Localized Construction of Curved Surfaces from Polygon Meshes: A Simple and Practical Approach on GPU

Yuen-Shan Leung Charlie C.L. Wang* Yunbo Zhang
Department of Mechanical and Automation Engineering,
The Chinese University of Hong Kong, China

Abstract

We present a method for refining n-sided polygons on a given piecewise linear model by using local computation, where the curved polygons generated by our method interpolate the positions and normals of vertices on the input model. Firstly, we construct a Bézier curve for each silhouette edge. Secondly, we employ a new method to obtain C^1 continuous cross-tangent functions that are constructed on these silhouette curves. An important feature of our method is that the cross tangent functions are produced solely by their corresponding facet parameters. Gregory patches can therefore be locally constructed on every polygons while preserving G^1 continuity between neighboring patches. To provide a flexible shape control, several local schemes are provided to modify the cross-tangent functions so that the sharp features can be retained on the resultant models. Because of the localized construction, our method can be easily accelerated by graphics hardware and fully run on the *Graphics Processing Unit* (GPU).

***Keywords*:** Curved surface; Localized construction; Hardware acceleration; G^1 continuity; Sharp feature.

1 Introduction

In many applications, polygonal meshes have become an important representation of computer generated objects for visual effects. They are simple and versatile, but the lack of continuity between neighboring polygons on the models would be a problem. To improve the visual quality, a common strategy is to subdivide the input mesh surface into a finer mesh, which however quickly increases the consumption of memory and transmission time (either through network or from main memory to the graphics hardware). To overcome this difficulty, an ideal way is that we transmit the relatively coarse mesh during communication, and refine the coarse mesh only when it is about to be displayed. Specifically, we wish to send a coarse input mesh M^0 to the graphics hardware and a smooth dense mesh M^r is then produced in real time through interpolating M^0 by using the parallel computational power of *Graphics Processing Unit* (GPU). The input mesh M^0 is composed of n-sided polygons (with $n \geq 3$). To accomplish this purpose, the proposed method should satisfy the following requirements.

- The resultant surface M^r interpolates the vertices on M^0 as well as their normal vectors.
- M^r possesses G^1 continuity.

*Corresponding Author; E-mail: cwang@mae.cuhk.edu.hk

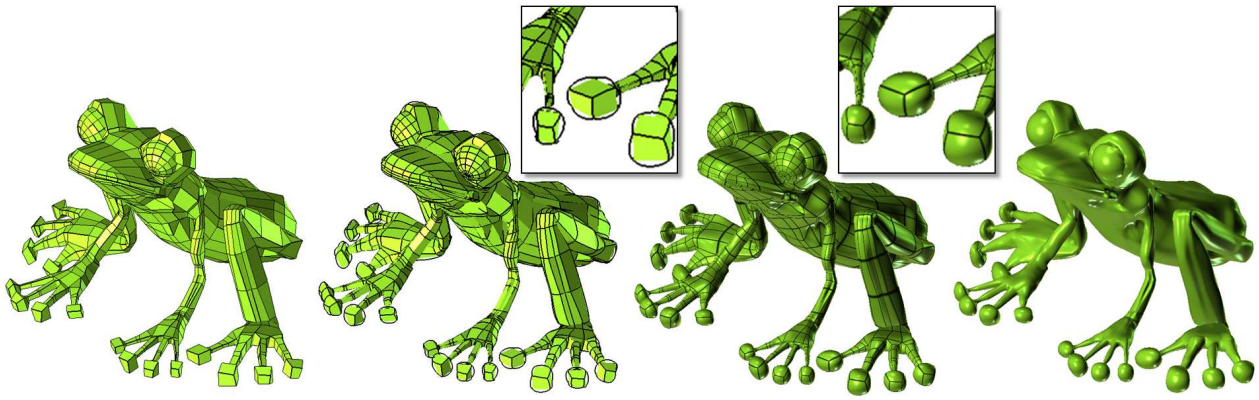

Figure 1: An example of our localized construction of curved polygons. The first image (far left) illustrates an input control mesh. The second image (middle left) shows the silhouette curves generated on the edges of the input control mesh. The boundaries of Gregory patches are displayed by grids in the third image (middle right) and the final (far right) image – the surface of curved polygons generated from the input control mesh.

- The construction procedure must be computed locally on each facet of M^0 so that it can employ the strength of GPUs extensively.

Moreover, unlike the GPU-based subdivision schemes [1, 2, 3] and the visualization based normal processing [4], we intend to form a continuous parametric patch representation on M^r, which is able to evaluate surfaces at arbitrary parametric points – this is important for those non-visualization applications (e.g., distance query and evaluation).

In this paper we offer a local construction approach to create Gregory patches on every n-sided polygon of a coarse mesh M^0. These n-sided Gregory patches require vertices and normal vectors on M^0 and maintain G^1 with their adjoining patches. To build the surface, Bézier curves are first substituted for every silhouette edges. Then, we customize C^1 continuous cross-tangent functions that exclusively rely on boundary curves and are independent of the vicinity facets. Therefore, a Gregory patch can be constructed. Because of this localized scheme, our approach can be processed and accelerated on graphic hardware. Figure 1 shows an example of curved polygons constructed by our approach. To extend the basic construction method, we introduce several schemes to produce sharp features (e.g., the examples shown in Fig.15). The outcome mesh surface M^r with sharp features can be displayed by tessellating the reconstructed Gregory patches on the GPU.

1.1 Related work

The work presented in this paper relates to the existing research in several aspects, including localized parametric patch construction approaches, GPU-based subdivision surface evaluation methods, surface tessellation schemes on the GPU, and n-sided patches researches. They are reviewed below.

Vlachos et al. presented a method in [5] to construct curved point-normal (PN) triangles based only on the three vertices and three vertex normals of given flat triangles. The main principle of their approach is based on substituting the geometry of a three-sided cubic Bézier patch for the triangles's flat geometry, and a quadratically varied normal for Gouraud shading. Visually smooth models can be generated; however, these method does not preserve G^1 continuity across the boundary of neighboring PN triangles. Similar to the PN-triangle approach [5], some work has been done to improve the appearance of a smooth surface generated from triangles [4, 6]. Again, neither of these approaches provides the shape of smooth surfaces preserving G^1 continuity across the boundaries. Another limitation is that only triangular meshes are supported. The approach

of Volino and Magnenat-Thalmann [7] can generate the substitute smooth geometry for n-sided polygons, but still cannot preserve tangent continuity across boundaries. The work presented in this paper also relates to the research of generating surface which interpolate curve networks (e.g., [8, 9, 10]). Nevertheless, in order to borrow the parallel computing power on GPUs, a localized construction approach needs to be exploited.

Subdivision surface provides an effective way to convert an input coarse mesh surface into a fine surface with high resolution. Although a subdivision surface can be composed of an infinite set of polynomial patches (especially essential for complex model), more efforts need to be made to evaluate subdivision surfaces using GPU (ref. [1, 2, 3]). Stam [11] presented a method to exactly evaluate Catmull-Clark surfaces using the framework provided by Halstead et al. [12], but this method can only operate on quadrilateral patches with at most one extraordinary vertex. To apply it to surfaces with triangles or patches with more than one extraordinary vertices, the surface must be subdivided up to two times to provide sufficiently separate extraordinary vertices. Due to the considerable resource requirements, many researchers have investigated methods for approximating subdivision surface. Recently, Loop et al. [13] developed a new method which replaces those irregular patches with a single rational patch that preserves G^1 continuity across the boundary to the surrounding patches. However, only quad/triangle patches are allowed to be constructed in their approaches. A more relevant approach by Christoph et al. [14] is to define a triangle patch with its vertex normals and the three edge neighbor triangles. Although they achieved G^1 continuity on GPU efficiently, the transmitted data still spend a lot of resources on the connectivity information and the surfaces are limited to triangle patches. On the contrary, we employ n-sided Gregory patch interpolation [15, 16] to generate parametric surfaces that interpolate the positions and normal vectors on the vertices of M^0. For an n-sided polygon, only $2n$ vectors are required to perform an evaluation on a smooth surface – when having sharp edges, at most $4n$ vectors plus a 2-bit number are needed. This contributes to a significant improvement in the speed of data transmission.

Another relevant research line is the adaptive tessellation method on GPU [17, 18, 19, 20]. Dyken et al. [17] presented an algorithm for detecting and extracting the silhouette edges of a triangle mesh in real time using GPU. The smooth silhouette is reconstructed through a continuous blend between Bézier patches with varying level of details. Their recent work in [20] introduced an adaptive tessellation scheme for surfaces consisting of parametric patches. However, different from ours (for n-sided polygons), their approaches only work on triangular mesh surfaces. The method of Boubekeur and Schlick [18] was only for mesh refinement on triangles too. The recent work of Schwarz and Stamminger [19] provides a new framework for performing on-the-fly crack-free adaptive tessellation of surface primitives completely on the GPU. Their implementation is based on CUDA of nVIDIA whereas the surface reconstructed in our approach is generated by GL shading language, which can be supported by graphics hardware with Shader Model 4.0 in a variety of brands.

Regarding to the study on n-sided surfaces, many researchers have put much effort to this area for many years. Piegl et al. [21] presented an algorithm to fill an n-sided region with G^k continuous NURBS patches, whereas Wang et al. [22] used Varady patches to fit an arbitrary n-sided region. Other approaches [23, 24] described techniques to insert a region with multiple patches. The smoothness of the filled patches along the shared edges requires some special treatment. Basically, the computations involved in the methods of [21, 22, 23, 24] used iterative algorithms or constrained optimization, which are too computational expensive to fit in the pipeline of shader programs running on the graphics hardware. Our method aims at improving visual quality with as less transmitted data as possible. Therefore we provide an algorithm which is able to tackle the twist incompatibility and boundary incompatibility in a local manner and result in G^1 continuous surfaces.

