

Perception-motivated multiresolution rendering on sole-cube maps

Bin Sheng · Weiliang Meng · Hanqiu Sun ·
Wen Wu · Enhua Wu

Published online: 19 January 2013
© Springer Science+Business Media New York 2013

Abstract This paper presents a novel GPU-based multiresolution rendering on sole-cube maps (SCMs), which is a variant of geometry images built upon spherical parameterization. Given spherical parametrization of a manifold mesh, the sphere domain is gnomonically projected to a closed cube, which constitutes the 6-chart sole-cube maps. A quadtree structure of SCMs and normal map atlas are then constructed by using the regular re-sampling. Then, by packing the quadtree nodes into the SCMs texture atlas, a new parallel multiresolution rendering is processed on the latest GPU in two rendering passes: the multiresolution node selection in fragment shader; the triangulation in vertex shader followed by the node culling operation in geometry shader. The proposed approach generates adaptive mesh surfaces dynamically, and

B. Sheng (✉)

Department of Computer Sci. & Eng., Shanghai Jiao Tong University, Shanghai, China
e-mail: shengbin@cs.sjtu.edu.cn

B. Sheng

State Key Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

W. Meng

LIAMA-NLPR, Institute of Automation,
Chinese Academy of Sciences, Beijing, China

H. Sun

The Chinese University of Hong Kong, Hong Kong, China

W. Wu · E. Wu

University of Macau, Macao, China

E. Wu

Institute of Software, Chinese Academy of Sciences, Beijing, China

can be fully implemented in GPU parallelization. The proposed scheme alleviates the computing load of multiresolution mesh refinement on CPU, and our GPU-based multiresolution rendering is demonstrated with a variety of examples. Our user study confirmed that the visual quality of the SCMs multiresolution rendering, in comparison with the meshes/geometry images rendering, is also highly efficient especially for complex models in large-scale virtual environment.

Keywords Multiresolution rendering • Sole-cube maps • GPU processing • Mesh refinement

1 Introduction

Recently, demands to represent multiresolution 3D models are advancing rapidly in graphics applications. Even though the CPU performance, the main memory capacity, and the geometry processing performance are progressing dramatically, the main trend to maximize or fully utilize the GPU parallelization for level-of-details processing and rendering has been gaining momentum. However, due to the fact that the geometric models are usually represented by irregular triangle meshes, GPU-based multiresolution rendering is still a challenging task in graphics research. On the other hand, significant progress was made in decomposing meshes into a regular domain. And then geometry images [14] (GIM) can implicitly encode the connectivity information of a regular mesh, allowing standard image processing and compression techniques to be applied to geometrical models. Carr and Hart [6] and Purnomo et al. [32] show how square charts can be efficiently packed into 2D texture space in order to minimizing the memory space cost. Polycube maps [38] can warp and project geometry onto the quadrilateral faces of a manually-constructed cuberille exoskeleton. Ji et al. [21] proposes an adaptive mesh refinement based on polycube maps, which requires intensive user interaction for the polycube construction. Hernández and Rudomín [17] directly uses single-chart geometry image for rendering. Wang et al. [43] proposes the concept of polycube splines, which are built naturally upon the polycube maps.

Since the charts are constructed manually in the approach of polycube maps, even those small and intricate features in the charts can seriously challenge polycube construction. Our approach therefore aims at achieving the automatic construction of seamless geometry images, so as to facilitate multiresolution rendering on GPU by generating the quadtree-based texture atlas. Inspired by the harmonic parametrization of manifold splines [43], a new seamless geometry image structure based on spherical parametrization, namely the sole-cube maps (SCMs), is automatically generated. Since SCMs is the regular and seamless representation of 3D surfaces, it can be used to construct a hierarchical quadtree structure. Then all the nodes in the quadtree are packed into a mipmap-like texture atlas, and the processing pipeline is shown in Fig. 1. By taking advantage of SCMs, we can generate adaptive meshes dynamically via GPU processing, and render multiresolution surfaces.

Our multiresolution rendering approach includes two rendering passes, both performed on GPU. In the first pass, all multiresolution node data are sent to fragment shader in the form of SCMs atlas textures for the multiresolution node selection. In the second pass, the selection result is applied to the rendering vertices

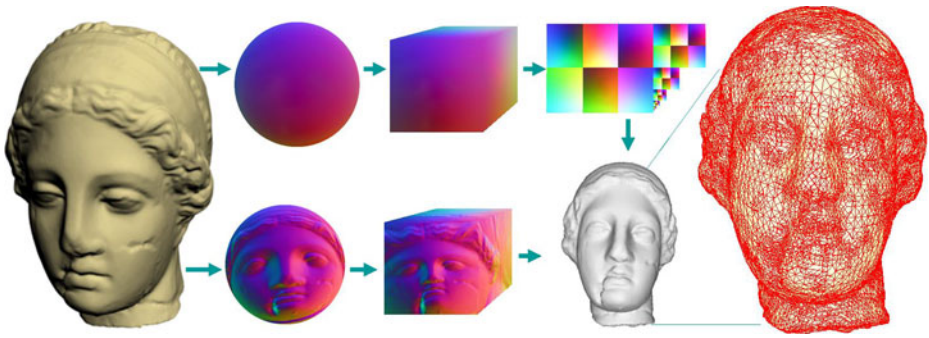


Fig. 1 GPU-based multiresolution rendering on sole-cube maps

in the vertex shader and geometry shader, to triangulate the selected nodes and cull the inactive nodes respectively. In this way, we can use SCM to generate adaptive watertight meshes dynamically.

The main contribution of this paper is the new multiresolution rendering approach, which can perform the parallel GPU rendering based on sole-cube maps. Given the regular mesh obtained from the sphere parametrization, our approach can handle the manifold mesh with flexible patterns, rather than limited subdivision refinement patterns [4, 5] of the vertex-vertex connections. Since the GPU shaders operate in parallel, our approach can significantly accelerate the computation, hence reducing the workload of CPU. We demonstrate the SCMs rendering approach on several manifold models with up to millions of vertices. We also show the advantages of increased flexibility of our multi-grained hierarchy, as well as the performance of our current system.

2 Related work

We briefly review the related work on spherical parameterization and sampling, geometry images and GPU rendering.

Spherical parametrization and sampling Surface parameterization has been an active topic in computer graphics. However, a detailed surface parameterization technique survey is beyond the scope of this paper. A nice survey of parameterization methods can be found in [11] or refer to the literature. Parametrization of a closed manifold mesh with genus 0 should be processed on its natural domain: the unit sphere S^2 . Previous work addressed the problems in generating spherical parameterizations [1, 12, 43]. The earlier work [15, 24] attempted to generalize barycentric coordinates for planar parameterization of 3D meshes [10, 39]. Meanwhile, another problem in spherical parameterizations is the spherical map and its sampling [35], where a lot of spherical map projections [42] have been developed by researchers in astronomy. The spherical maps are widely used in computer graphics, including longitude-latitude map [2], equal-area cylindrical map [37], cubemap [13], dual paraboloid map [16], and octahedron [9] and icosahedron-based spherical maps [34].

However, most parameterizations have adopted the six-face cubemap, due to its computational simplicity and memory-friendly rectilinear structure.

