

Adaptive Geometry Image

Chih-Yuan Yao and Tong-Yee Lee, Member, IEEE

Abstract—We present a novel postprocessing utility called adaptive geometry image (AGIM) for global parameterization techniques that can embed a 3D surface onto a *rectangular*¹ domain. This utility first converts a single rectangular parameterization into many different tessellations of square geometry images(GIMs) and then efficiently packs these GIMs into an image called AGIM. Therefore, undersampled regions of the input parameterization can be up-sampled accordingly until the local reconstruction error bound is met. The connectivity of AGIM can be quickly computed and dynamically changed at rendering time. AGIM does not have T-vertices, and therefore, no crack is generated between two neighboring GIMs at different tessellations. Experimental results show that AGIM can achieve significant PSNR gain over the input parameterization, AGIM retains the advantages of the original GIM and reduces the reconstruction error present in the original GIM technique. The AGIM is also suitable for global parameterization techniques based on quadrilateral complexes. Using the approximate sampling rates, the PolyCube-based quadrilateral complexes with AGIM can outperform state-of-the-art multichart GIM technique in terms of PSNR.

Index Terms—GIM, AGIM, T-vertices, zippering, tessellations.



1 INTRODUCTION

GEOMETRY image (GIM) [12], [26], [31] is a well-known technique for representing a regular mesh without storing its connectivity. GIMs are very suitable for hardware-assisted rendering, LOD generation and compression. Without a good 3D mesh data parameterization, the regular GIM sampling cannot evenly represent regular samplings on the surface. Undersampling can occur, thereby causing a high reconstruction error. This problem is connected to lack of area preservation of mapping. The multichart geometry image (MCGIM) [34] presents an atlas-based parameterization to overcome the distortion problem in GIMs [12]. MCGIM consists of many irregular charts and needs a zippering algorithm to determine the pixel neighbors across charts. The state-of-the-art MCGIM yields a significant PSNR (i.e., peak signal-to-noise ratio) gain over GIM and yields better geometric approximations than other semi-regular remeshing methods. Irregular-chart MCGIM solves the GIM limitation but creates other problems:

1. not suitable for dynamic LOD manipulation (i.e., zippering connectivity is fixed a priori in preprocessing),
 2. less efficiency in packing irregular charts,
 3. less efficient implementation for applications such as texture synthesis and mesh editing mentioned in [3], and
1. It is easy to reparameterize an irregular embedding onto a regular/or square embedding.

• The authors are with the Computer Graphics Group/Visual System Laboratory, Department of Computer Science and Information Engineering, National Cheng-Kung University, No. 1, Ta-Hsueh Road, Tainan 701, Taiwan, R.O.C. E-mail: ippo@csie.ncku.edu.tw, tonylee@ncku.edu.tw.

Manuscript received 29 Sept. 2007; revised 6 Dec. 2007; accepted 25 Feb. 2008; published online 28 Feb. 2008.

Recommended for acceptance by B. Guo.

For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2007-09-0149. Digital Object Identifier no. 10.1109/TVCG.2008.39.

4. high-valence vertices likely generated by their zipper algorithm due to irregularity of charts. The high-valence vertices can result in visual artifacts.

A good zippering algorithm should not create high-valence vertices and, also, can quickly compute connectivity at rendering time.