Prior to this work, we presented a Gregory patch based smooth force rendering method in [25]. The approach presented in this paper is extended from that algorithm by 1) providing a formal proof and analysis about the correctness of the basic local construction scheme, 2) investigating the

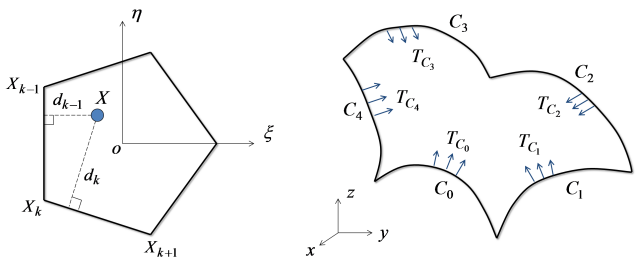

Figure 2: Gregory patch interpolation: (left) the parametric domain Γ_G of a Gregory Patch with five sides, and (right) the Gregory patch defined by the boundary silhouette curves $\mathbf{C}_i$ and the cross tangent functions $\mathbf{T}_{C_i}$ on $\mathbf{C}_i$s.

extended schemes for the flexible shape control and 3) developing a GPU-based algorithm.

1.2 Gregory patch interpolation

To construct G^1 continuous surfaces for input polygons, we first construct a Gregory corner interpolator function for every corner of an n-sided polygon. Second, the final surface of this polygon is blended as a weighted sum of the n functions (ref.[15, 16]).

Let $\mathbf{P}(u)(0 \leq u \leq 1)$ and $\mathbf{Q}(v)(0 \leq v \leq 1)$ be two regular curves in $\Re^3$ with $\mathbf{P}(0) = \mathbf{Q}(0)$, and $\mathbf{T}_P(u)(0 \leq u \leq 1)$ and $\mathbf{T}_Q(v)(0 \leq v \leq 1)$ be two C^1 continuous vector functions associated with the curves satisfying $\mathbf{T}_P(0) = \mathbf{Q}'(0)$ and $\mathbf{T}_Q(0) = \mathbf{P}'(0)$, the Gregory corner interpolator of the four functions, $\mathbf{P}(u)$, $\mathbf{Q}(v)$, $\mathbf{T}_P(u)$ and $\mathbf{T}_Q(v)$, is a surface in $\Re^3$ defined by

$$\begin{aligned}\mathbf{r}(u,v) &= -\mathbf{P}(0) - v\mathbf{T}_P(0) - u\mathbf{T}_Q(0) + \mathbf{P}(u) + v\mathbf{T}_P(u) \\ &+\mathbf{Q}(v) + u\mathbf{T}_Q(v) - uv(v\mathbf{T}'_P(0) + u\mathbf{T}'_Q(0))/(u+v)\end{aligned} \tag{1}$$

The Gregory corner interpolator function $\mathbf{r}(u,v)$ agrees with $\mathbf{P}(u)$ and $\mathbf{Q}(v)$ along the two sides (i.e. $\mathbf{r}(u,0) = \mathbf{P}(u)$ and $\mathbf{r}(0,v) = \mathbf{Q}(v)$). Also, its partial derivatives with respect to u and v agree with $\mathbf{T}_P(u)$ and $\mathbf{T}_Q(v)$ along the respective sides as $\mathbf{r}_u(u,0) = \mathbf{T}_P(u)$ and $\mathbf{r}_v(0,v) = \mathbf{T}_Q(v)$.

The parametric domain of a Gregory Patch with n sides is defined as a unit length regular n-gon in the $\xi - \eta$ domain (as shown in Fig.2). We define the domain of a Gregory patch as Γ_G, where each sub-patch Γ_G^k contains a corner $\mathbf{X}_k$ $(k = 0, 1, ..., n-1)$ and the corners of Γ_G are placed in the anti-clockwise order. Given a point (ξ, η) in the parametric space, when computing its position defined by the k-th Gregory corner $\mathbf{r}_k(u_k, v_k)$, the parameters (u_k, v_k) of the point are defined as

$$(u_k, v_k) = \left(\frac{d_{k-1}}{d_{k-1} + d_{k+1}}, \frac{d_k}{d_{k-2} + d_k}\right) \tag{2}$$

with d_k being the perpendicular distance from (ξ, η) to the edge $\mathbf{X}_k\mathbf{X}_{k+1}$. The final position of (ξ, η) on the Gregory patch is obtained by blending its positions defined by all the corner interpolators as

$$\mathbf{G}(\xi, \eta) = \sum_{k=0}^{m-1} w_k(\xi, \eta)\mathbf{r}_k\left(u_k(\xi, \eta), v_k(\xi, \eta)\right) \tag{3}$$

where the weight function is

$$w_k(\xi, \eta) = \prod_{j \neq k-1, k} d_j^2 / \sum_{l=0}^{m-1} \prod_{j \neq l-1, l} d_j^2. \tag{4}$$

The Gregory patch defined in this way interpolates all the boundary curves and the cross-tangent functions defined on them. The difficulty left is how to construct the boundary curves and the cross-tangent functions in a local manner so that G^1 continuity can be preserved between adjacent patches. We solve this in the section 2.1.

1.3 Main features

The main features of our proposed method in this paper are as follows.

- A localized construction method to generate smooth parametric surface patches interpolating (instead of approximating) the vertices and the normal vectors of an input coarse mesh. The constructed Gregory patches preserve G^1 continuity between neighboring patches. The technical novelty here comes from the localized construction of silhouette curves and cross-tangent functions for preserving G^1 continuity. More than that, the basic scheme can be extended to provide flexible control on the cross-tangent functions, so that sharp features can be retained and the shape of the reconstructed surface patches can be edited.
- The amount of information needs to be sent is less than other relevant approaches in literature. For an n-sided polygon, only $2n$ vectors are required to perform evaluation on a smooth surface. When having sharp edges, at most $4n$ vectors plus a 2-bit number are needed.
- Our method works on n-sided polygons while most of the existing local construction approaches in literature are designed for quadrangles and/or triangles.

These main features result in a highly parallel algorithm that suits the production of curved polygons on graphics hardware.

The rest of the paper is organized as follows. The local scheme of constructing curved polygons is introduced in section 2, and the important properties of the method are studied. The algorithm to convert a triangular mesh surface into a polygonal model with less number of facets is also developed in section 2. The method to reconstruct sharp features on the curved polygons is presented in section 3. Section 4 describes the implementation details of our method on the GPU, section 5 gives the experimental results, and lastly our paper ends with the conclusion section.

2 Local Construction of Curved Polygon

2.1 Basic construction scheme

The construction of curved polygons consists of two steps. Firstly, we construct the silhouette curve for each edge on the given polygon according to the positions and normals at vertices. Secondly, we develop a new method to obtain C^1 continuous cross-tangent functions on these silhouette curves.

2.1.1 Silhouette curve interpolation

Different from [25] that uses Hermite curves, cubic Bézier curve is employed here to build the silhouette edges on a given n-sided polygon $P = \{\mathbf{p}_i\}$ with $(i = 0, \cdots, n-1)$. This is because the evaluation of a Bézier curve can be conducted by the famous de Casteljau linear interpolation, which is much more efficient than the polynomial computation on a Hermite curve (ref. [26]). Every vertex $\mathbf{p}_i$ is equipped with a vertex normal vector $\mathbf{n}_i$, which also defines a surface tangent plane at $\mathbf{p}_i$. For two neighboring vertices $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ on P, the silhouette curve $\mathbf{C}(t)$ is defined by the cubic Bézier curve with four control points $\{\mathbf{p}_i, \mathbf{q}_i, \mathbf{q}_{i+1}, \mathbf{p}_{i+1}\}$. To generate a surface patch $\mathbf{G}_P$ preserving G^1 continuity across the boundary, any curve on $\mathbf{G}_P$ passing through $\mathbf{p}_i$ must have the curve tangent

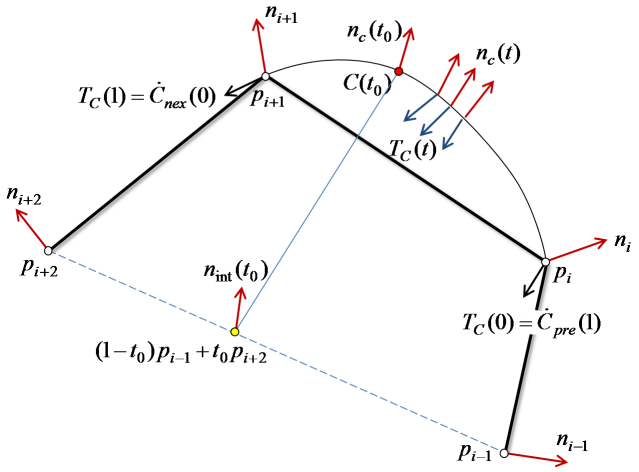

Figure 3: The illustration of the construction of cross tangent functions.

at $\mathbf{p}_i$ in the surface tangent plane $(\mathbf{p}-\mathbf{p}_i)\cdot\mathbf{n}_i=0$. Assume that $\mathbf{t}_i$ and $\mathbf{t}_{i+1}$ are tangent vectors of $\mathbf{C}(t)$ at $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$, they must satisfy the criteria that $\mathbf{t}_i\cdot\mathbf{n}_i=0$ and $\mathbf{t}_{i+1}\cdot\mathbf{n}_{i+1}=0$. According to the first derivative on the cubic Bézier curve that $\mathbf{t}_i=3(\mathbf{q}_i-\mathbf{p}_i)$, we have that

$$\mathbf{q}_i=\mathbf{p}_i+\frac{1}{3}\|\mathbf{p}_i\mathbf{p}_{i+1}\|\frac{\mathbf{n}_i\times((\mathbf{p}_i\mathbf{p}_{i+1})\times\mathbf{n}_e)}{\|\mathbf{n}_i\times((\mathbf{p}_i\mathbf{p}_{i+1})\times\mathbf{n}_e)\|} \tag{5}$$

where $\mathbf{n}_e=\frac{1}{2}(\mathbf{n}_i+\mathbf{n}_{i+1})$ is the average normal vector. Similarly, $\mathbf{q}_{i+1}$ is defined by

$$\mathbf{q}_{i+1}=\mathbf{p}_{i+1}-\frac{1}{3}\|\mathbf{p}_i\mathbf{p}_{i+1}\|\frac{\mathbf{n}_{i+1}\times((\mathbf{p}_i\mathbf{p}_{i+1})\times\mathbf{n}_e)}{\|\mathbf{n}_{i+1}\times((\mathbf{p}_i\mathbf{p}_{i+1})\times\mathbf{n}_e)\|} \tag{6}$$

2.1.2 Cross-tangent function

While Bézier curves define the boundaries of a patch, the cross tangent function $\mathbf{T}_C(t)$ on a boundary curve $\mathbf{C}(t)$ tells the developing direction of the patch across the edge $\mathbf{p}_i\mathbf{p}_{i+1}$ (i.e., $\mathbf{C}(t)$). We construct the cross-tangent functions by using the following method as illustrated in Fig.3.

