Multimedia Tools and Applications manuscript No.
(will be inserted by the editor)

Yuncen Huang · Jieqing Feng

Efficient Skeleton-guided Displaced Subdivision Surfaces

Received: 20 August 2016/ Revised: 4 January 2017 / Accepted: 23 January 2017

Abstract Displacement mapping is a computer graphics technique that uses scalar offsets along normals on a base surface to represent and render a model with highly geometric details. The technique natively compresses the model and saves memory I/O. A subdivision surface is the ideal base surface, due to its good geometric properties, such as arbitrary topology, global smoothness, and multi-resolution via hardware tessellation, among others. Two of the main challenges in displacement mapping representation are constructing the base surface faithfully and generating displacement maps efficiently. In this paper, we propose an efficient skeleton-guided displaced subdivision surfaces method. The construction of the base mesh is guided by a sketched skeleton. To make the shape of the base surface fit the input model well, we develop an efficient progressive GPU-based subdivision fitting method. Finally, a GPU-based raycasting method is proposed to sample the input model and generate the displacement maps. The experimental results demonstrate that the proposed method can efficiently generate a high-quality displacement mapping representation. Compared with the traditional displaced subdivision surface method, the proposed method is more suitable for the modern rendering pipeline and has higher efficiency.

Keywords displacement mapping · subdivision surfaces · GPU · skeleton · raycasting

1 Introduction

A mesh is a prevalent representation of geometry in computer graphics due to its simplicity, flexibility and versatility. Triangular and quadrilateral meshes are the most commonly used meshes. As the requirements for large-scale meshes with abundant geometric details, such as laser scan range data, have rapidly increased, representing and rendering such a complex mesh model directly is not an efficient approach because since it will degrade the computing and rendering performances, and significantly consume memory and I/O bandwidth.

Alternatives have been proposed to express the geometric details compactly. Among such alternatives, storing various details in a 2D texture form is feasible and flexible approach, which adapts well to the rendering pipeline. These representations are widely used in 3D computer games and in the film industry. For example, bump mapping [3] consists of to simulating bumps and wrinkles on the surface by perturbing the surface normals of the object. Normal mapping [8] can be regarded as 3-channel bump mapping, which stores the surface normals as regular RGB images. Parallax mapping [20] further improves the visual quality of bump mapping and normal mapping by further displacing the texture coordinates. In these methods, a complex model with abundant geometric details is expressed as a coarse base surface together with 1-channel or 3-channel textures. Although

Jieqing Feng
State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou, 310058, China
E-mail: jqfeng@cad.zju.edu.cn

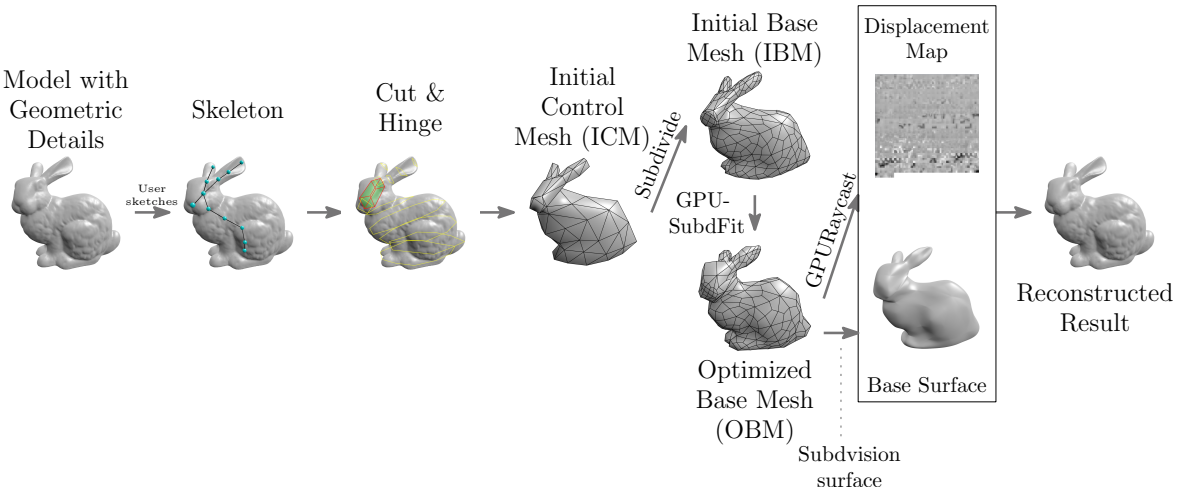


Fig. 1 The framework of our method.

these techniques can create visually plausible shading effects, artifacts can still be easily detected, particularly along the silhouettes since there is no actual geometric change on the surface.

Displacement mapping [9] treats the geometric details as genuine offsets along the surface normals of a base surface. Consequently, the silhouette artifacts can be alleviated and the geometric details can still be natively compressed. Because of its fidelity, the displacement mapping technique has become popular, particularly in the film industry, 3D games, and so forth. In general, the displacement maps are obtained manually using sculpting tools, such as MudBox[2], ZBrush[21], and so forth. It is a forward design procedure. However, it is more attractive and effective to automatically extract the displacement maps from a given complex model. In such a case, a base mesh needs to be constructed first. However, constructing a well-shaped base mesh remains a challenge.

Following the releases of OpenGL v4.0 and Direct3D v11, the GPU rendering pipeline supports a tessellation stage, whose inputs are *patches* and outputs are tessellated primitives. A subdivision surface is suitable to be processed via hardware tessellation, and many practical algorithms on GPUs have been developed [29]. In this paper, an efficient skeleton-guided displaced subdivision surfaces method is presented, which converts a complex mesh model into a displacement mapping representation. The construction of the initial base mesh is constructed guided by a skeleton rather than mesh simplification [24]. In this way, the structure of the model can be well captured and the user's design intents can easily be expressed. Furthermore, the computational complexity is greatly decreased.

Since the initial base mesh may not fit the model well, artifacts may arise when sampling the geometric details of the complex object. To this end, a progressive GPU-based subdivision fitting method is developed. Then, a GPU-based raycasting algorithm is proposed to sample the geometric details and generate the displacement maps, and the algorithm is fully implemented in the rendering pipeline. The main contributions of the proposed method can be summarized as follows:

- an efficient skeleton-guided base mesh generation method;
- a progressive GPU-based subdivision fitting algorithm to optimize the base surface;
- a GPU-based raycasting method for sampling the geometric details to generate displacement maps.

The remainder of this paper is organized as follows. Section 2 reviews the related work. The proposed efficient skeleton-guided displaced subdivision surfaces are described in detail in Section 3. We present the implementation results and compare the proposed method with other methods in Section 4, and we draw conclusions in Section 5.

