b_{210}\left(u,v,w\right) wu + 3 b_{210}\left(u,v,w\right) w^2 \\ & +3\left(D_{\vec{u}} b_{120}\left(u,v,w\right)\right) wu^2 - 3 b_{120}\left(u,v,w\right) u^2 \\ & +6 b_{120}\left(u,v,w\right) wu + 3\left(D_{\vec{u}} b_{201}\left(u,v,w\right)\right) w^2 v \\ & -6 b_{201}\left(u,v,w\right) wv + 3\left(D_{\vec{u}} b_{021}\left(u,v,w\right)\right) u^2 v \\ & +6 b_{021}\left(u,v,w\right) uv + 3\left(D_{\vec{u}} b_{102}\left(u,v,w\right)\right) wv^2 \\ & -3 b_{102}\left(u,v,w\right) v^2 + 3\left(D_{\vec{u}} b_{012}\left(u,v,w\right)\right) uv^2 \\ & +3 b_{012}\left(u,v,w\right) v^2 + 6\left(D_{\vec{u}} b_{111}\left(u,v,w\right)\right) wuv \\ & -6 b_{111}\left(u,v,w\right) uv + 6 b_{111}\left(u,v,w\right) wv\end{aligned}$$

$$\begin{aligned}D_{\vec{v}}S(u,v,w) = {} & \\ & -3 b_{300} w^2 + 3 b_{003} v^2 + 3\left(D_{\vec{v}} b_{210}\left(u,v,w\right)\right) w^2 u \\ & -6 b_{210}\left(u,v,w\right) wu + 3\left(D_{\vec{v}} b_{120}\left(u,v,w\right)\right) wu^2 \\ & -3 b_{120}\left(u,v,w\right) u^2 + 3\left(D_{\vec{v}} b_{201}\left(u,v,w\right)\right) w^2 v \\ & -6 b_{201}\left(u,v,w\right) wv + 3 b_{201}\left(u,v,w\right) w^2 \\ & +3\left(D_{\vec{v}} b_{021}\left(u,v,w\right)\right) u^2 v + 3 b_{021}\left(u,v,w\right) u^2 \\ & +3\left(D_{\vec{v}} b_{102}\left(u,v,w\right)\right) wv^2 - 3 b_{102}\left(u,v,w\right) v^2 \\ & +6 b_{102}\left(u,v,w\right) wv + 3\left(D_{\vec{v}} b_{012}\left(u,v,w\right)\right) uv^2 \\ & +6 b_{012}\left(u,v,w\right) uv + 6\left(D_{\vec{v}} b_{111}\left(u,v,w\right)\right) wuv \\ & -6 b_{111}\left(u,v,w\right) uv + 6 b_{111}\left(u,v,w\right) wu\end{aligned}$$