Firstly, we equip a normal function $\mathbf{n}_c(t)$ onto the silhouette curve $\mathbf{C}(t)$ as

$$\mathbf{n}_c(t)=\mathbf{C}'(t)\times(((1-t)\mathbf{n}_i+t\mathbf{n}_{i+1})\times\mathbf{p}_i\mathbf{p}_{i+1}), \tag{7}$$

which also satisfies $\mathbf{n}_c(0)=\mathbf{n}_i$ and $\mathbf{n}_c(1)=\mathbf{n}_{i+1}$. As will be proved later, if the tangent function $\mathbf{T}_C(t)$ has $\mathbf{T}_C(t)\cdot\mathbf{n}_c(t)\equiv 0$ $(\forall t\in[0,1])$ been satisfied on both sides of $\mathbf{C}(t)$, G^1 continuous geometry is generated.

Making $\mathbf{T}_C(t)$ perpendicular to $\mathbf{n}_c(t)$ is easy. A more difficult task is to make $\mathbf{T}_C(0)=\mathbf{C}'_{pre}(1)$ and $\mathbf{T}_C(1)=\mathbf{C}'_{nex}(0)$, where $\mathbf{C}_{pre}(t)$ and $\mathbf{C}_{nex}(t)$ are the silhouette curves defined on $\mathbf{p}_{i-1}\mathbf{p}_i$ and $\mathbf{p}_{i+1}\mathbf{p}_{i+2}$ respectively. As shown in Fig.4, if the vector function $\mathbf{T}_P(u)$ and $\mathbf{T}_Q(u)$ are not correctly defined, the constructed Gregory patches will NOT preserve G^1 continuity across the boundary. We first construct an interpolation function related to the vertex normal vectors and the face normal vector of the polygon P. We can employ a linear interpolation function

$$\mathbf{n}^L_{int}(t)=\frac{(1-t)}{2}(\mathbf{n}_{i-1}+\mathbf{n}_i)+\frac{t}{2}(\mathbf{n}_{i+1}+\mathbf{n}_{i+2}) \tag{8}$$

or a quadratic interpolation function

$$\mathbf{n}^Q_{int}(t)=(2t-1)(t-1)\frac{\mathbf{n}_{i-1}+\mathbf{n}_i}{2}+4t(1-t)\alpha\mathbf{n}_f+t(2t-1)\frac{\mathbf{n}_{i+1}+\mathbf{n}_{i+2}}{2} \tag{9}$$

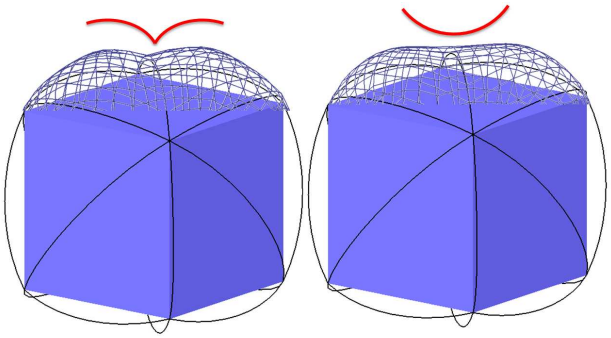

Figure 4: The vector functions $\mathbf{T}_P(u)$ and $\mathbf{T}_Q(u)$ are important for preparing G^1 continuity across the boundary: (left) with incorrect cross-tangent functions $\mathbf{T}_P(u)$ and $\mathbf{T}_Q(u)$; (right) with functions defined in Eq.(11) – G^1 continuity is guaranteed.

with $\mathbf{n}_f$ being the face normal of the polygon and α being a shape factor to control the distribution of $\mathbf{T}_C(t)$. The strategy of selecting α is discussed later. The major drawback of $\mathbf{n}^L_{int}(t)$ is that it degenerates when $\sum_{j=i-1}^{i+2}\mathbf{n}_j = 0$. By Eq.(9), we have $\mathbf{n}^Q_{int}(0) = \frac{1}{2}(\mathbf{n}_{i-1}+\mathbf{n}_i)$, $\mathbf{n}^Q_{int}(\frac{1}{2}) = \alpha\mathbf{n}_f$ and $\mathbf{n}^Q_{int}(1) = \frac{1}{2}(\mathbf{n}_{i+1}+\mathbf{n}_{i+2})$. The direction of the tangent function is then defined by

$$\mathbf{t}_c(t) = \mathbf{n}_{int}(t) \times ((1-t)\mathbf{p}_{i-1} + t\mathbf{p}_{i+2} - \mathbf{C}(t)) \times \frac{\mathbf{n}_c(t)}{\|\mathbf{n}_c(t)\|}. \tag{10}$$

After considering the magnitude, the final cross tangent function defined on $\mathbf{C}(t)$ is

$$\mathbf{T}_C(t) = \|(1-t)\mathbf{p}_{i-1} + t\mathbf{p}_{i+2} - \mathbf{C}(t)\|\frac{\mathbf{t}_c(t)}{\|\mathbf{t}_c(t)\|}. \tag{11}$$

It is easy to prove that $\mathbf{T}_C(t)$ and $\mathbf{C}'(t)$ are perpendicular to $\mathbf{n}_c(t)$, and we have $\mathbf{T}_C(0) = \mathbf{C}'_{pre}(1)$ and $\mathbf{T}_C(1) = \mathbf{C}'_{nex}(0)$. Also, the function $\mathbf{T}_C(t)$ constructed in this way is a C^1 continuous function.

2.2 Property analysis

Several properties of the basic construction scheme of curved polygon will be analyzed in this section.

Property 1 The silhouette curve defined by a cubic Bézier curve $\mathbf{C}(t)$ using the control polygon $\{\mathbf{p}_i, \mathbf{q}_i, \mathbf{q}_{i+1}, \mathbf{p}_{i+1}\}$ is a planar curve.

Analysis It is easy to find from Eqs.(5) and (6) that all the control points are on the plane Ω defined by sweeping the vector $\mathbf{p}_i\mathbf{p}_{i+1}$ along the direction of $\mathbf{n}_e$. As any point $\mathbf{C}(t_0)$ ($t_0 \in [0,1]$) on the Bézier curve can be obtained by a sequence of the linear interpolations of the control points, $\mathbf{C}(t_0)$ must be a point on the plane Ω. Thus, $\mathbf{C}(t)$ is a planar curve.

According to Eqs.(5) and (6), when the criterion $(\mathbf{p}_i\mathbf{p}_{i+1}\cdot\mathbf{n}_i)(\mathbf{p}_{i+1}\mathbf{p}_i\cdot\mathbf{n}_{i+1}) > 0$ is satisfied, two points $\mathbf{q}_i$ and $\mathbf{q}_{i+1}$ must be located at the same side of the line $\mathbf{p}_i\mathbf{p}_{i+1}$. This seldom gives a Bézier curve with S-shape. Moreover, As the magnitude of two end tangent $\dot{\mathbf{C}}(0)$ and $\dot{\mathbf{C}}(1)$ on the Bézier curve are

$$\dot{\mathbf{C}}(0) = 3(\mathbf{q}_i - \mathbf{p}_i) \text{ and } \dot{\mathbf{C}}(1) = 3(\mathbf{p}_{i+1} - \mathbf{q}_{i+1}),$$

the magnitude of these endpoint tangents, $|\dot{\mathbf{C}}(0)| = |\dot{\mathbf{C}}(1)| = \|\mathbf{p}_i\mathbf{p}_{i+1}\|$, will not let the curve generate self-intersection. More discussion about the effects of the magnitudes of endpoint tangents on the shape of a cubic polynomial curve can be found in [27].

Proposition 1 G^1 continuity is preserved around a vertex by the basic construction scheme of silhouette curves and Gregory patches.

Proof First of all, for the silhouette curve $\mathbf{C}(t)$ adjacent to the vertex $\mathbf{p}_i$, its tangent vector $\dot{\mathbf{C}}(0)$ at $\mathbf{p}_i$ is $3\mathbf{p}_i\mathbf{q}_i$. By Eq.(5), $\mathbf{q}_i$ is located on the tangent plane $(\mathbf{p} - \mathbf{p}_i) \cdot \mathbf{n}_i = 0$. Thus, the tangent vector $\dot{\mathbf{C}}(0)$ is also on the tangent plane.

Now we need to figure out the following issue – how about those non-boundary curves on $\mathbf{G}_P$ passing through $\mathbf{p}_i$? From Eqs.(2-4), it is not difficult to know that, for the surface point at $\mathbf{p}_i$, its surface is only defined by the corner interpolator at $\mathbf{X}_i$ (i.e., Eq.(1)). If there is a non-boundary curve $\mathbf{D}(t)$ on this corner patch with $\mathbf{D}(0) = \mathbf{P}(0)$, $\dot{\mathbf{D}}(0)$ should lie on the tangent plane at $\mathbf{P}(0)$ defined by the tangent vectors $\dot{\mathbf{P}}(0)$ and $\dot{\mathbf{Q}}(0)$. With the boundary curve $\mathbf{C}(t)$ defined by our method stated above, this tangent plane is the same as the tangent plane defined by the vertex normal $\mathbf{n}_i$ at $\mathbf{p}_i$. Therefore, G^1 continuity at the corner vertices is maintained.

Q.E.D.

Proposition 2 G^1 continuity is preserved cross the boundary of two neighboring patches that are constructed separately by the basic construction scheme.

Proof Considering about two Gregory patches that are constructed separately but sharing the same edge $\mathbf{p}_i\mathbf{p}_{i+1}$, G^0 continuity is preserved between them as the silhouette curve since $\mathbf{p}_i\mathbf{p}_{i+1}$ is constructed only by the positions of $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ as well as the predefined normal vectors on them. In other words, the separately constructed curves will have the same set of control points (see Eqs.(5) and (6)).