Geometry images Recent work has demonstrated the feasibility and utility of decomposing meshes into a regular domain. Geometry images (GIM), initially presented by Gu et al. [14], can create the near-regular meshes. Sander et al. [33] extended GIM to multi-chart patterns. Although multi-chart GIM can reduce distortions during the parameterization, it also produces undesirable seams due to the irregular boundary of each chart. Recently, the rectangular MCGIM [7] (RMCGIM) was used to exploit rectangular patches onto tile surfaces, guaranteeing a one-to-one pixel correspondence across chart boundaries [27, 31] avoided the seams problem on meshes by using the sphere GIM or other homeomorphic geometries (e.g. cube, octahedron). In addition, [30] attempted to improve the compression ratio of GIMs using wavelet techniques. [26] proposes an automatic PolyCube-Maps construction scheme based on the mesh feature detection, where the parametrization quality is limited. [18] and [21] propose the dynamic LOD processing for geometry images on GPU. Wang et al. [43] created the polycube splines, a systematic way to construct polycube maps for surfaces of arbitrary topology. Recently [25] introduced the TileTree, which is a compact data structure for texture mapping surfaces.

GPU Rendering on meshes The long history of high-performance ray tracing on the CPU is summarized in [41]. While the GPU outperforms the CPU on streaming kernels such as ray intersection, the latter is much more efficient at maintaining and traversing complex data structures. State-based ray tracing on GPU runs almost 50 % faster, when the cells of the uniform grid hold a precomputed distance transform (called “proximity cloud”) that allows the traversal kernel to skip large empty regions of space [23]. Apart from rendering with GPU, subdivision algorithms have been proposed to rebalance the workload between CPU and GPU, and take advantage of parallel execution streams in programmable graphics hardware [3, 36]. However, their recursive definition imposes a non-trivial CPU overhead, hardly can be hidden in interactive applications. Hu et al. [19] presented the view-dependent LOD algorithm for vertex-level mesh refinement that operates entirely on the GPU. Boubekeur and Schlick [4, 5] used a vertex shader to do a generic mesh refinement on GPU. The key idea is to store the possible refinement configurations of a single triangle on the GPU. Such a mechanism, however, restricts the refinement within a limited number of pre-computed patterns. It performs poorly on the meshes with arbitrary topology, since these patterns only examine triangle meshes.

The proposed SCMs rendering approach handle the adaptive mesh simplification in a parallel manner, to control view-dependent visual quality. Compared with the polycube maps [23, 44, 45] which needs labor-intensive interactions, our SCMs rendering builds the geometry-to-image maps with spherical parameterization. Such a construction can be totally automatic, saving considerable user interactions. In addition, polycube maps build the cube connectivity by user interactions, the boundary of each texture chart usually have flexible connectivity patterns. Such a flexibility often causes the polycube maps to construct the texture charts with flexible adjacency, resulting in increased intricacy in the algorithm, hence hindering the searching and indexing of neighboring nodes in adjacent charts, furthermore

reducing the overall-timing performance of the multiresolution rendering. This is especially true when compared with our SCMs rendering. The SCMs rendering delivers real-time frame rates, with temporally coherent visual quality. Our user study has shown how our approach increases perceived quality in comparison with the other mesh-only methods. Our SCMs multiresolution rendering runs much faster than previous mesh-only methods, but producing the similar realism as original mesh rendering.

3 Construction of SCMs

The construction process of SCM includes two steps: parameterizing a model to a unit cube surface, and generating the seamless cube maps. The process of cube parameterization is shown in Fig. 2. Here we use Praun's method [31] to achieve sphere parameterization of a closed genus-0 manifold model, then we gnomonically project the sphere map onto a cube. We give the mathematical form of the projection onto the upper cube face, denoting ϕ as the azimuth angle and θ as the polar angle:

$$\mu = \frac{1}{2} + \frac{1}{2} \tan \theta \cos \phi, \quad \nu = \frac{1}{2} + \frac{1}{2} \tan \theta \sin \phi \quad (1)$$

Now the SCMs can be generated based on the parameterized coordinates and the topological connectivity, as shown in Fig. 3. In order to obtain the right-angled cube arris between adjacent faces of SCM, as shown in Fig. 2, we decompose the *span triangles* spanning two or three faces. Specifically, there are six types of parameterized triangles on cube faces, as shown in the Fig. 3. Further, we can see that *type 6* is a trivial case since all the vertices belong to the same face. In Fig. 3, *A*, *B*, and *C* are the triangle vertices, *D*, *E*, and *F* are the intersections of *BC*, *CA* and *AB* with the cube arris (if exist), and *P* is the corner of cube which is inside the triangle *ABC* (if exists). Thus, we record all these points and decompose the triangle to several planar parts before sampling the SCMs. For example, in *type 3*, quadrilateral *BCEF* and triangle *AFE*, rather than *ABC*, are drawn to generate SCMs; in *type 4*, we draw

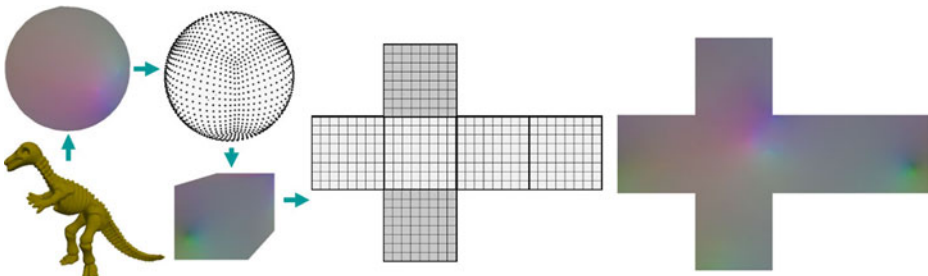


Fig. 2 Illustration of SCMs generation: unlike polycube maps, our sole-cube maps are generated without user interaction

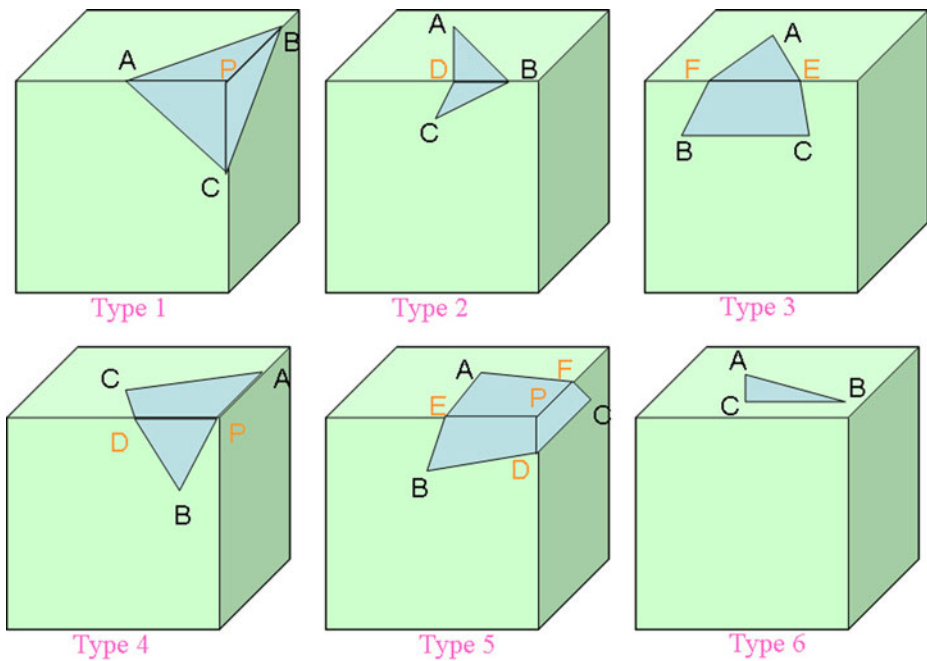


Fig. 3 Triangle types after cubeparameterization

$APDC$ (P is now superposed with F) and PBD . The position and normal values of D , E , F , are calculated by linear interpolation. Finally, we get a cube with right-angle faces as shown in Fig. 4.