In this paper, we present a novel postprocessing utility for global parameterization techniques, which can embed a 3D surface onto a rectangular domain such as GIMs and spherical parameterization [11], [31]. This novel technique first converts a rectangular parameterization into multiple GIMs with different resolutions and then efficiently packs these GIMs into an image called Adaptive Geometry Image (AGIM). The resolution of each GIM is adaptively determined to ensure that the reconstruction error is within a given bound. The AGIM retains the advantages of the original GIMs and reduces the reconstruction error present in GIMs. Experimental results show that AGIM can significantly benefit global parameterizations such as [12] and [31] with higher PSNR gain. The AGIM can directly compute approximation errors for different resolutions, which allows it to directly pick approximation errors and zipper GIMs. This ability has a strong advantage of benefiting applications such as view dependent LOD. Unlike MCGIMs, the AGIM does not create irregular charts, so there are several advantages over MCGIMs. First, the proposed zippering algorithm to stitch square GIMs tessellated at different rates can be quickly computed and dynamically changed at rendering time. In addition, the AGIM does not need to store the pixel connectivity across square GIMs, and it only requires very small amounts of adjacent GIM information. Second, the packing efficiency of AGIM can be very high. Third, for the AGIM representation, high-valence vertices for removing T-vertices are rare in our practical scenarios. In terms of PSNR, the AGIM, starting from a regular GIM, generally cannot outperform MCGIM. The AGIM is suitable for global parameterization techniques based on quadrilateral complexes. Using the approximate sampling rates, the PolyCube-based [37] quadrilateral complexes with AGIM can outperform state-of-the-art MCGIM technique in terms of PSNR. Finally, we will show

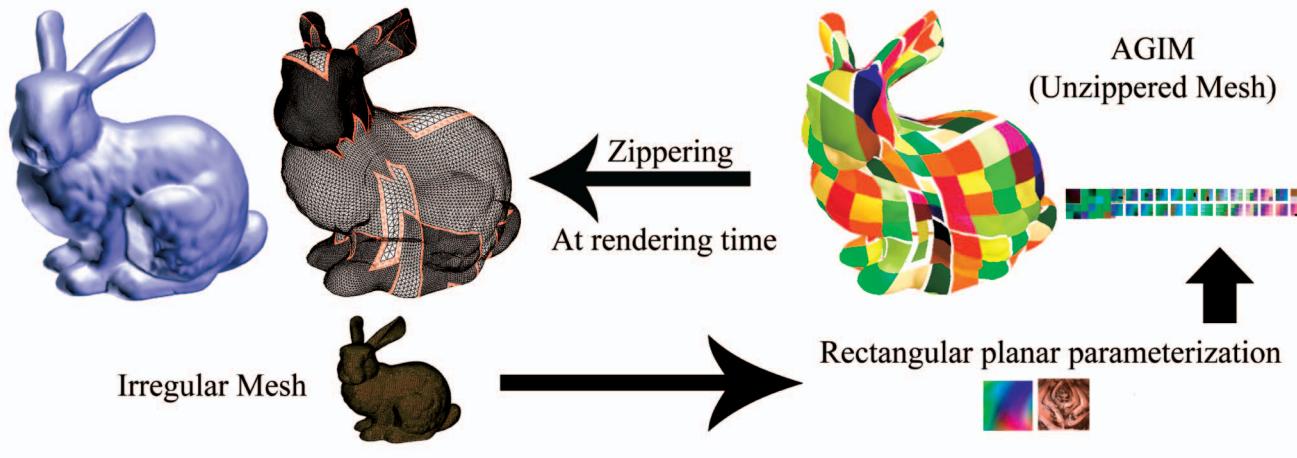


Fig. 1. Overview of AGIM.

that the AGIM can be partially accelerated by the GPU to achieve real-time rendering performance.

2 RELATED WORK

There have been several schemes such as [8] and [21] proposed for converting irregular meshes into semiregular remeshes. GIM [12] is a completely regular structure for representing an arbitrary surface. The major limitation of GIMs is its distortion problem. To parameterize an entire mesh into a single regular chart easily leads to greater distortion and less uniform surface sampling than can be achieved with irregular multicharts (MCGIM) [34]. The MCGIM achieves very high PSNR reconstruction quality but still has some problems, as mentioned in Section 1. Recently, the rectangular MCGIM [3] (RMCGIM) was used to exploit rectangular patches onto tile surfaces, guaranteeing a one-to-one pixel correspondence across chart boundaries. However, it is difficult to guarantee each tiled patch with a rectangular shape. The RMCGIM still has higher distortion than MCGIM, and its packing efficiency is not too high (about 80 percent). In contrast to multichart methods, several single chart methods such as spherical parameterization [31] and smooth GIMs [26] can reduce undersampling or distortion problem in GIMs. Although the spherical parameterization [31] only supports genus-0 meshes, this drawback is overcome by toroidal domain tessellation [35]. In addition, Peyré and Mallat [30] attempted to improve the compression ratio of GIMs using wavelet techniques. Hernández and Rudomin [14] and Ji et al. [17] proposed dynamic LOD of GIMs on GPU. To remove T-vertices, Ji et al. used a restricted quadtree triangulation (RQT) method [15]. With RQT, the levels of adjacent quadtree nodes differ by, at most, one.