From Eq.(11), we know that the directions of cross-tangent vectors interpolated by the Gregory patch on a boundary curve $\mathbf{C}(t)$ is defined by Eq.(10), which keeps perpendicular to the normal function $\mathbf{n}_c(t)$ defined in Eq.(7). Similar to the silhouette curve function $\mathbf{C}(t)$, $\mathbf{n}_c(t)$ is solely defined by the Hermite data points $(\mathbf{p}_i, \mathbf{n}_i)$ and $(\mathbf{p}_{i+1}, \mathbf{n}_{i+1})$. Therefore, these two Gregory patches sharing the edge $\mathbf{p}_i\mathbf{p}_{i+1}$ will have a consistent normal function define on the silhouette curve. Since the cross-tangent functions keep perpendicular to the normal function, the cross tangents on two sides of $\mathbf{C}(t)$ are coplanar with $\dot{\mathbf{C}}(t)$. In other word, a condition similar to G^1 continuity preservation on assembled parametric surfaces [28] is satisfied. Thus, we prove that G^1 continuity is preserved cross the boundary of two neighboring patches constructed by the basic construction scheme separately.

Q.E.D.

Property 2 The construction of a Gregory patch for a n-sided polygon needs only an input with $2n$ vectors.

Analysis The construction of a Gregory patch for a given n-sided polygon in this way only needs to locally access the vertices on the polygon and the normal vectors assigned on them, which avoids visiting the neighboring polygons. This makes parallel implementation easy. When computing on the GPU, only $2n$ vectors (n positions plus n normal vectors) are communicated between the main memory and the graphics hardware which is the bottleneck for data transmission. In fact, the amount of communicated data can be further reduced as only a unit vector is required for normal, which means that we can transfer only two components of a normal vector plus the sign for the third component. To the best of our knowledge, this is an approach requesting the minimal amount of data communication comparing to prior works.

By the localized construction scheme presented in this section, we can generate G^1 continuous Gregory patches that interpolate the positions and normals of vertices on a given polygonal mesh surface (see the frog model in Fig.1). However, in some applications, sharp features should be

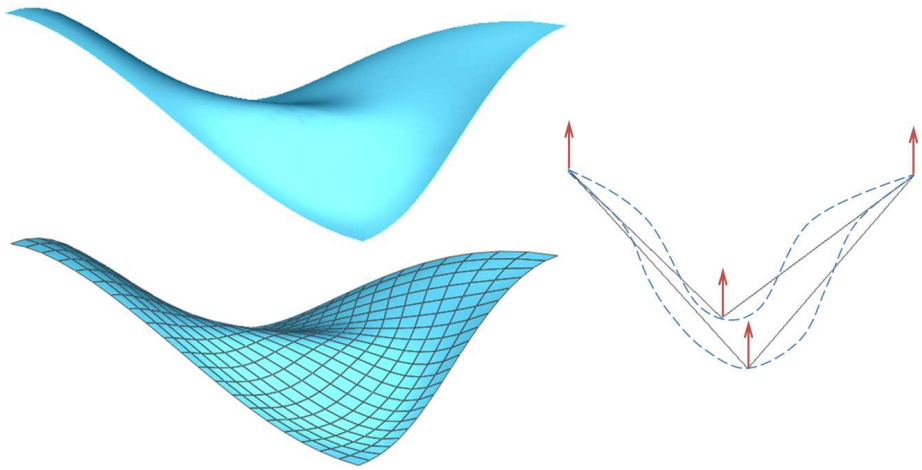

Figure 5: An example for the curved polygon constructed on a non-planar polygon, where we may wish to generate a surface folded in a different direction (as specified by the dash lines).

reserved at some places. After analyzing some important properties of our method in section 2.3, we will introduce methods in section 3 to extend our basic construction scheme to acquire this function. Several shape control methods are presented.

2.3 Discussions

Basically, the aforementioned construction scheme of curved polygons works well on a mesh model consists of n-sided polygons. However, poor results may be generated on some special cases, which will be discussed below. We also provide a solution to generate a mesh model that rarely presents such cases leading to poor results.

First of all, the basic construction scheme assumes that all the polygons on the given model are nearly planar. When using quadratic interpolation function $\mathbf{n}_{int}^{Q}(t)$ in Eq.(9) to construct the direction of the tangent function $\mathbf{t}_c(t)$ in Eq.(10), the face normal $\mathbf{n}_f$ of the polygon is needed. For a polygon with n vertices, we compute the face normal (not normalized) by

$$\mathbf{n}_f^* = \frac{1}{n}\sum_i \frac{\mathbf{q}_i\mathbf{q}_{i+1} \times \mathbf{q}_i\mathbf{q}_{i-1}}{\|\mathbf{q}_i\mathbf{q}_{i+1} \times \mathbf{q}_i\mathbf{q}_{i-1}\|}. \tag{12}$$

Then, $\mathbf{n}_f$ is obtained by $\mathbf{n}_f^*/\|\mathbf{n}_f^*\|$. When the positions of vertices are far from planar, although we can still obtain the face normal $\mathbf{n}_f$ by the above method, it however introduces a lot of uncertainty on the final resultant surface – i.e., the resultant surface may be quite different to what you wish to have (see Fig.5 for an example).

Secondly, we always wish the polygons be convex when they are nearly planar. If the concave shape is shown at a vertex $\mathbf{p}_i$, it means that $(\mathbf{p}_i\mathbf{p}_{i+1} \times \mathbf{p}_i\mathbf{p}_{i-1}) \cdot \mathbf{n}_i < 0$ or $(\mathbf{p}_i\mathbf{p}_{i+1} \times \mathbf{p}_i\mathbf{p}_{i-1}) \cdot \mathbf{n}_f < 0$. The defect of having such a concave vertex is twofold. On one side, this will lead to an inverse contribution to the face normal comparing to other convex vertices (i.e., the cross-product at $\mathbf{p}_i$ in Eq.(12) points to an inverse direction). On the other aspect, this will make some parts of $\mathbf{t}_c(t)$ point outward (i.e., leaving the boundary of the surface patch). This wrong direction of the cross-tangent vector will generate an inversely folded surface (e.g., see Fig.6).

Thirdly, the shape of triangular surface constructed by Gregory patch sometimes have some unwanted bump generated in the middle of the surface patch. The bump shape is more significant

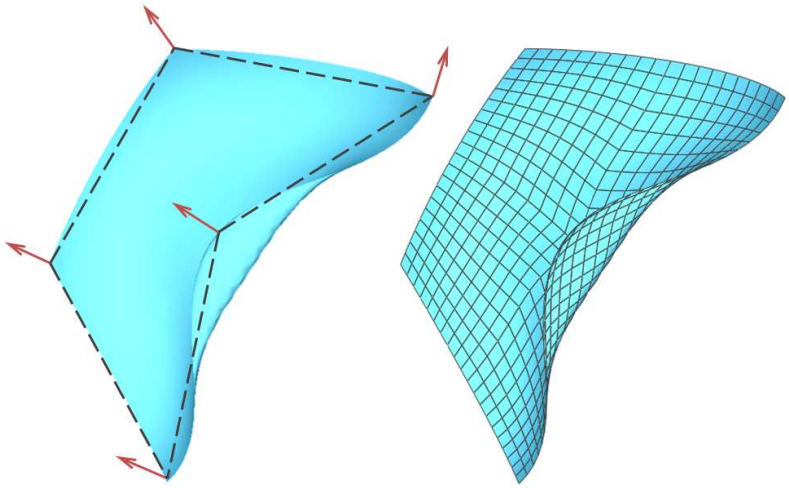

Figure 6: An example for the curved polygon constructed on a non-convex polygon, where the surface patch is inversely folded at the concave vertex.

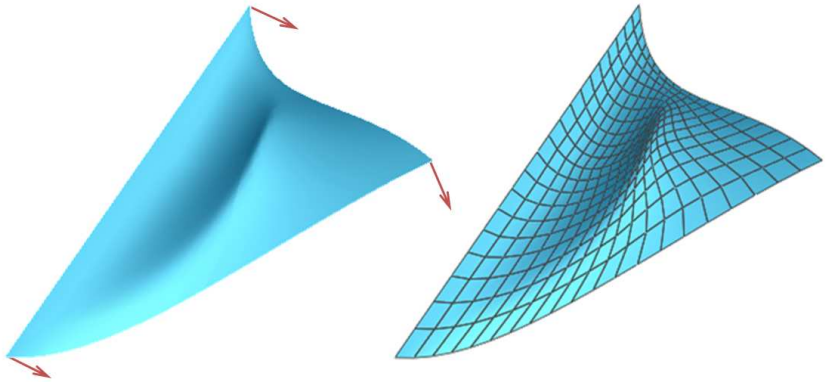

Figure 7: An example for the curved polygon constructed on a triangle, where the surface patch holds an unwanted bump in the middle.

when the normal vectors at three vertices of the triangle are much different from each other (see Fig.7). The reason for generating such bump shape is that the directions of cross-tangent function (defined in Eq.(10)) on neighboring silhouette curves are coupled since there is only three vertices on a polygon.