2 Related Work

Displacement mapping represents a model as a base surface and additional geometric offsets. It can be expressed as follows:

$$\mathbf{r}(u,v) = \mathbf{s}(u,v) + \mathbf{H}(u,v)$$

where $\mathbf{r}(u,v)$ is the underlying model with high-frequency geometric details, $\mathbf{s}(u,v)$ is the base surface, and $\mathbf{H}(u,v)$ is the displacement function [32]. In scalar displacement mapping, the offsets are displaced along the normals of the base surface; thus, $\mathbf{H}(u,v)$ can be re-written as

$$\mathbf{H}(u,v) = \mathbf{n}(u,v) \times h(u,v)$$

where $\mathbf{n}(u,v)$ is the unit normal of the base surface, and $h(u,v)$ is a scalar height function, which is often stored in 2D 1-channel textures.

Wang et al.[33] proposed a technique called view-dependent displacement mapping for high-quality rendering of detailed appearances, such as fine-scale self-shadows, occlusions and silhouettes along the viewing direction. Jang and Han [18] sampled more vertices on the higher-frequency feature parts of an input model with one extra 3D map for each patch, called the feature-preserving displacement map, to re-map sample points in the domain shaders. Jang and Han [19] proposed the indirect scalar displacement maps (ISDMs), aiming for maintaining the features while animating. Hirche et al.[15] proposed a local ray tracing method to efficiently render displaced surfaces, where the algorithm for the ray-surface intersection is completely designed using pixel shaders. Wang et al.[34] introduced a five-dimensional displacement map, called the *generalized displacement map* (GDM). The purpose of the map was to reduce texture distortions and to generate rich shadow effects. Nießner and Loop [28] introduced a smooth analytic displacement function and stored the coefficients in a tile-based texture format. In this way, the normals of displaced surface can be evaluated analytically rather than querying from a pre-computed normal map.

In the aforementioned methods, the generation of the base mesh is not well studied in the displacement mapping representation. Many works [23,22,14] adopt a parametric surface as the base surface because the normals of the parametric surface can be evaluated analytically and in parallel on GPU. However, constructing a globally smooth 2-manifold shape by smoothly joining the piecewise parametric surfaces together remains a challenge. On the other hand, subdivision surfaces, which were first introduced by Catmull and Clark [6] and by Doo and Sabin [12], are inherently globally smooth. A subdivision surface has many good properties, such as arbitrary topology, scalability, multiresolution, global smoothness, analytical computation [31], and so forth. Such surfaces are currently widely used in the film industry, 3D games, and so forth to address many complex shape modeling challenges [11]. Recently, many works on subdivision surface rendering on GPUs have been developed[29,5]. Among them, Loop and Schaefer approximated Catmull-Clark subdivision surfaces using bicubic Bézier patches [27]. Their approximate Catmull-Clark subdivision surface is C^1 everywhere, eliminating normal discontinuity between patches.

Lee et al.[24] introduced subdivision surfaces into displacement mapping representation, called *displaced subdivision surfaces* (DSS). They used edge-collapse simplification to obtain the base mesh from the input model in a recursive manner. It was extended to address animated meshes [25] by considering the errors caused by motion. One drawback of DSS is that they need to delete the edges one by one. For a large-scale model, the process can require hours. In the proposed method, the construction of the initial base mesh is guided by a skeleton, reducing the computational complexity. Furthermore, through hardware tessellation, subdivision fitting and the sampling procedures become more efficient.

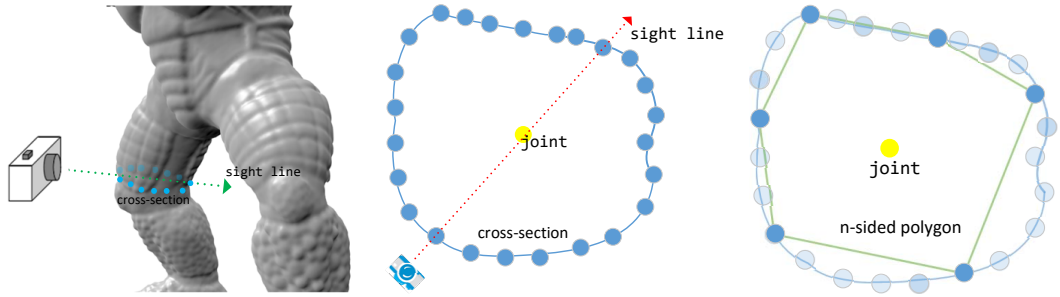
3 Efficient Skeleton-guided Displaced Subdivision Surfaces

The framework of the proposed method is shown in Fig. 1. The input of the proposed method is a mesh model with abundant geometric details. First, a user sketches a *skeleton* on the input. Then, the *initial control mesh* (ICM) is generated by properly connecting *cuts* and *hinges* along the skeleton. Next, the ICM is subdivided into the *initial base mesh* (IBM). A *progressive GPU-based subdivision fitting method (GPUSubdFit)* is developed to optimize the IBM to further approximate the input model, and the *optimized base mesh* (OBM) is obtained. Finally, the Catmull-Clark subdivision

surface of the OBM serves as the *base surface* and a *GPU-based raycasting (GPURaycast) method* is developed to sample the input model for displacement map generation.

3.1 Generation of Skeleton and Base Mesh

3.1.1 Skeleton Generation



(a) a sight line from the camera (b) a cross-section (c) a cut (n-sided polygon)

Fig. 2 Illustration of a joint of skeleton and a cut generation.

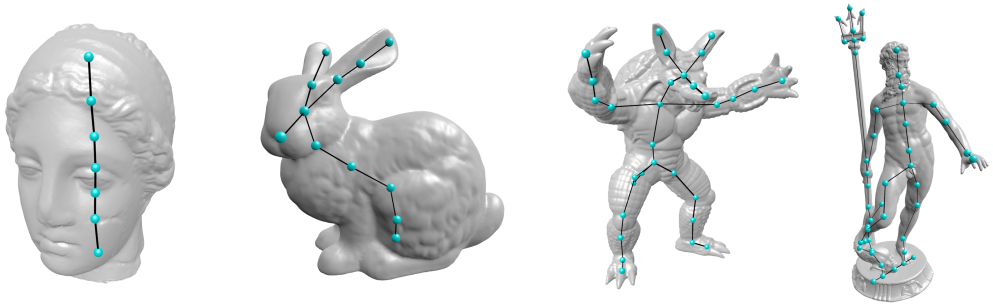


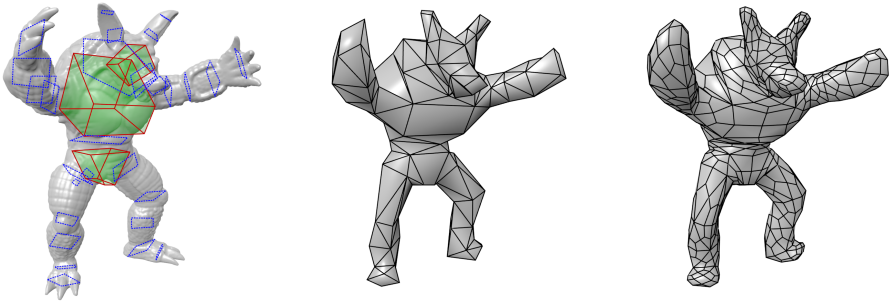
Fig. 3 The skeletons of Venu, bunny, armadillo and Neptune are generated through sketching.