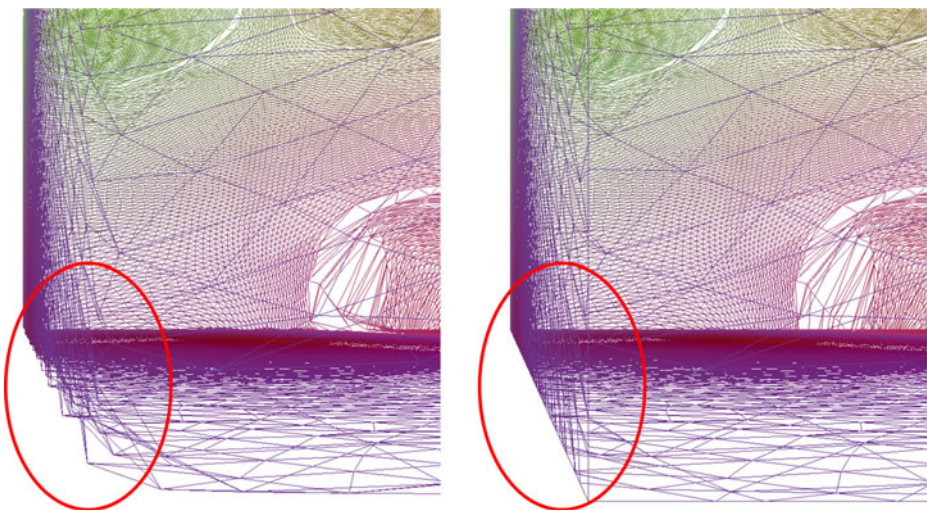


Fig. 4 Generation of right-angle faces parameterized cube

4 Construction of quadtree SCMs textures

After packing 6 cube faces into 6 square charts, we use the 2D grid sampling to generate the geometry images. Our goal is to build the quadtree-based SCMs, in square charts, which are flexible to be packed for complex models.

4.1 Quadtree node construction

The hierarchical data in our multi-resolution rendering are organized in the quadtree structure of sole-cube maps. Construction of a quadtree requires the sampling of square charts in $(2^m + 1) \times (2^m + 1)$. Therefore we generate the 3×3 blocks over the square chart. Figure 5 shows the the node structure in geometry space, where n is the normal of the center vertex, and r is the radius of the node's bounding sphere. In order to realize the visibility culling in SCMs rendering, we also recode the normal cone for each quadtree node. These quadtree nodes are the elementary rendering primitive in our rendering approach. We construct the quadtree-based SCMs in a bottom-up manner, in which the radius of the bounding sphere, the normal and the angle of normal cone (see Fig. 5), are computed as node attributes, which are utilized in the following processes. In our multi-resolution rendering, T-junction may appear when resolution levels of the two neighboring squares are not the same. Considering that when a mesh with T-junctions is rendered, it may produce some artifacts like cracks or jags, we choose the restricted quadtree triangulation [40], to tessellate parametric surfaces, hence avoiding the artifacts. We constrained the subdivision by limiting the difference of the levels of adjacent quadtree nodes within one, as shown in Fig. 7.

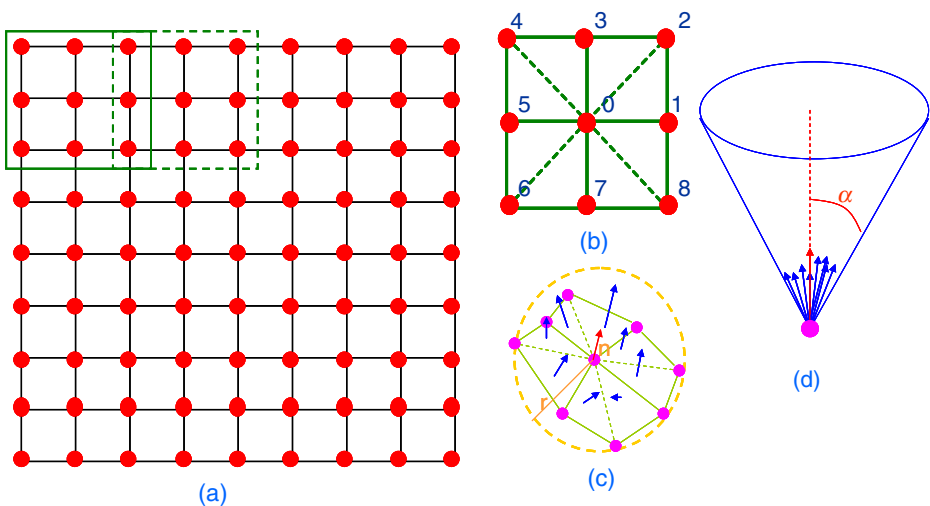


Fig. 5 The structure of a P-Quadtree node: **a** P-quadtree node array, **b** parameter space, **c** object space and **d** normal cone

To ensure that the quadtree is confined in our multiresolution rendering, we construct the quadtree by pre-computing the geometry error of the nodes. The error of a node is greater not only than that of its own children, but also than its adjacent nephew node. For example, the error of node *A* must surpass that of the nodes *B* and *C*, the 2 southern nephew nodes of *A* (see Fig. 7). The nephew nodes' resolution levels determine whether the edge vertex is activated or not. According to the seamless sole-cube maps, the nephew of boundary nodes along the chart boundary can be obtained from adjacent charts, which can be processed in the same way as the interior nodes.

4.2 Quadtree stacking

In order to complete the multiresolution node selection on the fragment shader, all the quadtree nodes are packed into the SCMs atlas. Since the SCM atlas contains square charts, we stack all the levels of quadtree side by side. The stacked pattern is shown in Fig. 6. A quadtree node, the 3×3 block of pixels in SCMs, is mapped onto one pixel in the SCMs atlas. Suppose the SCMs atlas is in $(2^m + 1) \times (2^m + 1)$, the number of the finest level nodes (level 0) is $2^{m-1} \times 2^{m-1}$, and level 1 nodes are in a quarter size of the level 0 nodes, and so on, the root's size is 1. With 6 square charts packed together, all of the quadtree nodes can be stacked in a rectangle texture with a ratio of 1 : 2.25 (see Fig. 6). Meanwhile, all the node's attributes for multiresolution node selection are organized in the texture atlas. There are four textures in our algorithm: the normal map, the geometry map, the SCMs parameter map (the node's error, normal cone and the radius of the bounding sphere) and the index map (to address its parent node in the fragment program). The geometry map and the normal map are used to store the central position and the normal of the nodes, which are computed by the geometry image and the normal map of the vertices. These maps are used for backface culling in our multiresolution rendering. The node's error in SCMs parameter map is the metric for multiresolution node selection. Each node is processed independently in the fragment shader.