Like GIMs, the terrain data is always represented with a regular grid structure. Recently, Pajarola and Gobbetti [29] present a very thorough survey on semiregular multi-resolution representation for interactive terrain rendering. The adaptive terrain rendering algorithm can dynamically generate adaptive meshes. The T-vertices may appear when two neighboring meshes are tessellated at different rates. A mesh with T-vertices easily creates cracks. Many techniques [4], [20], [25] have been proposed to optimize adaptive terrain mesh without generating T-vertices. However, many high-valence vertices are always generated to remove T-vertices. To avoid these types of vertices, most previous

methods require that the levels of adjacent mesh tessellations differ by at most one or two [29]. Alternatively, Lossasso et al. [7] do not remove T-vertices and propose to use Fragment shader to blend rendering artifacts caused by T-vertices. This approach can work well in terrain rendering. However, it is not suitable for geometry processing such as mesh editing and deformation. The quaddominant mesh [6], [13], [36], [38] is also a very popular model representation with a regular data structure. The same problem will be encountered as the adaptive mechanism is applied to quadsurface [1], [27], [33]. Like MCGIMs, a zippering algorithm can be used to remove T-vertices.

Surface parameterization research has been very active 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 [10] or refer to the literature. In our setting, the parameterization results from previous approaches can be viewed as inputs to the proposed AGIM method. Our method converts their results into a single AGIM image with lower geometric reconstruction error and more uniform sampling than that which can be achieved with a GIM.

3 APPROACH OVERVIEW AND DEFINITIONS

In Fig. 1, given a surface mesh M and its surface parameterization Φ , our goal is to convert Φ into an AGIM. To achieve this goal, we borrow parameterization results in [31] as the input to the proposed method. The parameterization data from other techniques globally parameterizing a 3D surface onto a rectangular domain can also be used. From Φ , we initially create a low resolution $(2^i + 1) \times (2^i + 1)$ GIM denoted as G_i at tessellation rate i . In our setting, this G_i is treated as a virtual GIM, and each $grid(x, y)$ is viewed as a control vertex. In Fig. 2, each control vertex is illustrated by a black dot. The surface parameterization Φ is also displayed for easy explanation of our idea. For each $grid(x, y)$, our method adaptively determines a local $(2^{i'} + 1) \times (2^{i'} + 1)$ GIM centralized at (x, y) , denoted as $g_{i',x,y}$. Therefore, the neighbor of $g_{i',x,y}$ can be tessellated at different rates, thus creating T-vertices. Section 4 will present more details to generate the AGIM. The zippering algorithm will be introduced to remove T-vertices at rendering time in Section 5.

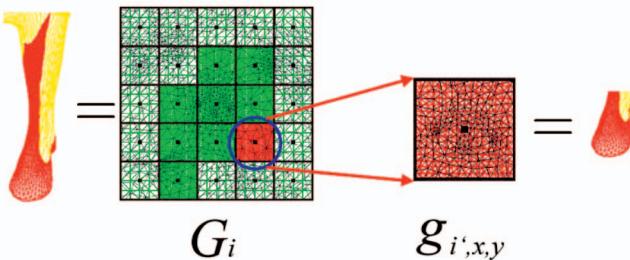


Fig. 2. Control vertex and its corresponding GIM obtained using regular surface parameterization sampling Φ .