The skeleton represents the global structure of a model. Thus, it is a dominant control structure that is generally used in animation. The skeleton can be extracted automatically or interactively [10]. The automatic extraction methods are convenient for users. However, capturing the structure and shape of the model or expressing user's design intents is challenging. For example, a joint, which is not equipped with prominent geometric features and thus, it is difficult to be extracted automatically. In this paper, we develop an intuitive sketching interface to define the skeleton of an input model. The user picks a point on the screen near an articulation. The viewpoint and the picked point form a ray in world space, as shown in Fig. 2(a). The middle point of the nearest and the farthest intersections between the ray and the model is taken as the default position of the joint (Fig. 2(b)). The desired position of the joint can be further adjusted manually if necessary. After the joints are specified as above, they are connected properly as the skeleton of the model. In fact, it will not influence the initial mesh generation if the position of the joint is not accurate. Thus, the proposed skeleton generation method is simple and intuitive. The sketched skeletons of Venus, bunny, armadillo and Neptune are shown in Fig. 3.

3.1.2 Generation of Initial Control Mesh and Initial Base Mesh

Based on the sketched skeleton, the ICM and the IBM will be constructed using the following steps.

Cut generation. A cross-section, as shown in Fig. 2(b), is generated at each articulation joint, whose valence is not greater than 2. Then, the cross-section is approximated as an n-sided polygon (Fig. 2(c)), called a *cut*, which is an element of the ICM. We adopt the method of [13] to generate the cut. The point list of the polygon is initialized to be the farthest point pair among the input points, and then the farthest point among the rest of input points is iteratively added into the current point list until the n is meet. In our implementation, the "n" of a cut is determined according to the ratio between the perimeter of the cross-section and the bounding box of the input model. In contrast to the ICM generation method [35], in which all cuts are quadrilaterals, the cuts in the proposed method allow n-sided polygons because an n-sided polygon can better approximate the shadpe of the model. The resulting cuts of the armadillo are shown as blue dashed lines in Fig. 4(a).



(a) cuts & Hinges (b) ICM (c) IBM

Fig. 4 The generated cuts (blue dashed polygons), hinges (red-line polyhedrons), ICM and IBM of the armadillo.

Hinge generation. A hinge is a polyhedron, in which each face intersects with at most one skeleton segment. We adopt the method of [35] to construct a *hinge* on each joint whose valence is larger than 2. The hinges in the armadillo are shown as red-line polyhedrons in Fig. 4(a).

Connection of cuts and hinges. For each skeleton segment, we connect two adjacent cuts, or a cut and a face of a hinge sharing the segment, to form a *pipe*. For two adjacent cuts or a cut and a face of a hinge, we map one of them onto the plane which the other one is on and connect the nearest pair of points to form a pipe. Since the hinge is generated in such that each face of the hinge has at most one skeleton segment intersecting with the face, the pipes and hinges can form a well-shaped skeleton-defined ICM. An example is presented in Fig. 4(b).

From ICM to IBM. Since quadrilaterals have relatively less distortions in piecewise texture misalignment than triangles, we use a tile-based texture format to record displacement maps, similar to Ptex[4], where each tile corresponds to a quad face of the base mesh. To this end, we perform Catmull-Clark subdivision [6] one time on the ICM to obtain the all-quadrilateral IBM. To reduce the shrinking problem caused by Catmull-Clark subdivision, we perform *progressive interpolation of Catmull-Clark subdivision* [7] one or two times prior to the subdivision. An example is presented in Fig. 4(c). Note that the proposed framework is compatible with the base surface generated through simplification [24].

3.2 Progressive GPU-based Subdivision Fitting

Although the IBM can be used as the control mesh to generate the Catmull-Clark subdivision surface, it may not sufficiently approximate the input model, which may lead to an unfaithful displacement map. To this end, we present a progressive *GPU-based subdivision fitting* method (GPUSubdFit) inspired by the method in [26], where the IBM is optimized as the OBM. The main idea is to

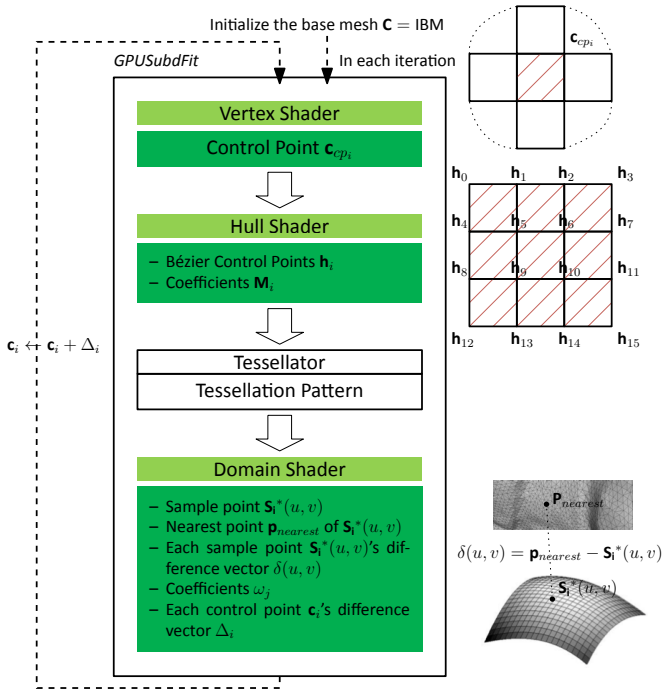
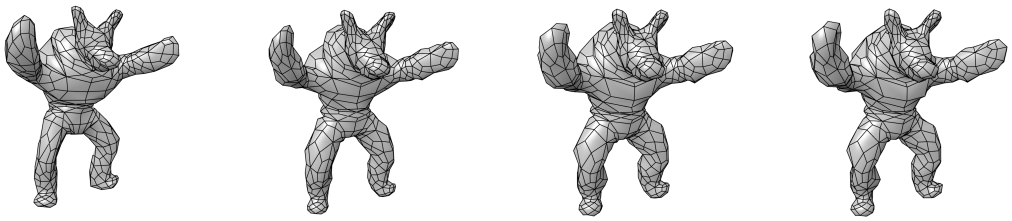


Fig. 5 Overview of the rendering pipeline of the progressive GPU-based subdivision fitting method, where programmable stages are in green and the fixed stages are in white.