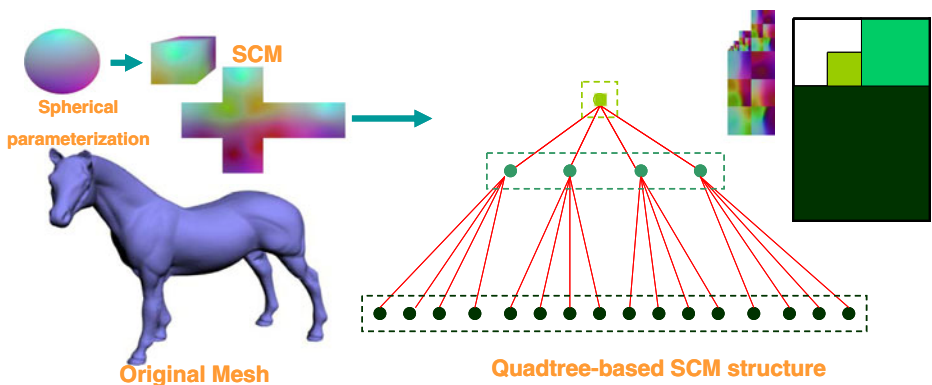


Fig. 6 Quadtree stacking of SCMs

5 GPU-based multiresolution rendering

Our GPU-based multiresolution rendering is composed of two rendering passes: the first pass tests the nodes in fragment shader, and the 2nd pass applies the vertex triangulation by vertex texturing in vertex shader, and culls the inactive nodes in geometry shader.

5.1 Multiresolution node selection on GPU

The SCMs atlas flow to the fragment shader, whose pixels are *1-to-1* mapped to the texels. The error threshold for multiresolution node selection is computed from the viewpoint on CPU, and is delivered to a parameter register. A fragment program is used in the multiresolution node selection. The testing result is stored in float-point for vertex texturing. At each fragment testing, we discard the null texels by killing the corresponding fragment. The null texels, whose indices are set at (0, 0), are tested from the parent's index. For the valid nodes, their error threshold for multiresolution selection test is read from the parameter register. If its error exceeds the threshold, the node is discarded; otherwise the test is passed.

The multiresolution selection can be seen as an execution of a *LOD cut* through the quadtree, by the criterion that each root-to-leaf path contains precisely one node—the first one to have passed the selection. There is a problem of redundant nodes in the selection test: if the parent node pass the selection test, as the finer level quadtree

Algorithm 1 Multiresolution node selection (fragment shader)

```

1: Read the threshold and the viewpoint parameters from registers;
2: Fetch the parent node's address  $\rightarrow p$ ;
3: if  $p$  is (0, 0) then
4:   Kill the fragment; {NULL Texel}
5: end if
6: Lookup the parent's position error to  $Error$ ;
7: if  $Error \leq$  the threshold then
8:   Kill the fragment; {The parent node is possibly selected}
9: end if
10: Fetch the current node's error  $\rightarrow Error$ ;
11: if  $Error > threshold$  then
12:   Kill the fragment; {The current node fails the selection test}
13: end if
14: Fetch the angle of normal cone to  $\alpha$ ;
15: Fetch the radius of the bounding box;
16: Fetch the current node's position and the normal;
17: Estimate the angle of view cone to  $\beta$ ;
18: Estimate the angle between view vector and normal to  $\theta$ ;
19: if  $\alpha + \beta + \theta < \pi/2$  then
20:   Kill the fragment; {Backface culling}
21: end if
22: Output to the buffer; {All tests are passed}

```

node, the children nodes will definitely satisfy the selection criteria. However, for the rendering's purpose, these children nodes must never be rendered. In order to prevent the redundant nodes, two selection tests are conducted for each node: one for its parent, and the other for itself. If the parent of a node passes the test, the current node is redundant and therefore dropped. We use an index map to address the parent node in the fragment program. If a node passes the multiresolution node selection, visibility culling is performed using the normal and the normal cone (see Fig. 9). The backfaces are determined by the criterion in [28]. Specifically, the pseudocode of our multiresolution node selection is given in Algorithm 1.

5.2 SCM triangulation

After the multiresolution node selection, the output selection buffer is applied to the quadtree nodes in the second rendering pass, to triangulate the quadtree nodes and discard redundant nodes in the vertex shader and the geometry shader respectively.

In this pass, the nodes are drawn as triangle fans. The multiresolution selection buffer should be accessible for the nodes to ensure that the selection result can be applied on the nodes. However, reading the buffer back into the host memory and sending the result back to the graphics card would be unacceptably expensive. In order to cost-effectively generate selection results of the quadtree nodes, we map the buffer to vertices of the nodes by using vertex texturing. The vertex shader process each vertex independently, imposing triangulation on each node according to its local information. The quadtree triangulation in our approach is restricted by limiting the difference of the levels of adjacent quadtree nodes within one (Fig. 7). Only the 4 edge vertices (1, 3, 5 and 7 in Fig. 5), located in the middle of the node's edge, need to be checked, while the other vertices can be spared this test. Since deleting or adding

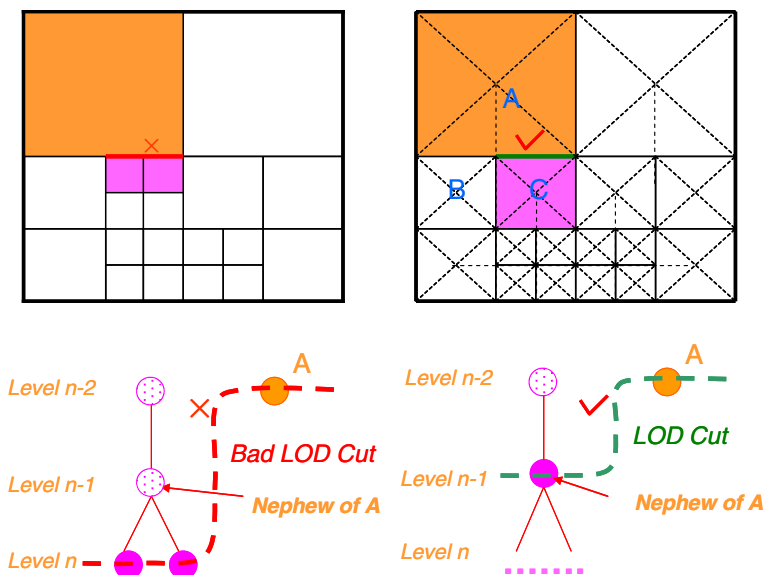


Fig. 7 Restricted quadtree triangulation

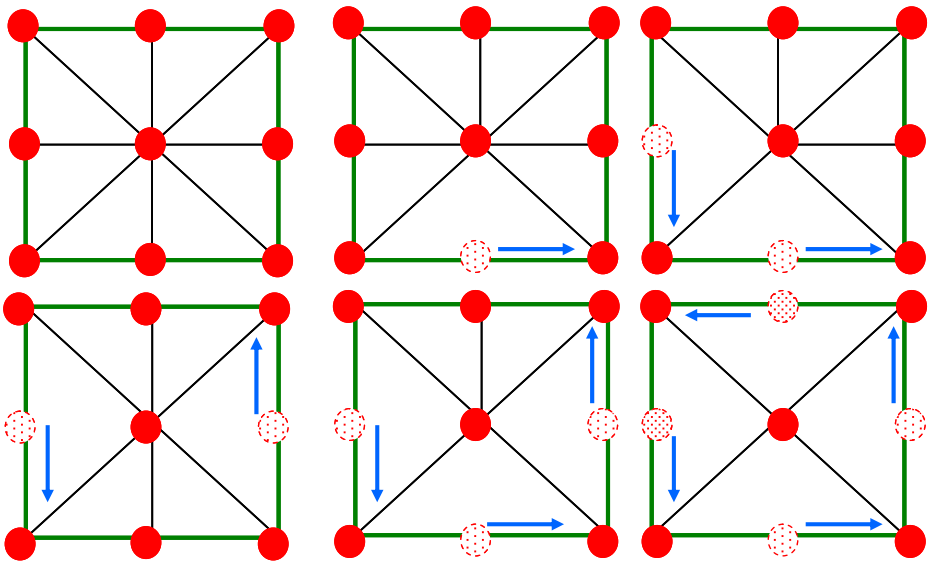


Fig. 8 The 6 triangulation patterns of the P-Quadtree Node

a vertex in a 3×3 quadtree node impedes the simple triangulation, we respond by moving some vertices, and sending all the 9 vertices of the node to GPU by drawing a triangles fan. However, the 4 edge vertices have two different position attributes: one for itself, and the other for its neighboring corner vertex. If an edge vertex is not activated, it is moved to the position of that corner, giving the appearance that the vertex has disappeared. That corner's position can be sent to the vertex shader by a generic attribute input register. Hence, the six triangulation patterns are formed for SCMs quadtree nodes, as shown in Fig. 8. We check the state of an edge vertex's adjacent nephew nodes adjacent (see Section 4.1) to see whether it is activated. Using the restricted quadtree (only 1 level below the current level), only two adjacent nodes