Lastly, there is a case which will turn down our method to construct the silhouette curve. In Eqs.(5) and (6), when $\mathbf{n}_e = \frac{1}{2}(\mathbf{n}_i + \mathbf{n}_{i+1}) = 0$, the silhouette curve would fail to build. To solve this problem, we will use different weights on two normal vectors when $\frac{1}{2}(\mathbf{n}_i + \mathbf{n}_{i+1}) = 0$.

$$\mathbf{n}_e = \beta\mathbf{n}_i + (1-\beta)\mathbf{n}_{i+1} \tag{13}$$

The optimal value of β is chosen to have a small perturbation around $\frac{1}{2}$ so that will not make $\mathbf{n}_e$ vanished. Under this definition, Eqs.(8) and (9) need to be redefined using the same weight as

$$\mathbf{n}^L_{int}(t) = (1-t)(\beta\mathbf{n}_{i-1} + (1-\beta)\mathbf{n}_i) + t(\beta\mathbf{n}_{i+1} + (1-\beta)\mathbf{n}_{i+2}) \tag{14}$$

and

$$\mathbf{n}^Q_{int}(t) = (2t-1)(t-1)(\beta\mathbf{n}_{i-1} + (1-\beta)\mathbf{n}_i) + 4t(1-t)\alpha\mathbf{n}_f + t(2t-1)(\beta\mathbf{n}_{i+1} + (1-\beta)\mathbf{n}_{i+2}). \tag{15}$$

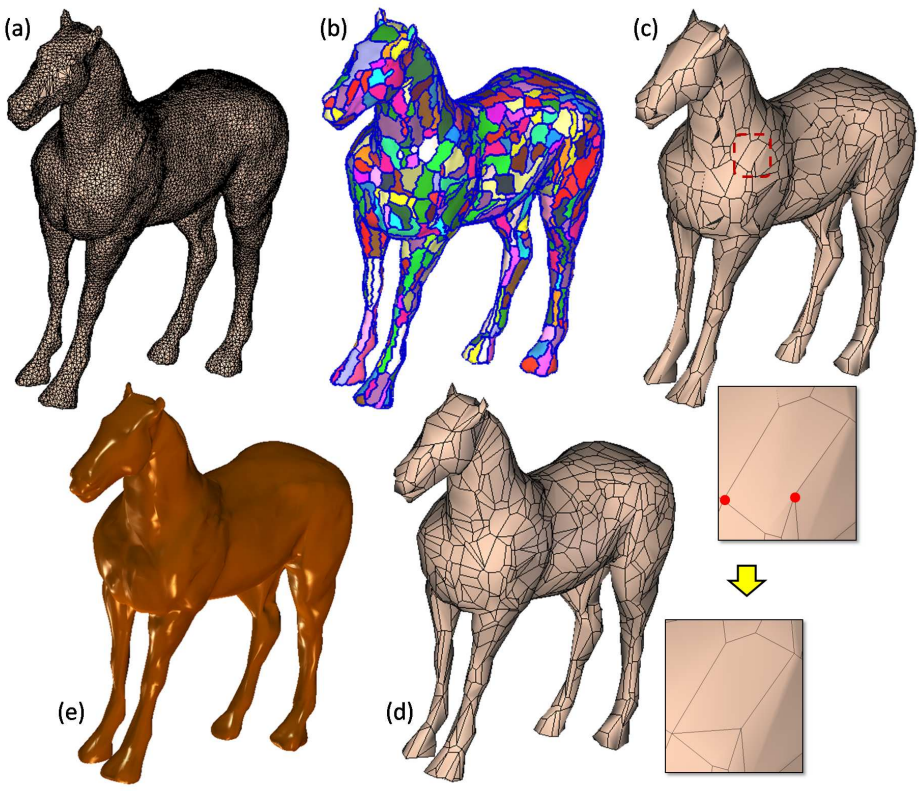

Figure 8: An example of using VSA algorithm [29] to generate the simplified polygonal model from a given freeform object represented by triangular meshes: (a) the given triangular mesh model, (b) the segmentation result of VSA, (c) the polygons generated by VSA segmentation, (d) the concave vertices are eliminated by repeatedly applying the splitting operation, and (e) the reconstructed freeform model by using curved polygons proposed in this paper.

2.4 Algorithm for generating n-sided polygons

A majority of freeform objects are represented by triangular mesh surfaces. We have developed an algorithm to simplify a triangular mesh surface M_T into a polygonal mesh model M_P that can have a good shape reconstructed by our curved polygon approach. As analyzed in above subsection, the polygonal mesh model M_P is expected to have 1) nearly planar polygons, 2) convex polygons and 3) polygons with more than three edges.

To satisfy the requirements of polygons on the resultant model, we first apply the Variational Shape Approximation (VSA) approach [29] to convert M_T into a model M_V with user specified number of nearly planar polygons. Then, the polygons on M_V are checked one by one to see if there is a concave vertex. When a concave vertex $\mathbf{v}$ is found, it will be eliminated by adding an edge from $\mathbf{v}$ to its closest non-neighboring vertex to split the polygon. Repeatedly applying this simple splitting algorithm will result in a polygonal model consists of nearly planar convex polygons. An example about how to generate such polygonal models is shown in Fig.8.

3 Flexible Shape Control

We provide several ways to flexibly adjust the appearance of models. The shape control schemes introduced here are rather simple and competent, which merely involve some minor changes of the basic construction scheme introduced above. The extended construction schemes are detailed below.

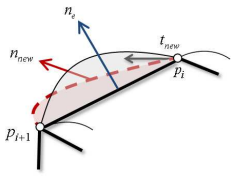

Figure 9: The plane holding the silhouette curve can be reoriented by assigning a new edge normal (extended scheme 1).

The shape modification is based on the way how we construct the silhouette curve and the cross tangent function.

3.1 Possibility of extending the basic scheme

In the basic construction scheme, the silhouette curve generated by the Bézier curve interpolating $\mathbf{p}_i$, $\mathbf{p}_{i+1}$ and preserving G^1 continuity at them is a curve in the plane defined by $\mathbf{p}_i\mathbf{p}_{i+1}$ and the sweeping vector $\mathbf{n}_e$, which is an average of $\mathbf{n}_i$ and $\mathbf{n}_{i+1}$. This is the first factor that can be adjusted below for the flexible shape control.

After that, the cross tangent function is constructed by retaining the tangent vector $\mathbf{t}_c(t_0)$ (that is perpendicular to $\mathbf{n}_c(t_0)$) in the plane defined by sweeping the line segment $(\mathbf{p}_v - \mathbf{C}(t_0))$ along $\mathbf{n}_{int}(t_0)$, where $\mathbf{p}_v$ is a virtual vertex obtained by the linear interpolation between $\mathbf{p}_{i-1}$ and $\mathbf{p}_{i+2}$. See Fig.3 for the illustration. The direction of the cross-tangent function is the second factor that can be adjusted for the flexible shape control.

3.2 Reorienting silhouette curve (extended scheme 1)

In the basic scheme, the silhouette curve $\mathbf{C}(t)$ on $\mathbf{p}_i\mathbf{p}_{i+1}$ is defined in the plane swept from $\mathbf{p}_i\mathbf{p}_{i+1}$ along the vector $\mathbf{n}_e = \frac{1}{2}(\mathbf{n}_i + \mathbf{n}_{i+1})$ (see Fig.9). The orientation of $\mathbf{C}(t)$ can be adjusted by assigning a new edge normal $\mathbf{n}_{new}$ to replace $\mathbf{n}_e$ in Eqs.(5) and (6). The rest of the steps in the basic scheme remain unchanged. By this modification, G^1 continuity is still preserved. However, the amount of data to be communicated is increased from $2n$ vectors to $3n$ vectors if edge normals are assigned to all the edges of input polygons.

3.3 Straight silhouette (extended scheme 2)

There are several methods to generate a straight silhouette on an edge $\mathbf{p}_i\mathbf{p}_{i+1}$ of a given n-sided polygon. The simplest way is to let $\mathbf{n}_i = \mathbf{n}_{i+1} = \mathbf{n}_v$ (see Fig.10) when using Eqs.(5) and (6) to compute the control points of the silhouette curve. When using the same normal, same tangent vectors are generated so that the Bézier curve is a straight line segment. When computing the cross tangent function $\mathbf{T}_C(t)$ across this edge, if G^1 continuity is expected, the original vertex normals $\mathbf{n}_i$ and $\mathbf{n}_{i+1}$ are employed in Eqs.(7)-(11). Note that, at the two endpoints of the straight silhouette, only G^0 continuity is achieved. When they are replaced by $\mathbf{n}_v$ for generating $\mathbf{T}_C(t)$, a straight crease is produced on the resultant surface.

3.4 Crease (extended scheme 3)

A crease can be generated on the edge $\mathbf{p}_i\mathbf{p}_{i+1}$ without changing the shape of the silhouette curve $\mathbf{C}(t)$ defined by the basic scheme. The basic idea is to break the condition for G^1 continuity across the silhouette curve $\mathbf{C}(t)$. To do that, we replace $\mathbf{n}_i$ and $\mathbf{n}_{i+1}$ employed in Eq.(7) by (see Fig.11)

$$\mathbf{m}_i = (1 - \gamma_s)\mathbf{n}_i + \gamma_s \mathbf{p}_i\mathbf{p}_{i-1}, \tag{16}$$

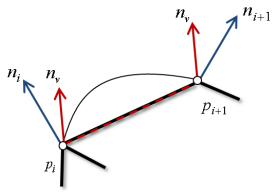

Figure 10: Generation of straight silhouette (extended scheme 2).

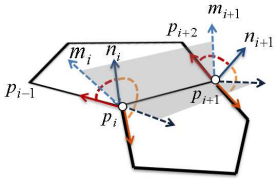

Figure 11: Generation of crease on the silhouette curve (extended scheme 3).

and

$$\mathbf{m}_{i+1} = (1-\gamma_e)\mathbf{n}_{i+1} + \gamma_e \mathbf{p}_{i+1}\mathbf{p}_{i+2}, \tag{17}$$

where γ_s and γ_e are two factors to control the sharpness of this crease. Similar adjustment should be given to the Gregory patch on the other side of this edge as well. Therefore, two patches sharing the same silhouette curve will not have coplanar cross-tangent vectors, which makes a crease at the silhouette curve.

3.5 Silhouette adjacent to crease (extended scheme 4)

When a silhouette has one vertex on crease (the silhouette itself is *NOT* a crease), some modifications must be given to generate a satisfactory shape. For example, as shown in Fig.12, if the vertex $\mathbf{p}_{i+1}$ is on a crease, the new edge vector $\mathbf{n}_s$ is employed in Eq.(6) to replace $\mathbf{n}_{i+1}$ when constructing the silhouette curve $\mathbf{C}(t)$. Letting $\mathbf{n}_s$ be the average of the left and right polygons' normal vectors along the edge $\mathbf{p}_i\mathbf{p}_{i+1}$ is a good choice. When building the cross tangent function $\mathbf{T}_C(t)$ in Eqs.(7)-(11), $\mathbf{n}_s$ is also adopted to replace $\mathbf{n}_{i+1}$.