(a) IBM (b) 1st iteration, $\epsilon = 24.731\%$ (c) 2nd iteration, $\epsilon = 18.045\%$ (d) 3rd iteration, $\epsilon = 8.236\%$

Fig. 6 The armadillo IBM and the result of each iteration in GPUSubdFit (tessfactor=16×16, ϵ is the RMS error, and the OBM is the resulting mesh of the 3rd iteration).

progressively adjust the control points of the IBM by evaluating the error between the subdivision surface of the IBM and the input model, until the fitting error is less than a pre-defined threshold.

The input detailed model is denoted as $\mathbf{P}$, whose vertices are denoted as $\{\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_{n_p-1}\}$. The base mesh, denoted as $\mathbf{C}$, is also referred to as the *control mesh*. The vertices of $\mathbf{C}$ are denoted as $\{\mathbf{c}_0, \mathbf{c}_1, \cdots, \mathbf{c}_{n_s-1}\}$. GPUSubdFit is an iterative method. At the beginning of GPUSubdFit, the base mesh $\mathbf{C}$ is initialized as the IBM. The Catmull-Clark subdivision surfaces of the base mesh $\mathbf{C}$ is denoted as $\mathbf{S}$. In each iteration of GPUSubdfit, we evaluate the *sample points* on $\mathbf{S}$, as well as their corresponding nearest points on $\mathbf{P}$, denoted as $\mathbf{p}_{nearest}$. The sample points are on a surface, which will be evaluated in the rendering pipeline. Then, each control point in the control mesh is progressively updated according to the related difference vectors between the sample points and their corresponding nearest points.

To facilitate the nearest point search, we adopt a space-partitioning data structure designed on a GPU, 3DDDA [1]. Because the normals of the base surface are important in the definition of displacement mapping, we take not only geometric positions but also normal similarity into consideration when searching the nearest points. Therefore, we map each vertex from the three-dimensional Eu-

clidean space into the six-dimensional feature-sensitive space [30] of the base surface when evaluating the nearest point.

To exploit the parallelism of GPU, we approximate the Catmull-Clark subdivision surface $\mathbf{S}$ with bicubic Bézier patches $\{\mathbf{S_i^*}, i = 0, ..., n_F - 1\}$ [27], where n_F is the number of faces of the base mesh. Each bicubic Bézier patch $\mathbf{S_i^*}$ corresponds to a quad face/patch $\mathbf{C}_i$ on the base mesh $\mathbf{C}$. The *bicubic Bézier control points (BCPs)* of $\mathbf{S_i^*}$ are evaluated as the weighted summation between the *control points(CPs)* of the 1-ring neighbor patches of $\mathbf{C}_i$ and the masks that are encoded by a set of coefficients [27]. A sample point on the bicubic Bézier patch $\mathbf{S_i^*}$ is denoted as $\mathbf{S_i}^*(u, v), 0 \leq u \leq 1, 0 \leq v \leq 1$. The sample density of (u, v) is defined by *tessfactors* in the rendering pipeline. The pipeline of GPUSubdFit is illustrated in Fig. 5. Each iteration of GPUSubdFit consists of the following four steps:

1. Evaluate the sample point $\mathbf{S_i}^*(u, v)$ on the bicubic Bézier patch, which approximates a patch of the Catmull-Clark subdivision surface, $\mathbf{S_i}^*(u, v) = \sum_{j=0}^{15} b_j(u, v)\mathbf{h}_j$, where $b_j(u, v)$ is the bicubic Bézier basis function, and $\mathbf{h}_j$ is the *bicubic Bézier control point(BCP)*. $\mathbf{h}_j$ is evaluated as follows,

$$\mathbf{h}_j = \sum_{k=0}^{N-1} \omega_{j,k}\mathbf{c}_{cp_k} = \mathbf{M}_j \begin{bmatrix} \mathbf{c}_{cp_0} \\ \mathbf{c}_{cp_1} \\ . \\ . \\ . \\ \mathbf{c}_{cp_{N-1}} \end{bmatrix}$$

   where $\mathbf{c}_{cp_k}, cp_k \in [0, n_s)$, is the *control point(CP)* and $\mathbf{M}_j$ is its masks [27]. N is related to the valences of the four corners vertices of the underlying patch on the base mesh. In our implementation, N is set as 32.
2. Find the closest vertex $\mathbf{p}_{nearest}$ on the input model $\mathbf{P}$ of each sample point $\mathbf{S_i}^*(u, v)$ and calculate its difference vector $\delta(u, v) = \mathbf{p}_{nearest} - \mathbf{S_i}^*(u, v)$.
3. Evaluate the coefficients w_k between each sample point $\mathbf{S_i}^*(u, v)$ and its control points $\mathbf{c}_{cp_j}, j = 0, ..., N - 1$,

$$\mathbf{S_i}^*(u, v) = \begin{bmatrix} b_0(u, v) \\ b_1(u, v) \\ . \\ . \\ . \\ b_{15}(u, v) \end{bmatrix}^T \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ . \\ . \\ . \\ \mathbf{h}_{15} \end{bmatrix} = \begin{bmatrix} b_0(u, v) \\ b_1(u, v) \\ . \\ . \\ . \\ b_{15}(u, v) \end{bmatrix}^T \begin{bmatrix} \mathbf{M}_0 \\ \mathbf{M}_1 \\ . \\ . \\ . \\ \mathbf{M}_{15} \end{bmatrix} \begin{bmatrix} \mathbf{c}_{cp_0} \\ \mathbf{c}_{cp_1} \\ . \\ . \\ . \\ \mathbf{c}_{cp_{N-1}} \end{bmatrix} = \begin{bmatrix} w_0 & w_1 & ... & w_{N-1} \end{bmatrix} \begin{bmatrix} \mathbf{c}_{cp_0} \\ \mathbf{c}_{cp_1} \\ . \\ . \\ . \\ \mathbf{c}_{cp_{N-1}} \end{bmatrix},$$

   then we have

$$w_k = \sum_{j=0}^{15} b_j(u, v)M_{jk}, k = 0, ..., N - 1,$$

   where M_{jk} is the jth element of $\mathbf{M_k}$.
4. Accumulate the difference vector Δ_i of each control point $\mathbf{c}_i$ from all its related sample points's difference vector δ_j according to the respective coefficients w_j,

$$\Delta_i = \frac{\sum_j w_j\delta_j}{\sum_j w_j}$$

   and update $\mathbf{c}_i$ with Δ_i, $\mathbf{c}_i \leftarrow \mathbf{c}_i + \Delta_i$.