Algorithm 2 Triangulation and node culling (vertex shader & geometry shader)

```

1: {Vertex Shader: Triangulation}
2: Fetch the multiresolution selection result  $\rightarrow state$ 
3: if state is selected then
4:   Read the neighboring children's address  $\rightarrow a_0$ ;
5:   Fetch the selection result of the neighboring child1  $\rightarrow state1$ ;
6:   Fetch the selection result of the neighboring child2  $\rightarrow state2$ ;
7:   if state1 is notselected AND state2 is notselected then
8:     Read the corner's position;
9:     Move the vertex to the corner;
10:  end if
11: end if
12: {Geometry Shader: Node Culling}
13: Fetch the multiresolution selection result  $\rightarrow state3$ ;
14: if state3 is notselected then
15:   Cull the vertex; {discard the current vertex}
16: end if

```

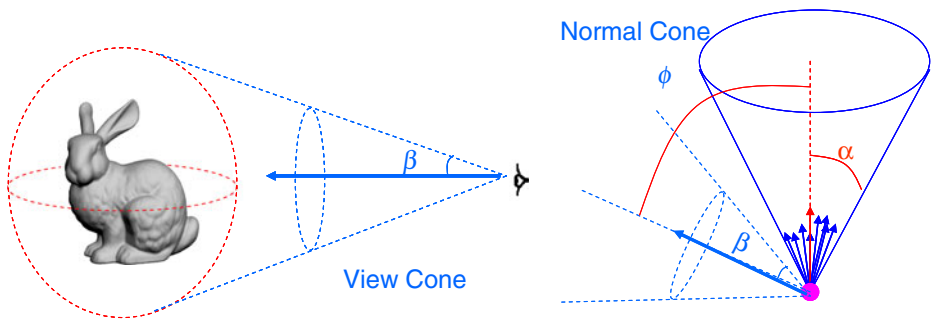


Fig. 9 Backface culling

will need to be tested. The states can be read from the output buffer, and determined by two sets of coordinates for the edge vertex.

By now, nodes culling can be performed in the geometry shader (see Algorithm 2). All vertices of the node are mapped to the same pixel in the buffer, for the selection result. Each vertex is tested by the pixel's (or texel's) value. Those who have passed the test are active quadtree nodes, those who failed should be discarded. If a node is culled, all its vertices are culled. The result of the geometry shader is then output to the rasterization. The lighting is computed in the fragment shader, and the final image of the model is generated (Fig. 9).

6 Results and discussion

The proposed SCMs approach is developed on 2.6GHz Intel Core(TM) CPU, together with nVidia GeForce 8800 GPU and 2G RAM. The GPU program described above has adopted a NVIDIA Cg program. Unlike polycube parametrization, our approach involves the automatic construction of texture charts on meshes.

6.1 User study

In order to assess the perceptual quality of our SCMs multiresolution rendering of complex models, we have performed validation study with two goals:

- investigate if there would show the perceptual difference between the original mesh rendering and our SCMs model rendering on GPU;
- quantify rendering quality of the SCMs approach with other view-dependent mesh rendering techniques from different simplified levels.

Guided by our goals, we have tested three approaches for multiresolution rendering of complex models: the mesh simplification [20, 29]; direct resampling of geometry images [14]; our SCMs multiresolution rendering. The inputs were the moving views in reference to the testing objects. We adopted user studies to evaluate the perception efficiency and visual quality. We invited 25 computer users who have little VR/graphics-related experience, to score the experimental results of our system. The statistics were made on the collected data to eliminate the individual instability. Our perceptual tests were performed on the common PC system, and scenes were

rendered at 500×500 pixels. The model sequences were presented on a black background of a CRT monitor using the pixel resolution 1024×768 at 75Hz, and the rendering conditions (lighting/background) are the same for all the tests.

Rather than using a rating task in which participants are first shown a sequence of moving model and then asked to rate them, we used a more systematic approach. For the four complex models (bunny, laurana, venus, armadillo) and three multiresolution algorithms (in *AB* and the inverse *BA* orders), we have tested $4 \times 3 \times 2$ (24) trials in total. For the first analysis, we checked the similarity score of the multiresolution rendering algorithms, by counting how many times it was (*incorrectly*) chosen as the ground truth, when compared to the original mesh model. The similarity scores are shown in Fig. 10 for the comparisons. The testing results have shown that the participants found significantly harder to identify the visual difference with ground truth, using SCMs multiresolution rendering than using direct resampling in geometry images. We also identified that our SCMs rendering has the same scale of similarity with the mesh multiresolution rendering, as they are mapped as view-dependent rendering as well.

Due to the fact that the model simplification is processed by the specific condition of view-dependent rendering, we have conducted the experiments using two groups of testing models: mesh multiresolution rendering, and our SCMs multiresolution rendering. The two testing groups used the same input of 3D models, but different representations (meshes, SCMs) and simplified processes. Four multiresolution levels are used in the testing groups: the numbers of rendering primitives (vertices) are 70 50, 30 and 10 % of that in the original 3D models. From the original model, the viewing distances of simplified models are adjusted accordingly.

The participants investigated the rendering sequences group by group, and rated each sequence according to how easy to capture the content and how impressed the rendering results. The score is given from 0 to 10 (worst to best). The participants were not aware of the simplified levels of the moving object, and they can repeat any motion sequences in the same group if they prefer. The mean opinion score (MOS) is obtained by averaging the scores given by each user, as follows:

$$MOS_k = \sum_{i=1}^{N_{\text{user}}} \frac{Mark_{i,k}}{N_{\text{exp}}} \quad (2)$$

where k is the motion sequence's id, N_{user} is the number of users (experimentally 25), and $Mark_{i,k}$ is the mark given by the i th user for the k th testing samples.