3.6 Smooth patch bulging (extended scheme 5)

As mentioned before, it is optional to utilize quadratic interpolation to compute the leading vector $\mathbf{n}_{int}(t)$ for generating the plane holding the cross tangent vector $\mathbf{t}_c$. When choosing $\mathbf{n}_{int}^{Q}(t)$, we define a shape factor α which scales $\mathbf{n}_f$ during the interpolation. Using different values for α, the reconstructed Gregory patch results in different shapes. Figure 13 shows some resultant shapes of using different α for the top patch of the model. When $\alpha \leq 0$, the shape of surface is degenerated.

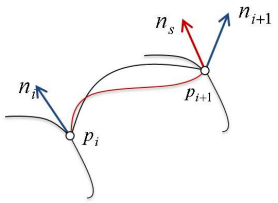

Figure 12: Modifying the silhouette curve that has one vertex on the crease (extended scheme 4).

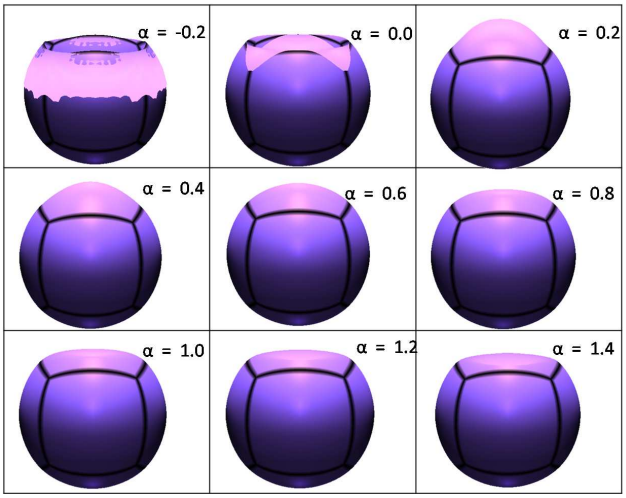

Figure 13: Smooth patch bulging – using different shape factors α for the top patch of the model (extended scheme 5). The last row shows the result by using the linear interpolation $\mathbf{n}_{int}^{L}(t)$.

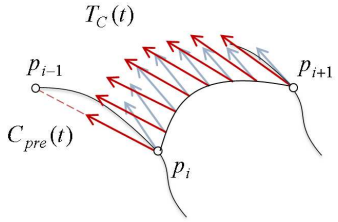

Figure 14: Cross tangent function for flat surfaces (extended scheme 6).

Moreover, during our studies we also find that the influence of this α coefficient on the shape of patch is most significant when applying on a quadrilateral patch.

3.7 Flat surface (extended scheme 6)

In some circumstances, flat surface is preferred. We obtain such a result by replacing $\mathbf{t}_c(t)$ defined in Eq.(10) and (11) with

$$\mathbf{T}_C(t) = (1-t)\mathbf{p}_{i-1} + t\mathbf{p}_{i+2} - \mathbf{C}(t). \tag{18}$$

See Fig.14 for the illustration. For this case, the compatibility condition $\mathbf{T}_C(0) = \dot{\mathbf{C}}_{pre}(1)$ and $\mathbf{T}_C(1) = \dot{\mathbf{C}}_{nex(0)}$ must also be preserved. This is satisfied when $\mathbf{p}_{i-1}$, $\mathbf{p}_i$, $\mathbf{p}_{i+1}$ and $\mathbf{p}_{i+2}$ are coplanar and $\mathbf{n}_{i-1} = \mathbf{n}_i = \mathbf{n}_{i+1} = \mathbf{n}_{i+2}$.

For the same input model M^0, when applying different combinations of the above modification schemes, different shapes can be generated. Figure 15 gives such an example with a simple torus polygonal mesh input.

4 Implementation on the GPU

We have implemented the approach presented above using the OpenGL Shading Language running on the GPU. The surface generation consists of two major steps: refinement and position mapping (see Fig.16 for the illustration).

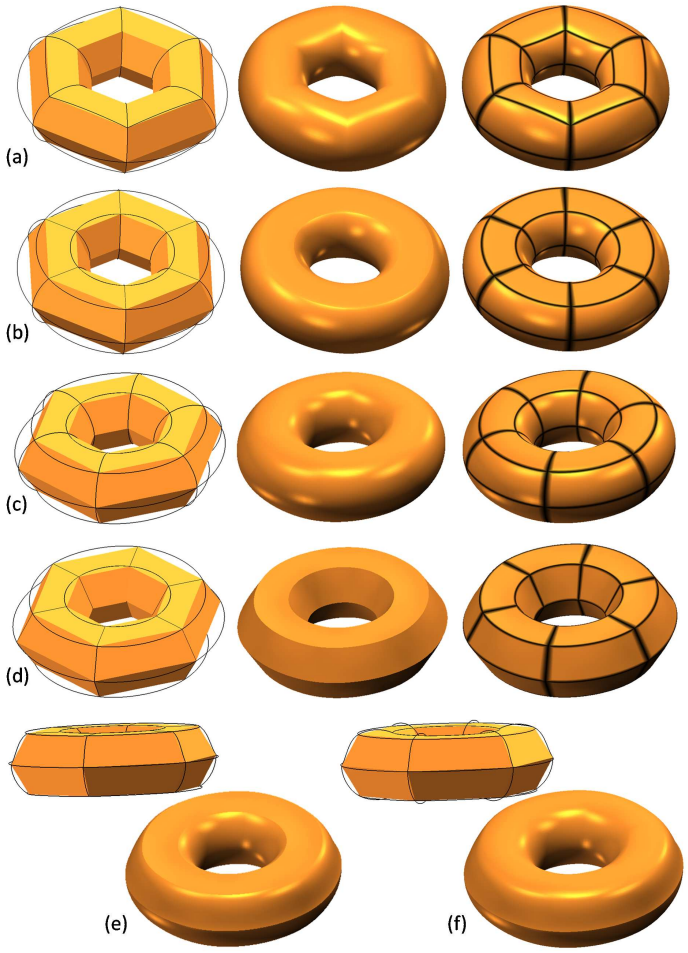

Figure 15: For a single polygonal torus model, a variety of shapes can be generated by our method: (a) the basic scheme, (b) make the top and bottom flat by using extended schemes 1, 2 and 6, (c) use extended scheme 1 to adjust the orientation of silhouette curves, (d) a torus with conical surfaces, (e) the top and bottom are flat and another crease is added in the middle, and (f) with the crease only in the middle.

4.1 Refinement

In the refinement step, an n-sided polygon is refined into nk^2 quadrangles, where each edge on the polygon is subdivided into $2k$ segments and the sub-patch corresponding to a corner interpolation becomes $k \times k$ grids. The refinement procedure is implemented in a geometry shader program exploring the new extension of OpenGL, *transform feedback*, which records elected vertex attributes for each primitive processed. For each patch on a coarse mesh, the n vertices equipped with indices $(0, 0)$ are uploaded to the *Vertex Buffer Object* (VBO). For each vertex, the geometry shader generates four vertices. For a vertex with indices (i, j), the emitted vertices are with indices: 1) (i, j), 2) $(i + 1, j)$, 3) $(i, j + 1)$ and 4) $(i + 1, j + 1)$, which are then written into another VBO and employed as the source of the geometry shader in the next round of feedback. The feedbacks is repeated for k times. However, simply emitting four vertices from a vertex will creates a large number of duplicated vertices. Therefore, for a vertex with (i, j), we emit vertices with indices

- (i, j) only when $i = j = 0$;
- $(i + 1, j)$ only when $j = 0$;
- $(i, j + 1)$ only when $i = 0$;

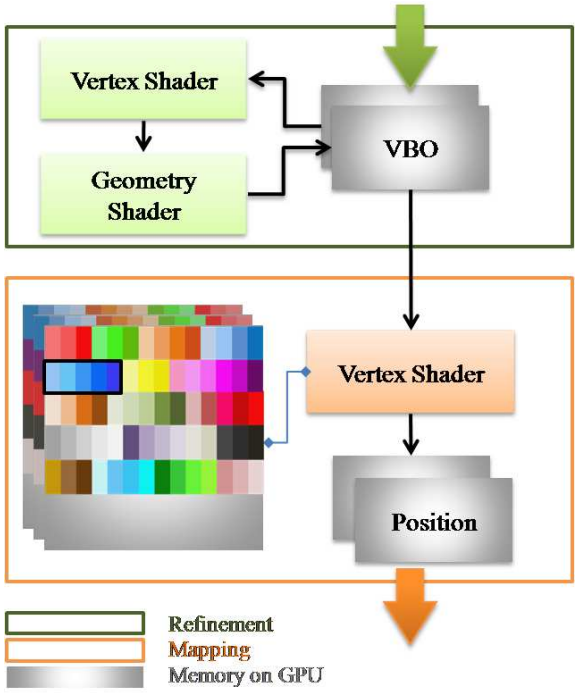

Figure 16: The implementation on the GPU consists of two steps: refinement and position mapping.

- $(i+1, j+1)$ for all i, j.

By these rules, the vertices passed to VBO can be reduced from $n4^{k+1}$ to $n(k+1)^2$.

4.2 Position mapping

For each patch, vertices are exclusively influenced by local information. The positions of refined vertices generated by the geometry shader are computed by fetching textures. Three textures are generated according to the input polygonal model. The first texture contains vertex positions, and the second texture stores vertex normals which align in the same way as the vertex positions on the first texture do. The third texture records the additional data that is required for generating sharp features. For each vertex, two float values are passed to VBO as stated in section 4.1, which actually encoded the following information 1) i and j, 2) where to access the information from the textures, 3) the number of sides of the polygon holding this vertex, and 4) its index in the polygon, etc.

4.3 Normal computation for shading

Normal vectors at newly generated vertices play an important role when visually displaying the reconstructed Gregory patches. Similar to other parametric surfaces, the normal vector of a particular point (ξ_0, η_0) on the surface $G(\xi, \eta)$ can be evaluated by

$$\frac{\partial G(\xi,\eta)}{\partial \xi} \times \frac{\partial G(\xi,\eta)}{\partial \eta}|_{(\xi_0,\eta_0)}.$$

However, the computation involved in this method is even more complex than the evaluation of a surface point. We therefore find a simpler method to compute normal vectors, which are obtained by

- replacing $\mathbf{P}(u)$ and $\mathbf{Q}(u)$ in Eq.(1) with the function $\mathbf{n}_c(t)$ in Eq.(7),
- and using $\mathbf{n}_v$ (the normal vector of the corner vertex v) for $\mathbf{P}(0)$ in Eq.(1).