In our implementation, we perform Laplacian smoothing in the first iteration to alleviate the potential self-intersections.

The above steps are performed iteratively until

$$\epsilon = \sqrt{\frac{\sum_{i=0}^{n_s-1} \delta_i^2}{n_s}}/B < \epsilon_0$$

where ϵ_0 is the threshold and ϵ is the *root-mean-square error*(RMS) between the fitted subdivision surface and the input model, normalized to the diagonal length of the bounding box, B. In our experiments, no more than five iterations are required and the threshold is set to 0.1. The base meshes and errors of armadillo after each GPUSubdFit iteration are shown in Fig. 6. The convergence of this algorithm is discussed in [26]. To summarize, GPUSubdFit is suitable for parallel implementation with hard tessellation.

3.3 Displacement Map Generation via Raycasting Sample on GPU

(a) Base surface(782 patches) (b) Casted sample rays

Fig. 7 The base surface and sample rays (the sample density on each patch is 16×16) of the armadillo.

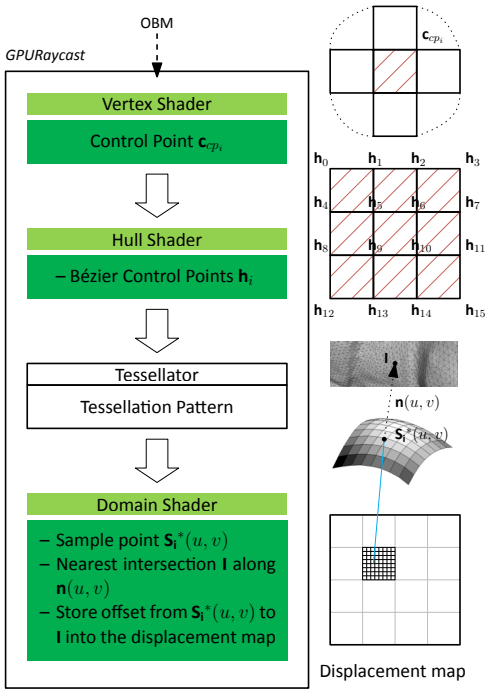
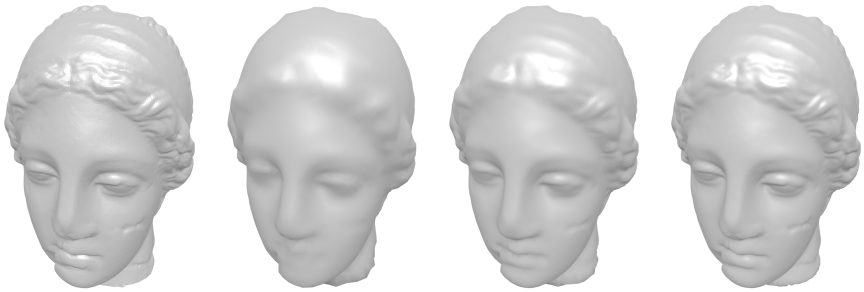
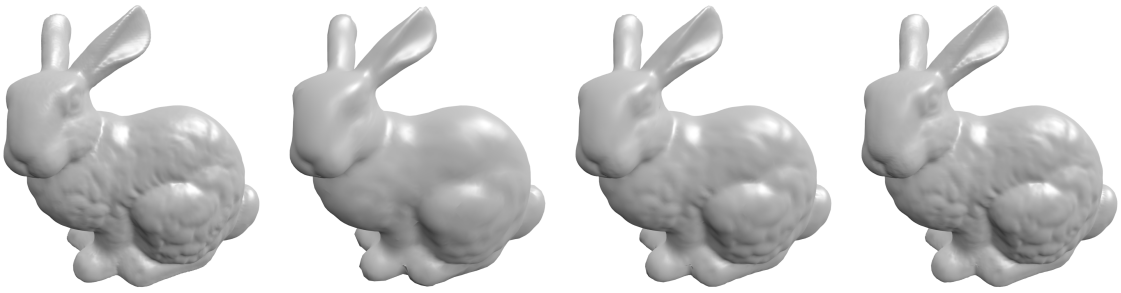


Fig. 8 Overview of the rendering pipeline of GPURaycast, where the programmable stages are in green and the fixed stages are in white.



(a) Input (b) sd=4 × 4, $E = 0.129\%$ (c) sd=8 × 8, $E = 0.052\%$ (d) sd=16 × 16, $E = 0.024\%$

Fig. 9 Results of Venus.



(a) Input (b) sd=4 × 4, $E = 0.190\%$ (c) sd=8 × 8, $E = 0.075\%$ (d) sd=16 × 16, $E = 0.037\%$

Fig. 10 Results of the bunny.

After we obtain the OBM, the differences between the base surface and the input model, which are geometric details, can be represented as the displacement mapping. Since each sample in the displacement maps via raycasting is independent, the procedure can be efficiently implemented on GPU. To this end, we introduce a GPU-based raycasting method (GPURaycast) that takes advantage of hardware tessellation to compute each sample of displacement maps. To avoid cracks between patches, the normals across the boundary of the patches are required to be continuous. Therefore, we approximate the base surface using piecewise smooth bicubic Bézier patches [27], which ensures globally C^1 continuous. The base surface of the armadillo is illustrated in Fig. 7(a).

To acquire the displacement map, a ray is casted from each sample point $\mathbf{S_i}^*(u, v)$ along its normal and the nearest intersection with the input model can be found in the domain shader. The density of the sample rays on each patch is defined by the *sample density*(sd). The offsets are stored in a 2D 1-channel texture as the displacement map. We adopt the same data structure [1] mentioned in Section 3.2 to accelerate the nearest intersection search. The overview of each stage of GPURaycast in the rendering pipeline is shown in Fig. 8, and the casted sample rays on the armadillo are shown in Fig. 7(b).

4 Results and Comparison

We implemented the proposed method using Direct3D v11.1 on a PC with a 2.80 GHz Intel Core i5-2300 CPU, an NVIDIA GeForce GTX 570 GPU, and Windows 10. All GPU codes are written in HLSL (high-slevel shading language).

Four complex models of different scales, from 50k to 2M vertices, are selected as test examples, i.e., Venus, bunny, armadillo and Neptune. The rendering results for these examples are shown in Figs. 9, 10, 11, and 12, respectively. We use E, which is the RMS error between the reconstructed surface and the input model normalized to the diagonal length of the bounding box, to measure the constructed quality. Tables 1 and 2 present the statistics of the models and performances.