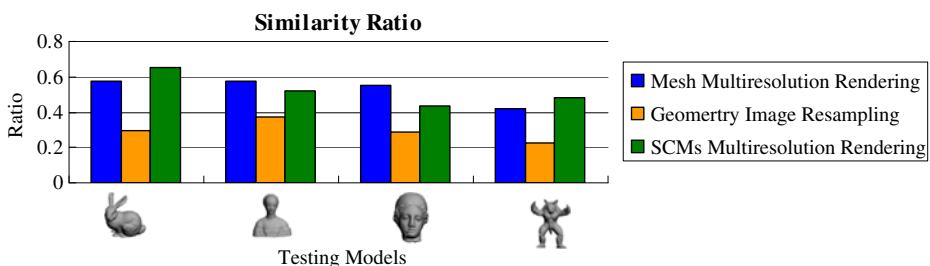


Fig. 10 Similarity scores for four test models (bunny, laurana, venus, armadillo), using the mesh multiresolution rendering, direct resampling geometry images, and our SCMs multiresolution rendering

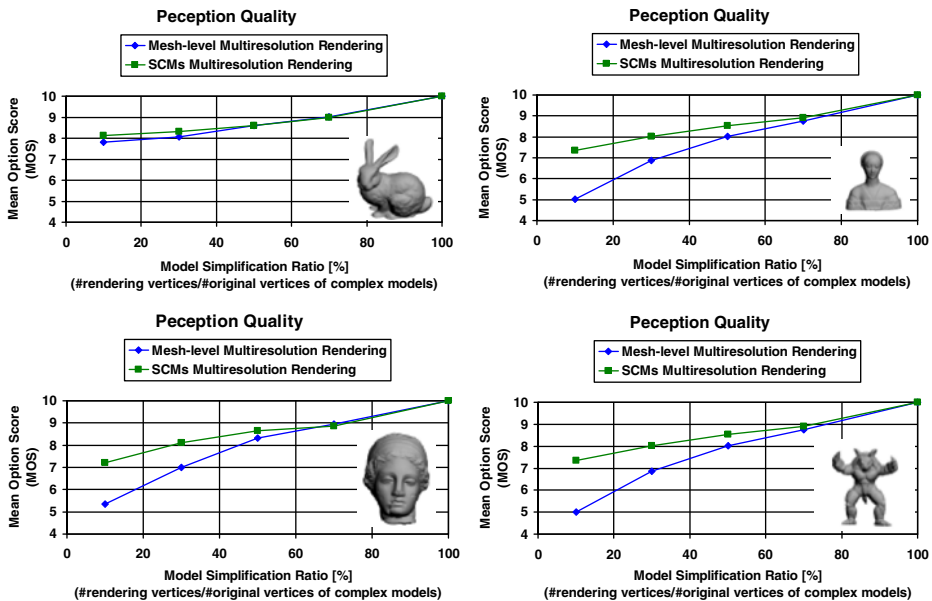





Fig. 11 Comparison of MOS between SCMs multiresolution rendering and the mesh multiresolution rendering

Figure 11 shows the curves of MOS for the four rendering models in interactive environments, with respect to the multiresolution levels of complex models. It is shown that for both SCMs rendering and mesh multiresolution rendering of 3D models, the Mean Option Score (MOS) is decreased when the number of rendering vertices is reduced. The SCMs multiresolution rendering has the higher MOS over the other tests. Especially when the model simplification ration becomes smaller than 20 %, the perception quality of SCM multiresolution rendering is much better than that of the mesh multiresolution rendering for most testing models. The necessary information for human perception is preserved better in the SCMs multiresolution rendering. The SCMs rendering provides consistent visual perception without temporal flick, due to the restricted quadtree simplification in regular quadtree domain. The rendering of 3D simplified models is commonly used in VR applications, to reduce the heavy overhead in computation, transmission and large volume of storages. We observe that the SCMs multiresolution rendering can support better perceptibility and subject quality of moving objects, especially for the rendering of 3D models simplified largely as shown in Fig. 11.

Table 1 ANOVA comparison of the mesh-level multiresolution rendering and SCM rendering

Source	df	Mean square	F	p
#Rendering method (SCM/Mesh-level)	1	17.828	76.594	<0.001
#Object type	3	4.418	18.979	<0.001
#Simplification ratio	4	64.728	278.082	<0.001

Table 2 Algorithm performance of our multiresolution node selection

		Models		
		Bunny 	Dragon 	Armadillo 
#Charts		6	6	6
SCMs size		6×257^2	6×513^2	6×513^2
Average rendered nodes		116,670	196,560	212,864
Our GPU	Times(ms)	1.98	2.68	3.06
implementation	CPU load	8–10 %	12–18 %	12–20 %
Our CPU	Times(ms)	10.6–15.2	16.5–21.0	17.1–23.2
implementation	CPU load	~ 31 %	~ 45 %	~ 48 %
Tree traversing	Times(ms)	6.42–8.83	10.4–18.0	12.5–19.7
on CPU	CPU load	28–41 %	42–55 %	43–58 %

We adopt the ANOVA analysis for validating the difference between SCM rendering and mesh-level rendering. The statistical analysis was performed with Statistical Package for Social Sciences version 17.0 (SPSS). The statistic result is shown in Table 1, we find that the test statistics for the rendering method were statistically significant. The result for the comparison of SCM rendering with the mesh-level rendering was $F = 75.574$ and $p < 0.001$ (a p value of 0.001 or less denotes a statistically significant result). From the test statistics, there is a significant difference between mesh multiresolution rendering and SCMs models. Additionally, for the comparisons of different simplification ratio and different objects, the results were also statistically significant. From this we can conclude that SCM rendering had significantly higher perception quality than that of mesh-level rendering ($P < 0.001$). These results demonstrate that SCM rendering may in fact be exploited to significantly reduce the geometrical complexity of complex models without having visible affect on the observer's s perception of the scene.

6.2 Rendering efficiency

Taking advantage of the regular structure of a quadtree, our multiresolution rendering implementation induced a lower run-time overhead (the rendering timing results are shown in Table 2). In addition, the GPU-based SCM rendering is more efficient than the tree traversing approach on CPU. In the proposed algorithm, when the multiresolution node selection is working in the first pass, the nodes are prepared on CPU simultaneously. The sequential point tree (SPT) [8] can be integrated into our

Table 3 Performance comparison of the multiresolution rendering on SCM with [22]






Models	Bunny 	Venus 	Head 
#Charts	6	6	6
Samples	40 k	134 k	376 k
P_quadtree SPM	25 <i>fps</i>	16 <i>fps</i>	11 <i>fps</i>
[22]	9.09 <i>fps</i>	2.38 <i>fps</i>	0.89 <i>fps</i>

Table 4 Packing performance for Polycube maps, multi-chart GIM and our Sole-cube maps

Texture packing rate of models	Bunny 	Laurana 	Armadillo 
Sole-cube maps	~ 88.9 %	~ 88.9 %	~ 88.9 %
Tarini et al. [38]	~ 59.5 %	~ 67.5 %	~ 65.8 %
Sander et al. [33]	~ 43.2 %	~ 55.6 %	~ 50.9 %

framework so as to reduce the quadtree nodes in GPU in the second pass. This can further improve the performance of our SCM rendering.

Table 3 compared the performance of our approach with that of the algorithm used in [22]. Our GPU based multiresolution rendering processing was initially applied to polycube maps [38] and multi-chart geometry images [33]. Then we recognized that the level-of-detail techniques inherently prefer to square texture patches, given the compact/flexible hierarchical construction, whereas polycube maps [38] project geometry on rectangular patches. Furthermore, multi-chart geometry images divide and map the mesh into texture patches in irregular shapes. To convert the non-square patches into square ones is less-than-efficient for multiresolution rendering, in terms of texture packing and indexing. Therefore, our method for converting irregular mesh into a cube-map structure can improve the rendering efficiency, both on the storage cost and processing load.

Table 4 compares SCM's performance to per-quadtree packing rates in other approaches. Table 5 shows the timing comparison of our SCMs rendering and simple GPU vertex buffer rendering. Although such averages may conceal the algorithmic complexity, they nonetheless show that the SCM has much more compact texture packing than the polycube maps and multi-chart geometry images that produce quadtree structures. Another reason for choosing SCMs over polycube map for rendering is that polycube maps manually construct the texture charts with flexible adjacency. This will make the neighboring nodes in adjacent charts too difficult to be searched and indexed, given the intricacy of the algorithm. Also, it undermines the overall performance of the multiresolution rendering. This is especially true when compared with our SCMs rendering. Figure 12 presents the rendering frame rates from different resolutions for test models.