Table 1: Computational Statistics in Construction Time

Model	Fig.	Polygon Number	Vertex Number	Shape Feature	Time* (ms)	
					$k = 3$	$k = 5$
Frog	1	1,776	7,054	No	125	184
Torus	15	36	144	No	79	80
				Yes	197	202
Monkey	17	500	1,968	No	87	111
Mushroom	18	20	84	No	81	79
				Yes	199	201
Chessman	19	90	368	No	85	87
				Yes	205	212
Armadillo	20	2,384	12,206	No	191	325
Head	21	1,360	5,358	No	115	159
Rabbit	22	2,451	12,474	No	192	331
Vase	23	1,119	5,792	Yes	294	384
Dinosaur	24	2,382	12,146	No	193	327

*The patches are constructed with $k \times k$ quad-grids for each corner.

When sharp features are involved, $\mathbf{P}(u)$ and $\mathbf{Q}(u)$ in Eq.(1) are changed to the normal functions defined by sharp features. Normal vectors evaluated in this way can well capture the shape and shading on the curved polygons.

5 Results and Comparison

The test results of our implementation using GLSL on a PC with Intel Core 2 Quad CPU Q6600 2.4GHz + 4GB RAM and GeForce GTX295 graphics card are very encouraging. On the models with a moderate level of complexity (e.g., around 1,000 polygons), the construction of Gregory patches takes about 100ms only. When sharp features are involved, performance becomes worse but still can achieve interactive response speed. This slowdown is because the work loadings on different patches (with and without sharp features) are not well balanced. Computational statistics in terms of time are listed in Table 1. Besides, we also study the required number of arithmetic operations when evaluating the position of a point on the local reconstructed Gregory patch. Both the best and the worst cases are calculated and shown in Table 2, where the best scenario happens with the basic construction scheme and the worst case is with sharp features on the silhouette curves. Although this is more complex than the prior method (e.g., vertex normal shading and PN-triangle [5]), it still can be performed at an interactive speed on the modern graphics hardware (see Table 1).

We show several examples in this section. Our method can construct curved Gregory patches on surfaces entirely composed of quads like the example in Fig.21, surfaces composed of both quads and triangles (e.g., the examples in Figs.1, 17 and 21), and models composed of n-sided polygons (e.g., those shown in Figs.18-20 and 22-24). The reconstructed Gregory patches interpolate the vertices of the input coarse mesh. By setting different combinations of the shape controlling cases, different shapes can be generated as shown by the examples in Figs.15, 18 and 19.

To compare the quality of our results with the prior methods, we implement the vertex normal based standard shading in OpenGL as well as the PN-triangle based shading [5]. As only triangular mesh models are supported by the PN-triangle approach, we need to triangulate the general polygonal

Table 2: Number of Arithmetic Operations for Point Evaluation

Curved Polygon with n sides				
Operation	$+$	$-$	$\times$	$/$
Best Case	$349n+7$	$239n+4$	$4n^2+758n+22$	$61n+6$
Worst Case	$541n+7$	$386n+4$	$4n^2+1159n+22$	$130n+9$
PN-triangle [5]				
Operation	$+$	$-$	$\times$	$/$
PN-triangle	117	41	219	35

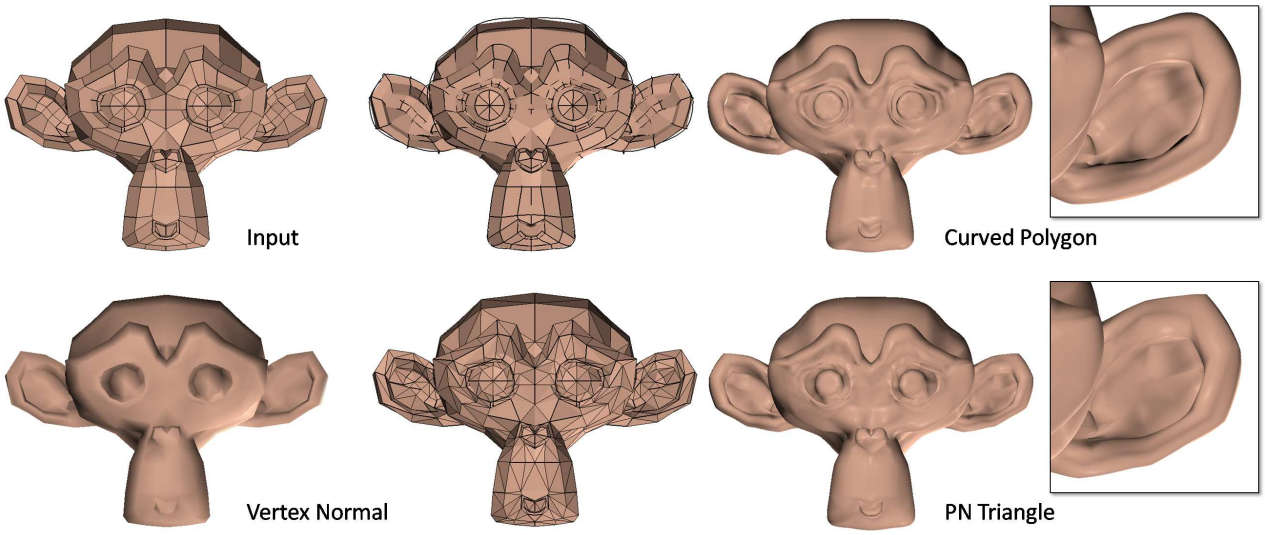

Figure 17: A monkey model with 500 polygons. Top row - from the left: the input coarse mesh, the coarse mesh with silhouette curves, and the model produced by our curved polygon method. Note that the model is composed of both quads and triangles. Bottom row - from the left: the vertex normal based shading by OpenGL, the refined triangular model, and the shading result of PN triangle [5]. From the zoom-views, we can find that the resultant model of our approach outperforms the PN triangle approach.

models into triangular ones before rendering. If this is conducted at the CPU side, the amount of data communication between the main memory and the graphics hardware is more than double of our curved polygon approach. If the triangulation is performed at the GPU side, two geometry shader programs must be developed and switched from one to another where the first one is for triangulation and the second takes care of the refinement according to the PN-triangles. More seriously, the results of PN-triangle approach does not preserve the G^1 continuity across the boundary of patches. As shown in Fig.17, the silhouette of the monkey's ear generated by PN-triangle is not as smooth as ours. In this example, the result of standard vertex normal shading is also shown, which is too coarse. Another comparison of our result to the PN-triangle approach is shown on the Dinosaur model in Fig.24, where unwanted creases and folds are generated by the PN-triangle approach. These are prevented by our method.

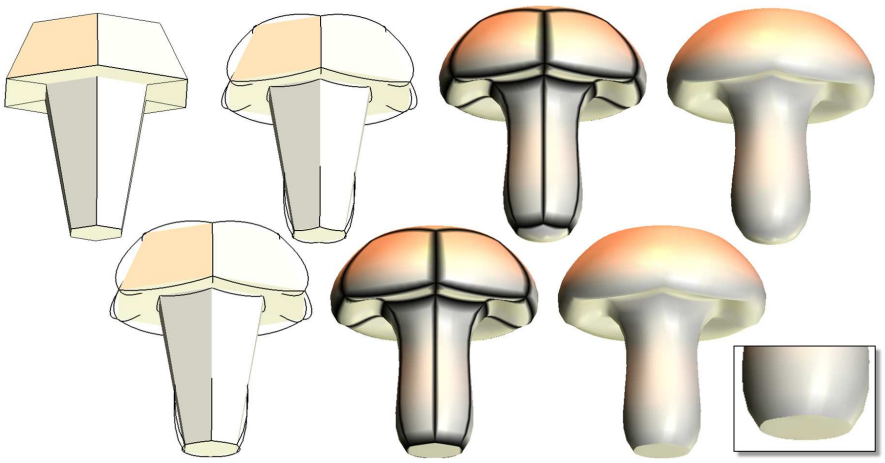

Figure 18: An example of a mushroom model – (top row): the smooth model without sharp features, and (bottom row): with sharp features reconstructed.

6 Conclusions

The approach proposed in this paper aims at providing a localized n-sided polygon refinement scheme using Gregory patch interpolation associated with a customized cross-tangent function. This method provides a smoother silhouette and more organic shape for polygonal models at a minimal cost for model preparation and rendering performance, which is proved by the comparisons with the standard smooth shading based on vertex normals and the results from PN-triangle [5] (see Figs.17 and 24). Its localization attribute enables our algorithm to be supported by the architecture of GPUs. Additionally, we introduce several schemes to control the shape and sharp features on the curved models. The experimental results shown in the paper verify the effectiveness and the efficiency of our approach.

Acknowledgments

The research presented in this paper is partially supported by the Hong Kong Research Grants Council (RGC) General Research Fund (GRF): CUHK/417508 and CUHK/417109.

References

[1] J. Bolz, P. Schröder, Rapid evaluation of Catmull-Clark subdivision surfaces, in: Web3D '02: Proceedings of the seventh international conference on 3D Web technology, ACM, New York, NY, USA, 2002, pp. 11–17.

[2] L.-J. Shiue, I. Jones, J. Peters, A realtime GPU subdivision kernel, ACM Trans. Graph. 24 (3) (2005) 1010–1015.

[3] A. Myles, Y. I. Yeo, J. Peters, GPU conversion of quad meshes to smooth surfaces, in: SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling, ACM, New York, NY, USA, 2008, pp. 321–326.

[4] M. Alexa, T. Boubekeur, Subdivision shading, ACM Trans. Graph. 27 (5) (2008) 1–4.

[5] A. Vlachos, J. Peters, C. Boyd, J. L. Mitchell, Curved PN triangles, in: I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics, ACM, New York, NY, USA, 2001, pp. 159–166.

[6] T. Boubekeur, M. Alexa, Phong tessellation, ACM Trans. Graph. 27 (5) (2008) 1–5.