(a) Input (b) sd=4×4, $E = 0.172\%$ (c) sd=8×8, $E = 0.078\%$ (d) sd=16×16, $E = 0.046\%$

Fig. 11 Results of the armadillo.

(a) Input (b) sd=4×4, $E = 0.195\%$ (c) sd=8×8, $E = 0.123\%$ (d) sd=16×16, $E = 0.101\%$

Fig. 12 Results of Neptune.

Table 1 The numbers of vertices and faces of the input models, the numbers of vertices and faces of ICM and IBM, and the number of joints in our experiments of Venus, bunny, armadillo, and Neptune.

Models	venus	bunny	armadillo	neptune
Input Model Vertices Number	50002	124037	172974	2003932
Input Model Faces Number	100000	248070	345944	4007872
ICM Vertices Number	60	80	137	225
ICM Faces Number	120	144	256	420
IBM/OBM Vertices Number	368	44a6	784	1292
IBM/OBM Faces Number	370	444	782	1294
Joints Number	7	12	31	47

4.1 Analysis of Results

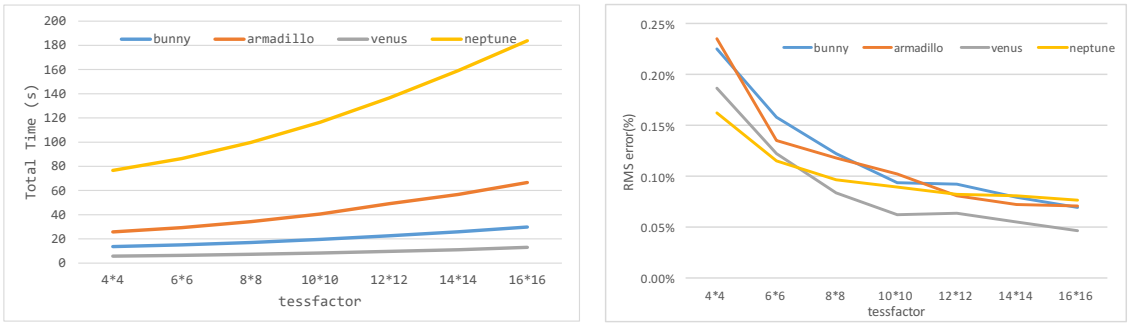
In the experiments, the proposed method can reconstruct the input complex model well while preserving the geometric details faithfully. Fig. 11(d) shows that geometric details on the armadillo's legs and its head are recovered faithfully. The geometric details of the hair of Venus, the body of the bunny, and the beard of Neptune are well preserved in the reconstructed meshes, as shown in Fig. Fig. 9(d), 10(d) and Fig. 12(d), respectively. Therefore, the proposed method can process not only the models of simple structures (such as Venus, whose skeleton is a straight line from top to bottom) but also the models with a complex topology (such as Neptune, which has 2 genus, 9 hinges and 38 cuts). Thanks to the hardware tessellation, GPUSubdFit and GPURaycast are highly efficient,

Table 2 Statistics of our experiments on different models, including execution time, compression ratio and RMS in different tessfactors and sample density. T. denotes the time measured in seconds(s).

Statistics	tessfactor = sample density	venus	bunny	armadillo	neptune
GPUSubdFit Iterations		2	4	3	5
T. IBM Generation (s)		0.009	0.013	0.069	0.104
RMS error (%)	4×4	0.187	0.225	0.236	0.162
Compression Ratio		42.175	86.738	68.670	481.124
T. GPUSubdFit (s)		4.790	11.861	21.828	66.573
T. GPURaycast (s)		0.9	1.785	3.892	9.939
T. Total (s)		5.765	13.659	25.789	76.616
RMS error (%)	8×8	0.084	0.123	0.118	0.097
Compression Ratio		14.456	29.624	23.403	164.033
T. GPUSubdFit (s)		4.794	11.767	21.577	65.424
T. GPURaycast (s)		2.512	5.313	12.706	34.228
T. Total (s)		7.315	17.093	34.352	99.756
RMS error (%)	16×16	0.046	0.070	0.071	0.023
Compression Ratio		4.174	8.581	6.797	47.574
T. GPUSubdFit (s)		4.790	11.735	21.529	65.601
T. GPURaycast (s)		8.256	18.085	45.039	118.115
T. Total (s)		13.055	29.833	66.637	183.820

particularly when the sampling density of the displacement map is high. Moreover, although the ICM and IBM are generated using CPU, the computational complexity of the base control mesh generation is substantially decreased due to the guidance of the skeleton. The statistics are presented in Table 2. Although the skeleton needs users to sketch, this process normally takes less than 2 minutes.

When the sampling density becomes high, there will be more rays cast from each patch of the base surface, and it will capture more geometric details from the input model at the expense of a decrease in performance. As shown in Fig. 13(a), the average performance decrease is approximately 2x when the sample density increases from 4×4 to 16×16 for the three input models. In our experiments, tessfactor=16×16 can provide the most reasonable results for practical applications.

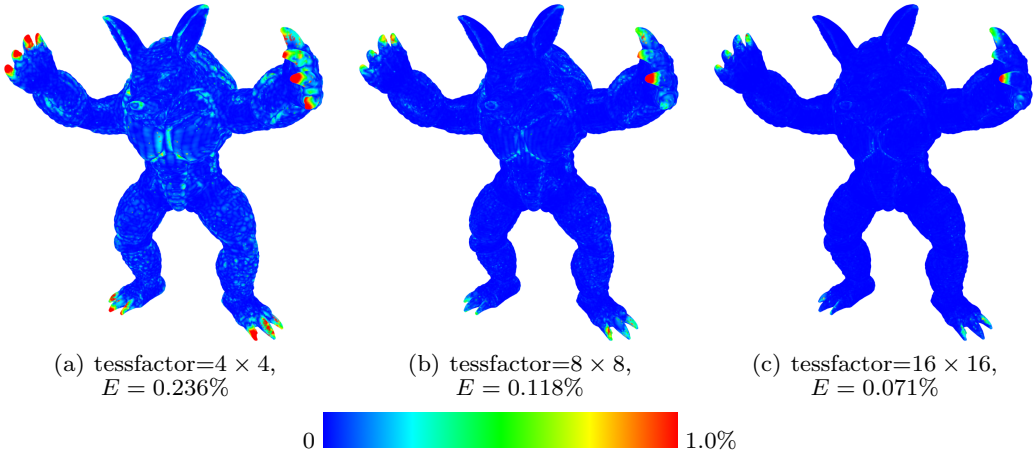


(a) total time costs (b) RMS errors

Fig. 13 The total time cost(left) and corresponding RMS errors(right) for constructing of the skeleton-guided displaced subdivision surfaces of the bunny, the armadillo, Venus and Neptune in different tessfactors. The total time includes base mesh generation, GPUSubdFit and GPURaycast.