4 GENERATING AN ADAPTIVE GEOMETRY IMAGE

An initial G_i at a small tessellation rate i is chosen to reduce the overall number of samplings on the AGIM. We also initially assign a 5×5 (i.e., $i' = 2$) local $g_{i',x,y}$ for each control vertex of G_i . We then measure the reconstructed surface error between the reconstructed surface (i.e., $g_{i',x,y}$) and original mesh using (1) with Metro [5]:

$$d(P, S) = \min_{p' \in S} \|P - p'\|_2, \quad (1)$$

where the vertex P belongs to the remeshed model, and S is the original model. If the error is beyond a threshold, this initial $5 \times 5 g_{i',x,y}$ will be replaced using a new $g_{i',x,y}$ with $i' > 2$. To determine an appropriate i' , we simply start testing from $i' = 3$ incrementally until the measured error is below the threshold. Fig. 3a shows a visualization of G_i consisting of many local $g_{i',x,y}$ tessellated at different rates. These square GIMs can be packed efficiently into an image called AGIM, as shown in Fig. 3b. Let k be the maximum i' among all $g_{i',x,y}$. In Fig. 4b, a container is designed as a $(2^k + 2) \times (2^k + 2)$ image, and an AGIM can consist of many containers in Fig. 4d. In Fig. 4c, several $g_{i',x,y}$ with $2 \leq i' \leq k$ are packed together into each container. The upper bound on the number of containers can be computed:

$$\left(\frac{\sqrt{\sum_{i'=2}^k n_{i'} \times (2^{i'} + 1)^2}}{2^k + 2} \right)^2, \quad (2)$$

where $n_{i'}$ is the number of GIMs at tessellation rate i' . The pseudocode of the packing algorithm is shown below. Fig. 4 illustrates the packing algorithm. Let us denote $queue_{(i')}$ as a queue that temporarily stores all GIMs at tessellation rate i' .

Using the pseudocode below, `Fill_in.Container`($\text{Image_block}_{(i')}$) will return true if $\text{Image_block}_{(i')}$ can be inserted into the current container. If this block cannot be inserted, we will try smaller blocks until no block can be added into the current container. Next, we start to insert blocks into a new container if there are still some blocks left to be added. Our packing heuristic is a greedy method for packing all blocks into a container. Whenever we can insert blocks into the container, we always pack the large blocks first and, then, the small ones. Once the number of square containers is known, it is easy to find a rectangular AGIM to fill up all containers with the minimal wasted space [2], [32]. Alternatively, we can pack image blocks into an AGIM directly to reduce the final size of an AGIM. To facilitate I/O of AGIM between file disks and memory, we decided to pack varying-size image blocks into containers with a fixed size and then pack these containers in the AGIM. Later, experimental

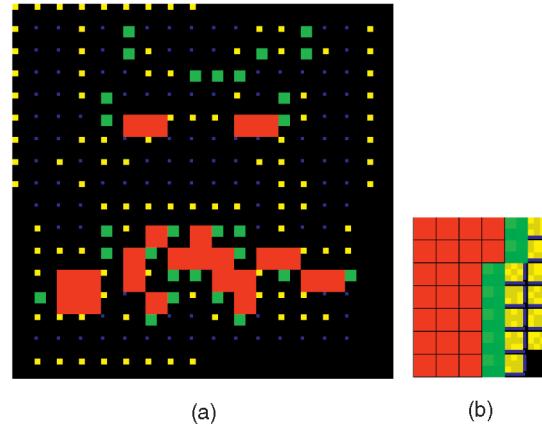


Fig. 3. (a) Visualization of G_i . (b) All square $g_{i',x,y}$ GIMs are packed into an image called AGIM. In (a), different colors represent different sizes of $g_{i',x,y}$.

results show that this arrangement still can have a high packing efficiency.

```

allocate a new container; /* AGIM initialization */
while(some blocks left in queues){
    i' = k; /* k is the maximum i' among  $g_{i',x,y}$  */
    while(i' ≥ 2){
        if((Image_block(i') = queue(i').pop())! = NULL){
            if(!Fill_in_Container(Image_block(i'))){
                queue(i').push(Image_block(i'));
            }
            if(i' == 2){
                allocate a new container;
                /* there is not enough space */
                i' = k;
            }else{
                i'; /* try smaller ones */
            }
        }else{
            i'; /* try smaller ones */
        }
    }
}