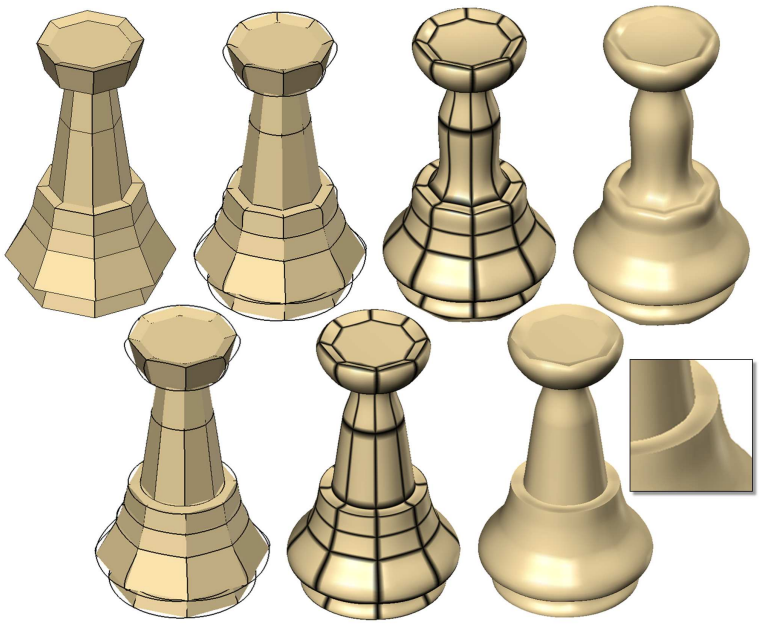

Figure 19: An example of a chessman model – (top row): the smooth model without sharp features, and (bottom row): with sharp features reconstructed.

[7] P. Volino, N. Magnenat-Thalmann, The SPHERIGON: A simple polygon patch for smoothing quickly your polygonal meshes, in: CA '98: Proceedings of the Computer Animation, IEEE Computer Society, Washington, DC, USA, 1998, p. 72.

[8] J. Gregory, n-sided surface patches, in: Mathematics of Surfaces, Clarendon Press, Oxford, UK, 1986, pp. 217–232.

[9] L. Huang, J. Zhen, X. Zhu, L. Yi, A surface interpolating method for 3d curves-nets, in: Technical Report: HZ-TMSurf-Huang02, Beihang University, 1996.

[10] T. Hermann, J. Peters, T. Strotman, A geometric criterion for smooth interpolation of curve networks, in: SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling, ACM, New York, NY, USA, 2009, pp. 169–173.

[11] J. Stam, Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values, in: SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, 1998, pp. 395–404.

[12] M. Halstead, M. Kass, T. DeRose, Efficient, fair interpolation using catmull-clark surfaces, in: SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, 1993, pp. 35–44.

[13] C. Loop, S. Schaefer, T. Ni, I. Casta no, Approximating subdivision surfaces with Gregory patches for hardware tessellation, in: SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers, ACM, New York, NY, USA, 2009, pp. 1–9.

[14] C. Fünfzig, K. Müller, D. Hansford, G. Farin, PNG1 triangles for tangent plane continuous surfaces on the GPU, in: GI '08: Proceedings of graphics interface 2008, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 2008, pp. 219–226.

[15] J. Gregory, P. Yuen, An arbitrary mesh network scheme using rational splines, in: Mathematical Methods in Computer Aided Geometric Design II - Lyche T. and Schumaker L.L. (eds.), Academic Press, Inc., 1992, pp. 321–329.

[16] R. Hall, G. Mullineux, Shape modification of Gregory patches, in: The Mathematics of Surfaces VII - Goodman T. and Martin R. (eds.), Information Geometers, 1997, pp. 393–408.

Figure 20: An example of the Armadillo model with 2,384 polygons and 12,206 vertices.

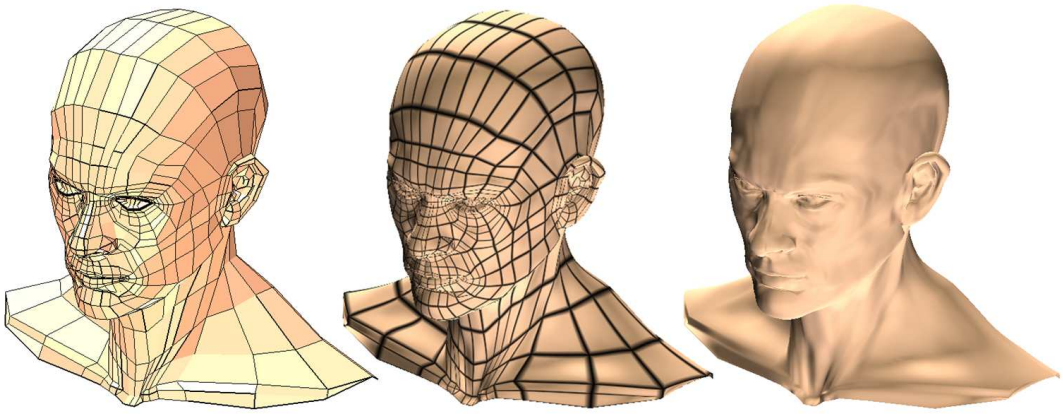

Figure 21: An example of a head model with 1,360 polygons and 5,358 vertices.

[17] C. Dyken, M. Reimers, J. Seland, Real-time GPU silhouette refinement using adaptively blended bézier patches, Comput. Graph. Forum 27 (1) (2008) 1–12.

[18] T. Boubekeur, C. Schlick, A flexible kernel for adaptive mesh refinement on GPU, Comput. Graph. Forum 27 (1) (2008) 102–113.

[19] M. Schwarz, M. Stamminger, Fast GPU-based adaptive tessellation with CUDA, Comput. Graph. Forum 28 (2) (2009) 365–374.

[20] C. Dyken, M. Reimers, J. Seland, Semi-uniform adaptive patch tessellation, Comput. Graph. Forum 28 (8) (2009) 2255–2263.

[21] L. A. Piegl, W. Tiller, Filling n-sided regions with NURBS patches, The Visual Computer 15 (2) (1999) 77–89.

[22] X. Wang, F. Cheng, B. A. Barsky, Blending, smoothing and interpolation of irregular meshes using n-sided Varady patches, in: Symposium on Solid Modeling and Applications, 1999, pp. 212–222.

[23] D. J. T. Storry, A. A. Ball, Design of an n-sided surface patch from Hermite boundary data, Comput. Aided Geom. Des. 6 (2) (1989) 111–120.

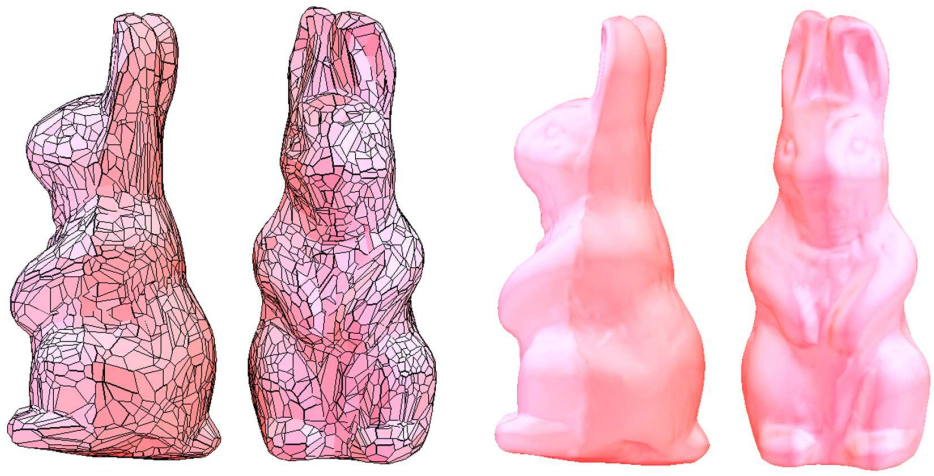

Figure 22: The rabbit model with 2,451 polygons and 12,474 vertices.

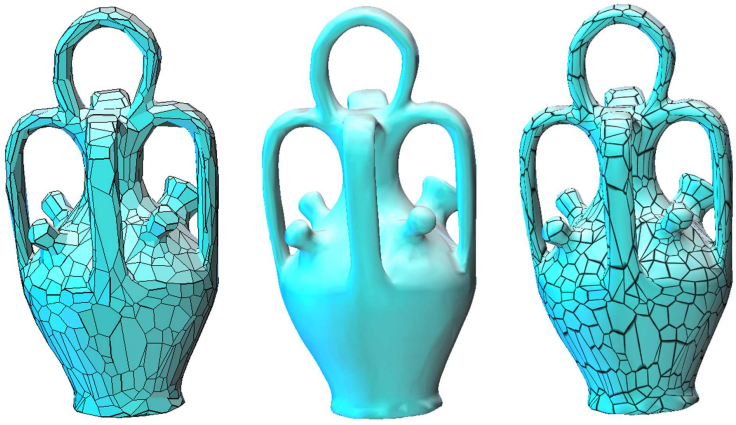

Figure 23: A vase model with 1,119 polygons and 5,792 vertices.

Figure 24: A dinosaur model with 2,382 polygons and 12,146 vertices, where in the zoom-views our results (left) are compared with the PN triangles (right). The unwanted creases shown on the PN triangle model do not appear on our result.

[24] N. Pla-Garcia, M. Vigo-Anglada, J. Cotrina-Navau, N-sided patches with B-spline boundaries, Comput. & Graph. 30 (6) (2006) 959–970.

[25] J. Wu, Y.-S. Leung, C. Wang, D. Wang, Y. Zhang, Smooth force rendering on coarse polygonal meshes, Computer Animation and Virtual Worlds 21 (3–4) (2010) 235–244.

[26] G. Farin, Curves and Surfaces for CAGD: A Practical Guide, Morgan Kaufmann Publishers, 2002.

[27] M. Mortenson, Geometric Modeling, Chapter 3.4, Wiley Computer Publishing, 1997.

[28] H. Chiyokura, F. Kimura, Design of solids with free-form surfaces, SIGGRAPH Comput. Graph. 17 (3) (1983) 289–298.

[29] D. Cohen-Steiner, P. Alliez, M. Desbrun, Variational shape approximation, in: SIGGRAPH '04: ACM SIGGRAPH 2004 Papers, ACM, New York, NY, USA, 2004, pp. 905–914.