There are still some artifacts, which are typically caused by a coarse ICM or sub-optimal OBM. For example, the fingers and toes of the armadillo are not perfectly reconstructed when sd=16×16 (Fig. 11(d)) because the ICM (Fig. 4(b)) represents the entire hand with a simple hexahedron. Although the hexahedron in ICM is subdivided and then optimized, its faces in the OBM (Fig. 6(d)) are still not able to expand into the small part of fingers and toes after GPUSubdFit. Consequently, there are not enough rays to sample the geometric details in these parts.

To quantitatively analyze the results, we measure the quality of the reconstructed mesh in Euclidean distance normalized to the diagonal length of the bounding box from the input model to the reconstructed mesh. We use the RMS errors of all the vertices to estimate the overall error of the



(a) tessfactor=4 × 4, $E = 0.236\%$ (b) tessfactor=8 × 8, $E = 0.118\%$ (c) tessfactor=16 × 16, $E = 0.071\%$

Fig. 14 The color-coded Euclidean distance, normalized to the bounding box, between the input model and reconstructed result.

entire mesh. The RMS errors of Venus, the bunny, the armadillo and Neptune in different tessfactors are shown in Fig. 13(b). Fig. 14 presents the color-coded errors of the armadillo, and the colored range is from 0 to 1.0%. As indicated by the results, the loss of detail of the reconstructed mesh decreases as the sample density increases from 4×4 to 16×16.

4.2 Compression

We also compare the output size of our representation with the input model without any additional compression techniques during encoding. The total size of our representation S can be calculated as $S = B + M$, where B and M are the sizes of the base mesh and the 2D texture, respectively. Since the IBM is subdivided from the ICM, the IBM can be reconstructed from its geometry G_{IBM} and ICM's topology T_{ICM}. Therefore, $B = G_{IBM} + T_{ICM}$. The 2D textures only require 1 channel for a scalar with a dimension of $M = F_{OBM} \times (sd+1) \times (sd+1)$, where F_{OBM} is the number of faces of OBM and sd is the sample density. We assume that the size of a vertex in geometry is three 32-bit float and a face in topology is 3 or 4 32-bit integer.

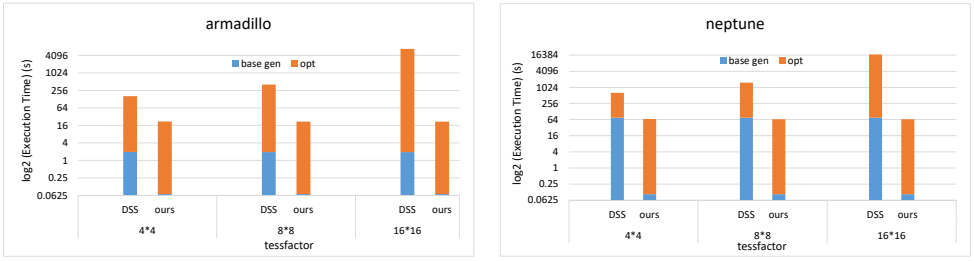
As indicated by the results presented in Table 2, the representation can compress the input model with abundant geometric details, saving the memory I/O on conditions of reconstructing the input faithfully. In the case of Neptune, it is compressed to nearly 50x, even when the sample density is 16×16. Thus, the proposed method has shown its potential for use in the film and game industries.

4.3 Comparison with Displaced Subdivision Surfaces

There are two processes of base surface construction in the method of displaced subdivision surfaces (DSS) [24]: base mesh construction through simplification and global optimization of the base mesh. Since their sampling stage can also be accelerated with hardware tessellation, we do not compare the performance of this stage.

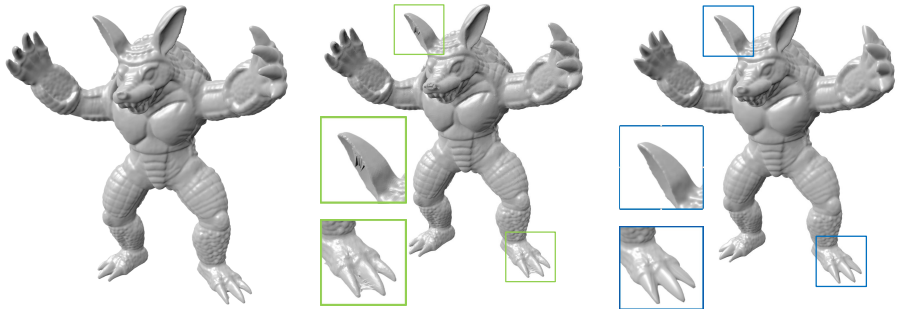
The DSS method use a prioritized sequence of edge collapse transformations[16] where edges are collapsed one by one, whereas the proposed method is performed guided by a skeleton, reducing the computational complexity. Moreover, the proposed GPUSubdFit exploits hardware tessellation, and exhibits better performance compared to DSS's base mesh global optimization [17]. Fig. 15 presents the execution time in different tessfactors, including the base mesh generation construction, the optimization and the total. When the tessfactor is 16×16, the execution time of both stages in DSS are more than 100x slower for both the armadillo and Neptune compared to the proposed method.

In the reconstruction results of the armadillo shown in Fig. 16, the majority of the geometric details of the model are recovered faithfully in both methods. However, in the two parts highlighted above, the incorrect displacements are detected, because there are vertices of high valences remained



(a) armadillo (b) neptune

Fig. 15 Base mesh generation, optimization time and total time consumption, illustrated on a logarithmic scale, compared with DSS.



(a) Original (b) DSS (c) Ours

Fig. 16 Our reconstruction result of the armadillo compared with DSS, where both of the numbers of base mesh faces are 256, and both of the sample densities are 16×16.

during simplification process, and their normals are not as smooth as ours. Note that the proposed framework shows compatibility with the DSS method in that the IBM can also be constructed through simplification[16] and then use GPUSubdFit for optimization.

5 Conclusion

We propose an efficient skeleton-guided displaced subdivision surfaces method, which converts an input model with abundant geometric details into a coarse base mesh with displacement maps, natively compressing the model and saving memory consumption and I/O bandwidth. The proposed method exploits the hardware tessellation and exhibits high efficiency. Therefore, the method is suitable for modern GPUs.

Features on a model are the most important parts that need to be recovered and reconstructed. Automatically detecting the features while simultaneously adaptively sampling them on the patches of the base surface, will be our future works.