```

5 ZIPPERING ALGORITHM

5.1 Algorithm Overview

Fig. 5 shows an overview of the proposed zippering algorithm. In this figure, given any two neighbors $g_{i'_1, x_1, y_1}$ and $g_{i'_2, x_2, y_2}$ with $i'_1 < i'_2$, some T-vertices will be created along their boundary. To remove these T-vertices, we first remove several 2×2 image blocks of $g_{i'_1, x_1, y_1}$ along the boundary. These 2×2 blocks are adjacent to $g_{i'_2, x_2, y_2}$. After this removal, there are many gaps created in this figure. These removed image blocks, called *zipper blocks*, are further classified into two categories: 1) *edge block* and 2) *corner block*.

Definition. A 2×2 zipper block is an edge block if only one of its four adjacent blocks belongs to the other GIM at different tessellation rates. Otherwise, it is a corner block.

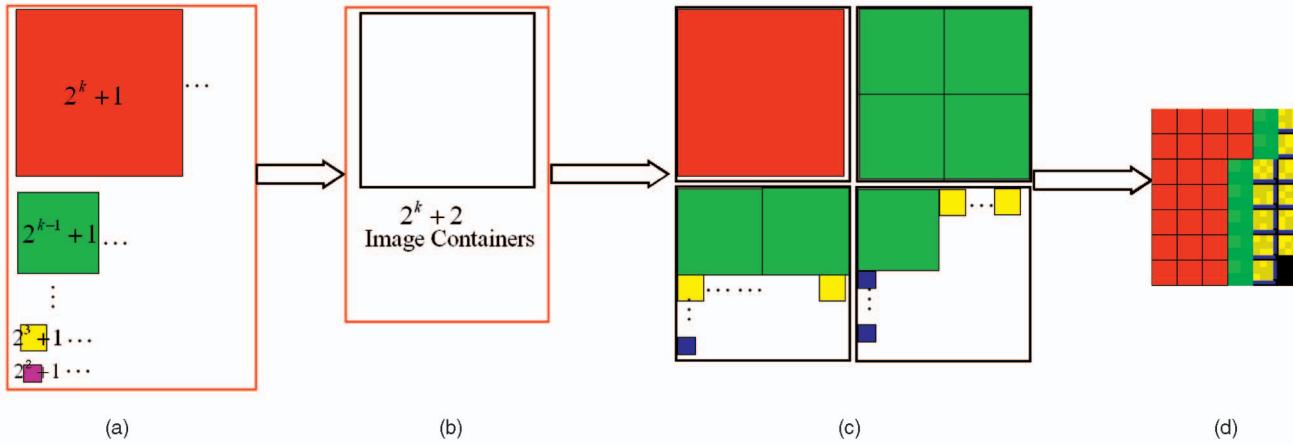


Fig. 4. Illustration of AGIM packing algorithm. (a) $queue_{(i')}$. (b) An image container. (c) AGIM consists of many containers. (d) AGIM.

Our zippering algorithm consists of three major steps (Fig. 5): 1) generating triangulation vertices on each zipper block, 2) triangulating all edge and corner blocks, independently, and 3) stitching each pair of adjacent triangulations.

5.2 Generating Triangulation Vertices

5.2.1 Up-Sampling a Zipper Block

Once each 2×2 edge block of g_{i_1, x_1, y_1} is detected, we first sample it up to the same tessellation rate with its adjacent block within g_{i_2, x_2, y_2} . After up-sampling each edge block, we build a treelike triangulation Γ within it to stitch g_{i_1, x_1, y_1} and g_{i_2, x_2, y_2} (Section 5.2.2). To smooth the zippering triangulation, we achieve this up-sampling using [3]. A corner block must have two adjacent blocks tessellated at k_1 and k_2 times with $k_1 \geq k_2$. There will be two treelike triangulations called Γ_{k_1} and Γ_{k_2} (Section 5.2.3) built within each corner block. First, we tessellate it at k_1 times to build a Γ_{k_1} . To build a Γ_{k_2} , it is not necessary to tessellate it again, since all $(2^{k_2} + 1) \times (2^{k_2} + 1)$ samples $\subseteq (2^{k_1} + 1) \times (2^{k_1} + 1)$ samples.