Our approach mainly works on the genus-0 manifold meshes, due to the inherent requirement of spherical parameterization. But with some simple mesh completion manipulations for non-manifold surfaces, we can also use SCM for multiresolution rendering (e.g. SCMs-based packing of Laurana model in Table 4). In our current system, the chart size of our SCMs and normal map atlas ranged from 33×33 to 513×513 . The visual qualities of the reconstructed meshes depend on the resolution

Table 5 Timing statistics for SCM rendering with traditional GPU rendering

Input meshes	Bunny	Tyra	Horse
# vertices	6,291,456	6,291,456	6,291,456
Traditional GPU rendering (FPS)	29.67	29.65	29.35
SCM rendering (FPS)	48.35	51.65	48.16

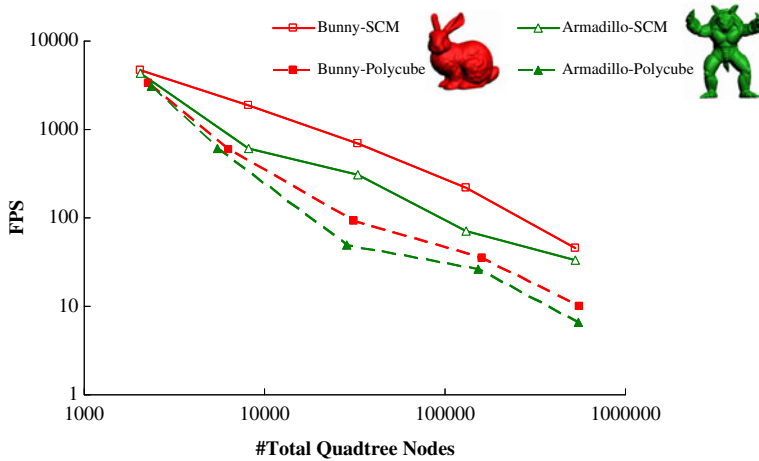


Fig. 12 Comparison between the frame rate obtained with multiresolution rendering of Polycube maps and that of our new Sole-cube maps (SCM). Note that our SCM can further accelerate the rendering due to employing simple neighboring node searching

adopted, especially for those highly detailed models. Our approach is able to generate adaptive meshes dynamically according to the viewpoint. Figure 13 shows the multiresolution rendering of these models.

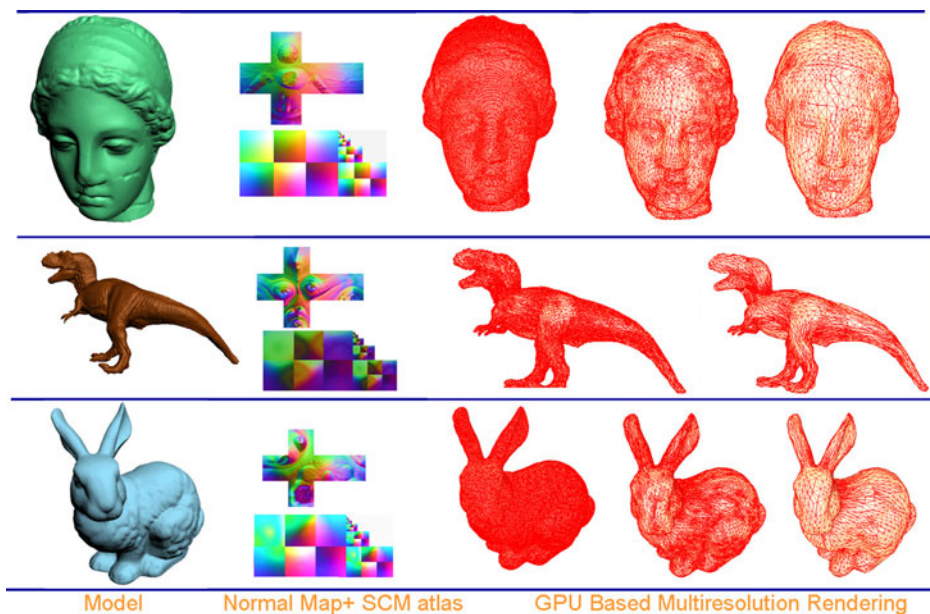


Fig. 13 The SCMs multiresolution rendering results with GPU implementation

7 Summary

In this paper, we have developed a novel multiresolution rendering on GPU, in which the SCMs texture atlas proved to be robust and efficient, with a reduced computing load of CPU. The quadtree structure is created based on seamless SCMs atlas, which is 3D surface representation in parameter space, combining the features of geometry images and poly-cube maps.

Our SCMs rendering approach generates adaptive meshes dynamically, and is fully implemented on GPU. Our approach does not need to construct additional lookup table to index texels in each chart, therefore further improves the efficiency of multiresolution node selection and reducing the computing load on CPU. We find that the quantization error of the geometry images can be reduced by employing the float-point pixel values. Our user study validated the visual quality of the SCMs multiresolution rendering, and evaluated the efficiency contribution of our approach by adaptively simplifying the models on GPU. Compared with the multiresolution rendering techniques based on polycube maps, ours can map the input meshes into SCMs automatically. We will further investigate the application of sole-cube maps, and try to extract the geometrical features from SCMs into our system, in order to apply the multiresolution rendering in interactive manipulations.

Acknowledgements We would like to thank the anonymous reviewers for their valuable comments. This work is supported by the National Basic Research Project of China (No. 2011CB302203), and the National Natural Science Foundation of China (No. 61202154, 61133009, 61202324, 61271431), RGC research grant (ref. 416311), UGC direct grant for research (no. 2050485, 2050454). This work is also partially supported by the Open Projects Program of National Laboratory of Pattern Recognition, and the Open Project Program of the State Key Lab of CAD& CG (Grant No. A1206), Zhejiang University.

References

1. Alexa M (2000) Merging polyhedral shapes with scattered features. *Vis Comput* 16(1):26–37
2. Blinn J, Newell M (1976) Texture and reflection in computer generated images. *ACM Commun* 19(10):542–547
3. Bolz J, Schröder P (2003) Evaluation of subdivision surfaces on programmable graphics hardware. <http://www.mutires.calte-ch.edu/pubs/GPUSubD.pdf>. Accessed 7 Aug 2012
4. Boubekeur T, Schlick C (2008) A flexible kernel for adaptive mesh refinement on GPU. *Comput Graph Forum* 27(1):102–113
5. Bouhekeur T, Schlick C (2005) Generic mesh refinement on GPU. In: *ACM SIGGRAPH/Eurographics Graphics Hardware*, pp 99–104
6. Carr NA, Hart CT (2002) Meshed atlases for real-time procedural solid texturing. *ACM Trans Graph* 21(2):106–131
7. Carr NA, Hoberock J, Crane K, Hart JC (2006) Rectangular multi-chart geometry images. In: *Symposium on geometry processing*, pp 181–190
8. Dachsbacher C, Vogelgsang C, Stamminger M (2003) Sequential point trees. In: *ACM SIGGRAPH 2003*, pp 657–662
9. Engelhardt T, Dachsbacher C (2008) Octahedron environment maps. In: *Proceedings of vision, modelling and visualization 2008*
10. Floater M et al (1997) Parametrization and smooth approximation of surface triangulations. *Comput Aided Geom Des* 14(3):231–250
11. Floater MS, Hormann K (2005) Surface parameterization: a tutorial and survey. In: *Dodgson NA, Floater MS, Sabin MA (eds) Advances in multiresolution for geometric modelling*. Springer Verlag, pp 157–186
12. Gotsman C, Gu X, Sheffer A (2003) Fundamentals of spherical parameterization for 3 D meshes. *ACM Trans Graph* 22(3):358