Acknowledgements

We would like to thank Dr. Chen Xue for her helpful discussions, and also thank the anonymous reviewers who gave valuable suggestions to improve the quality of the paper. This work was supported by the National Natural Science Foundation of China under Grant Nos. 61472349 and 61170138.

References

1. Amanatides, J., Woo, A.: A fast voxel traversal algorithm for ray tracing. In: EG 1987-Technical Papers. Eurographics Association (1987)

2. Ara, K.: Introducing Mudbox. John Wiley and Sons Ltd. (2010)
3. Blinn, J.F.: Simulation of wrinkled surfaces. In: Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '78, pp. 286–292. ACM, New York, NY, USA (1978)
4. Burley, B., Lacewell, D.: Ptex: Per-face texture mapping for production rendering. In: Proceedings of the 19th Eurographics Conference on Rendering, EGSR '08, pp. 1155–1164. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2008)
5. Cashman, T.J.: Beyond catmull clark? a survey of advances in subdivision surface methods. Computer Graphics Forum **31**(1), 42–61 (2012)
6. Catmull, E., Clark, J.: Recursively generated b-spline surfaces on arbitrary topological meshes. Computer-Aided Design **10**(6), 350–355 (1978)
7. Chen, Z., Luo, X., Tan, L., Ye, B., Chen, J.: Progressive interpolation based on catmull-clark subdivision surfaces. Computer Graphics Forum **27**(7), 1823–1827 (2008)
8. Cohen, J., Olano, M., Manocha, D.: Appearance-preserving simplification. In: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '98, pp. 115–122. ACM, New York, NY, USA (1998)
9. Cook, R.L.: Shade trees. In: ACM SIGGRAPH 1984 Papers, SIGGRAPH '84, pp. 223–231. ACM, New York, NY, USA (1984)
10. De Aguiar, E., Theobalt, C., Thrun, S., Seidel, H.P.: Automatic conversion of mesh animations into skeleton-based animations. Computer Graphics Forum **27**(2), 389–397 (2008)
11. DeRose, T., Kass, M., Truong, T.: Subdivision surfaces in character animation. In: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '98, pp. 85–94. ACM, New York, NY, USA (1998)
12. Doo, D., Sabin, M.: Behaviour of recursive division surfaces near extraordinary points. Computer-Aided Design **10**(6), 356–360 (1978)
13. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: The International Journal for Geographic Information and Geovisualization **10**(2), 112–122 (1973)
14. Gumhold, S., Hüttner, T.: Multiresolution rendering with displacement mapping. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, HWWS '99, pp. 55–66. ACM, New York, NY, USA (1999)
15. Hirche, J., Ehlert, A., Guthe, S., Doggett, M.: Hardware accelerated per-pixel displacement mapping. In: Proceedings of Graphics Interface 2004, GI '04, pp. 153–158. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada (2004)
16. Hoppe, H.: Progressive meshes. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '96, pp. 99–108. ACM, New York, NY, USA (1996)
17. Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., Stuetzle, W.: Piecewise smooth surface reconstruction. In: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '94, pp. 295–302. ACM, New York, NY, USA (1994)
18. Jang, H., Han, J.: Feature-preserving displacement mapping with graphics processing unit (gpu) tessellation. Computer Graphics Forum **31**(6), 1880–1894 (2012)
19. Jang, H., Han, J.: Gpu-optimized indirect scalar displacement mapping. Computer-Aided Design **45**(2), 517–522 (2013)
20. Kaneko, T., Takahei, T., Inami, M., Kawakami, N., Yanagida, Y., Maeda, T., Tachi, S.: Detailed shape representation with parallax mapping. In: Proceedings of ICAT, pp. 205–208 (2001)
21. Keller, E.: Introducing ZBrush. John Wiley & Sons (2011)
22. Kobbelt, L.P., Bareuther, T., Seidel, H.P.: Multiresolution shape deformations for meshes with dynamic vertex connectivity. In: Computer Graphics Forum, vol. 19, pp. 249–260. Wiley Online Library (2000)
23. Krishnamurthy, V., Levoy, M.: Fitting smooth surfaces to dense polygon meshes. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '96, pp. 313–324. ACM, New York, NY, USA (1996)
24. Lee, A., Moreton, H., Hoppe, H.: Displaced subdivision surfaces. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '00, pp. 85–94. ACM, New York, NY, USA (2000)
25. Lee, H., Ahn, M., Lee, S.: Displaced subdivision surfaces of animated meshes. In: ACM SIGGRAPH ASIA 2010 Sketches, SA '10, p. Article No.37. ACM, New York, NY, USA (2010)
26. Lin, H., Jin, S., Liao, H., Jian, Q.: Quality guaranteed all-hex mesh generation by a constrained volume iterative fitting algorithm. Comput. Aided Des. **67**(C), 107–117 (2015)
27. Loop, C., Schaefer, S.: Approximating catmull-clark subdivision surfaces with bicubic patches. ACM Trans. Graph. **27**(1), Article No.8 (2008)
28. Nießner, M., Loop, C.: Analytic displacement mapping using hardware tessellation. ACM Trans. Graph. **32**(3), Article No.26 (2013)
29. Nießner Mner, M., Keinert, B., Fisher, M., Stamminger, M., Loop, C., Schfer, H.: Real-time rendering techniques with hardware tessellation. Computer Graphics Forum **35**(1), 113–137 (2016)
30. Pottmann, H., Steiner, T., Hofer, M., Haider, C., Hanbury, A.: The isophotic metric and its application to feature sensitive morphology on surfaces. In: Computer VisionECCV 2004, Part IV, volume 3024 of Lecture Notes in Computer Science (2004)
31. Stam, J.: Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '98, pp. 395–404. ACM, New York, NY, USA (1998)

32. Szirmay-Kalos, L., Umenhoffer, T.: Displacement mapping on the GPU - State of the Art. Computer Graphics Forum **27**(1) (2008)
33. Wang, L., Wang, X., Tong, X., Lin, S., Hu, S., Guo, B., Shum, H.Y.: View-dependent displacement mapping. In: Proceedings of ACM SIGGRAPH 2003, SIGGRAPH '03, pp. 334–339. ACM, New York, NY, USA (2003)
34. Wang, X., Tong, X., Lin, S., Hu, S., Guo, B., Shum, H.Y.: Generalized displacement maps. In: Proceedings of the 15th Eurographics Conference on Rendering Techniques, EGSR'04, pp. 227–233. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2004)
35. Yao, C.Y., Chu, H.K., Ju, T., Lee, T.Y.: Compatible quadrangulation by sketching. Comput. Animat. Virtual Worlds **20**(23), 101–109 (2009)