5.2.2 Generating Triangulation Vertices in an Edge Block

Not all samples are used to build a triangulation within an edge block. A good Γ should be a triangulation reminiscent of a balanced tree. After up-sampling, each edge block is tessellated into a $(2^k + 1) \times (2^k + 1)$ block. See several examples in Fig. 6. Its border adjacent to g_{i_2, x_2, y_2} is called the *bottom border*, and the opposite border is called the *top border*. We define a local frame to easily address each sample at grid(x, y) by orienting each block (Fig. 6) such that its *bottom border* is aligned with the positive x -axis pointing right and the positive y -axis pointing up (from the *bottom border* to the *top border*). The origin $(0, 0)$ is set at its left-bottom corner. Each vertex of Γ is denoted as $L_{(x,y)}^{(\ell,i)}$, where $i(i \geq 1)$ indicates the i th vertex on the ℓ th ($\ell \geq 0$) level of a treelike Γ , and this vertex is selected from the sample at grid (x, y) of this block. There are $(2^k + 1)$ vertices on the 0th level of Γ , and these vertices are selected from $(2^k + 1)$ samples on the *bottom border*. The number, N_ℓ , of vertices on the ℓ th ($0 \leq \ell \leq k$) level of

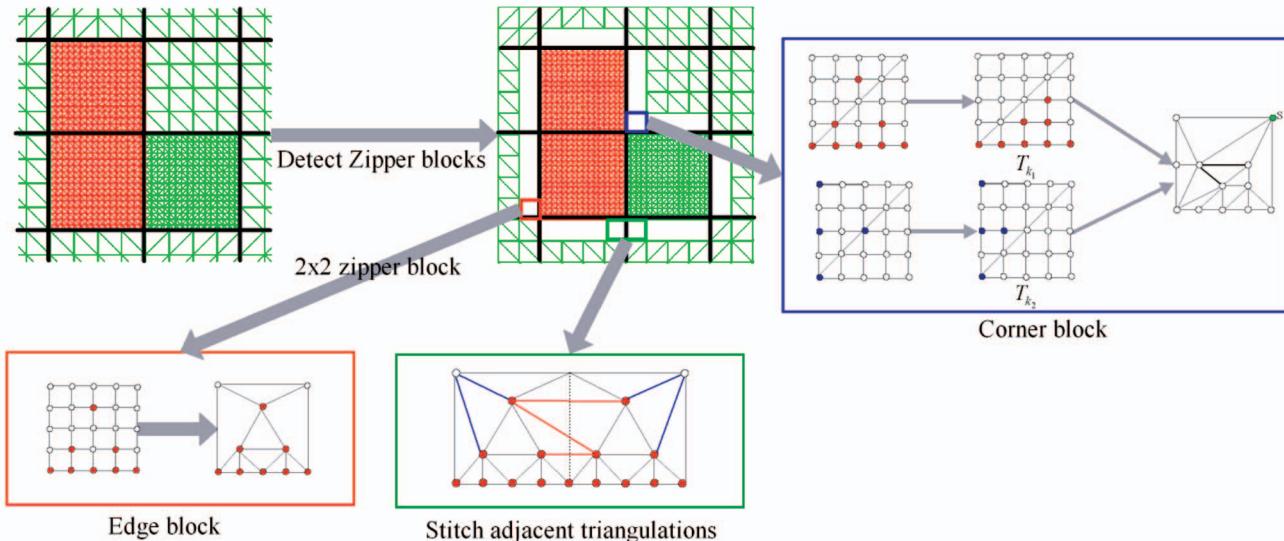


Fig. 5. Overview of our zippering algorithm.