13. Greene N (1986) Environment mapping and other applications of world projections. *IEEE Comput Graph Appl* 6(11):21–29
14. Gu X, Gortler SJ, Hoppe H (2002) Geometry images. In: *Proceedings of ACM SIGGRAPH 2002*, pp 355–361
15. Gu X, Yau S (2003) Global conformal surface parameterization. In: *Proceedings of the 2003 Eurographics symposium on Geometry processing*, pp 127–137
16. Heidrich W, Seidel H (1998) View-independent environment maps. In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on graphics hardware*
17. Hernández B, Rudomín I (2006) Simple dynamic lod for geometry images. In: *Proc. of GRAPHITE 2006*, pp 157–163
18. Hernández B, Rudomin I (2006) Simple dynamic lod for geometry images. In: *Proceedings of the 4th international conference on computer graphics and interactive techniques in Australasia and Southeast Asia*. ACM New York, NY, USA, pp 157–163
19. Hu L, Sander PV, Hoppe H (2009) Parallel view-dependent refinement of progressive meshes. In: *Proceedings of the 2009 symposium on interactive 3D graphics and games*. ACM, New York, NY, USA, pp 169–176
20. Hu L, Sander PV, Hoppe H (2010) Parallel view-dependent level-of-detail control. *IEEE Trans Vis Comput Graph* 16(5):718–728
21. Ji J, Wu E, Li S, Liu X (2005) Dynamic lod on GPU. In: *CGI '05: proceedings of the computer graphics international 2005*. IEEE Computer Society, Washington, DC, USA, pp 108–114
22. Kalaiah A, Varshney A (2003) Modeling and rendering of points with local geometry. *IEEE Trans Vis Comput Graph* 9(1):30–42
23. Karlsson F, Ljungstedt CJ (2004) Ray tracing fully implemented on programmable graphics hardware. Master's thesis, Chalmers University of Technology
24. Kobbelt L, Vorsatz J, Labsik U, Seidel H (1999) A shrink wrapping approach to remeshing polygonal surfaces. *Comput Graph Forum* 18(3):119–130
25. Lefebvre S, Dachsbacher C (2007) Tiletrees. In: *Proceedings of the ACM SIGGRAPH symposium on interactive 3D graphics and games*. ACM Press
26. Lin J, Jin X, Fan Z, Wang C (2008) Automatic polycube-maps. In: *GMP 2008: advances in geometric modeling and processing: 5th international conference*. Springer, p 3
27. Losasso F, Hoppe H, Schaefer S, Warren J (2003) Smooth geometry images. In: *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*. Eurographics Association, pp 138–145
28. Luebke D, Erikson C (1997) View-dependent simplification of arbitrary polygonal environments. *ACM Press/Addison-Wesley Publishing Co*. New York, NY, USA
29. Luebke D, Erikson C (1997) View-dependent simplification of arbitrary polygonal environments. In: *Proceedings of SIGGRAPH 1997*, pp 199–208
30. Peyre G, Mallat S (2005) Surface compression with geometric bandelets. *ACM Trans Graph* 24(3):601–608
31. Praun E, Hoppe H (2003) Spherical parametrization and remeshing. *ACM Trans Graph* 22(3):340–349
32. Purnomo B, Cohen JD, Kumar S (2004) Seamless texture atlases. In: *Proc symp geom*, pp 67–76
33. Sander PV, Wood ZJ, Gortler SJ, Snyder J, Hoppe H (2003) Multi-chart geometry images. In: *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*. Eurographics Association, pp 146–155
34. Schröder P, Sweldens W (1995) Spherical wavelets: efficiently representing functions on the sphere. In: *Proceedings of the 22nd annual conference on computer graphics and interactive techniques*, pp 161–172
35. Segal M, Akeley K (2004) The OpenGL graphics system: a specification, version 2.0.
36. Shiue LJ, Jones I, Peters J (2005) A realtime GPU subdivision kernel. *ACM Trans Graph* 24(3):1010–1015
37. Smolic A, McCutchen D (2004) 3DAV exploration of video-based rendering technology in MPEG. *IEEE Trans Circuits Syst Video Technol* 14(3):348–356
38. Tarini M, Hormann K, Cignoni P, Montani C (2004) Polycube-maps. *ACM Trans Graph* 23(3):853–860
39. Tutte W (1963) How to draw a graph. *Proc Lond Math Soc* 3(1):743
40. Von Herzen B, Barr A (1987) Accurate triangulations of deformed, intersecting surfaces. In: *Proceedings of the 14th annual conference on computer graphics and interactive techniques*, pp 103–110
41. Wald I (2004) Realtime ray tracing and interactive global illumination. Ph.D. thesis, Computer Graphics Group, Saarland University

42. Wan L, Wong TT, Leung CS (2007) Isocube: exploiting the cubemap hardware. *IEEE Trans Vis Comput Graph* 13(4):720–731
43. Wang H, He Y, Li X, Gu X, Qin H (2007) Polycube splines. In: *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pp 241–251
44. Wang H, Jin M, He Y, Gu X, Qin H (2008) User-controllable polycube map for manifold spline construction. In: *Proceedings of the 2008 ACM symposium on solid and physical modeling*. ACM, pp 397–404
45. Xia J, Garcia I, He Y, Xin S, Patow G (2011) Editable polycube map for GPU-based subdivision surfaces. In: *Symposium on interactive 3D graphics and games*. ACM, pp 151–158



Bin Sheng received his BA degree in English and BE degree in Computer Science from Huazhong University of Science and Technology in 2004, MS degree in software engineering from the University of Macau in 2007, and PhD degree in Computer Science from The Chinese University of Hong Kong in 2011. He is currently an assistant professor in the Department of Computer Science and Engineering at Shanghai Jiao Tong University. His research interests include virtual reality, computer graphics, and image based techniques.



Weiliang Meng received the B.E. degree in Computer Science from Civil Aviation University of China in 2003, M.Sc. degree in Computer Application from Tianjing University in 2006, and PhD degree in Computer Application from State Key Laboratory of Computer Science at Institute of Software, Chinese Academy of Sciences. He is currently a postdoctor in National Laboratory of Pattern Recognition (NLPR) at Institute of Automation, Chinese Academy of Sciences. His research interests include geometry modeling, and image based modeling and rendering.



Hanqiu Sun received her MS degree in electrical engineering from University of British Columbia, and PhD degree in computer science from University of Alberta, Canada. She is an associate professor in Department of Computer Science and Engineering, Chinese University of Hong Kong (CUHK). Her current research interests include virtual and augmented reality, interactive graphics/animation, hypermedia, mobile image/video processing and navigation, tele-medicine, realistic haptics simulations.



Wen Wu received her Bsc, Msc degrees respectively from Beijing Univ. of Aeronautics and Astronautics, and the Institute of Computing Technology of Chinese Academy of Sciences. She received her PhD degree from The Chinese University of Hong Kong. She is currently an assistant professor in Department of Computer and Information Science at University of Macau and also an honorary research associate in the Department of Computer Science and Engineering at The Chinese University of Hong Kong. Her research interests include medical simulation, virtual reality and physically-based animation.



Enhua Wu received the BS degree from Tsinghua University in 1970, and the PhD degree from the University of Manchester (UK) in 1984. He is currently a research professor at the Institute of Software, Chinese Academy of Sciences. He has also been teaching at the University of Macau since 1997, where he is currently the associate dean of Faculty of Science and Technology. His research interests include realistic image synthesis, virtual reality, and scientific visualization.