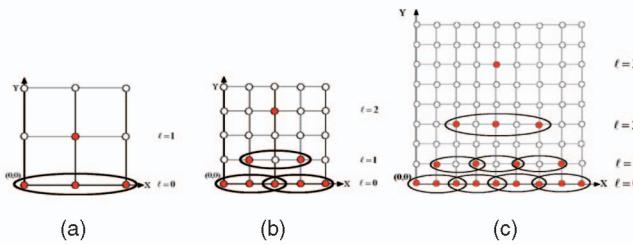


Fig. 6. The grid samples selected to build a Γ are marked in red. (a) $k = 1$. (b) $k = 2$. (c) $k = 3$.

Γ can be determined by $N_0 = 2^k + 1, N_1 = (2^k + 1 - 1)/2 = 2^{k-1}, N_2 = 2^{k-1} - 1, N_3 = (2^k - 1 - 1)/2 = 2^{k-2} - 1, \dots, N_{\ell=k} = (2^{k-(k-2)} - 1 - 1)/2 = 1$, respectively. The number of vertices is *odd* on each level of Γ excluding the first level when $k \geq 2$. Therefore, we need to handle this special case using the following three simple rules to determine the remaining vertices $\{L_{(x,y)}^{(\ell,i)} | \ell > 0\}$ of Γ :

Input. A $(2^k + 1) \times (2^k + 1)$ edge block and the i th vertex $L_{(x,y)}^{(\ell=0,i)}$ on the 0th level of Γ is located at the $(L_x^{(0,i)}, L_y^{(0,i)})$ grid of this block, where $0 \leq x \leq 2^k$ and $y = 0$. Incrementally, from $\ell = 0$ to $(k-1)$.

1. If $(N_\ell \in \text{odd})$, then $L_x^{(\ell+1,i)} = L_x^{(\ell,2*i)}, \forall i \in [1, (N_\ell - 1)/2]$.
2. If $(N_\ell \in \text{even})$, then $L_x^{(\ell+1,i)} = (L_x^{(\ell,i)} + L_x^{(\ell,i+1)})/2, \forall i \in [1, (N_\ell - 1)]$.
3. $L_y^{(\ell+1,i)} = L_y^{(\ell,i)} + \ell$.

According to the above rules, if $N_\ell \in \text{odd}$, each vertex on the $(\ell+1)$ th level corresponds to three vertices (1-to-3) on the ℓ th level, as circled in Fig. 6. If $N_\ell \in \text{even}$, the correspondence becomes 1-to-2. The goal of the rule 3 is to keep Γ spread over this $(2^k + 1) \times (2^k + 1)$ as much as possible.

5.2.3 Generating Triangulation Vertices in a Corner Block
Within a corner block, we build two triangulations Γ_{k_1} and Γ_{k_2} with $k_1 \geq k_2$. In Fig. 8a, Γ_{k_1} and Γ_{k_2} is located in its lower (i.e., red) and upper (i.e., blue) triangular regions. All possible configurations of k_1 and k_2 are shown in Fig. 8b, and they can be oriented to the standard configuration in Fig. 8a. Both Γ_{k_1} and Γ_{k_2} are built independently (Fig. 7) and separated by the main diagonal. All vertices of Γ_{k_1} cannot appear in the blue region (i.e., Γ_{k_2}) of this block, and vice versa. We find their vertices in three steps (1, 2, and 3). In Section 5.2.1, we tessellate this corner block into a $(2^{k_1} + 1) \times (2^{k_1} + 1)$ block, and then, we select Γ_{k_1} 's vertices (i.e., red samples) using the same rules for handling the *edge block*. Excluding a $L_{(0,0)}^{(0,1)}$ vertex, its vertices can be neither on the main diagonal nor on the upper triangular region of this block. Therefore, we need to give up some vertices violating these two constraints and reselect some new vertices using the following rules in steps 1, 2, and 3, respectively:

- 1. $L_x^{(1,1)} + 1$.
- 2.1. If $(N_\ell \in \text{odd})$, then

$$L_x^{(\ell+1,i)} = L_x^{(\ell,2*i+1)}, \forall i \in [1, (N_\ell - 1)/2].$$

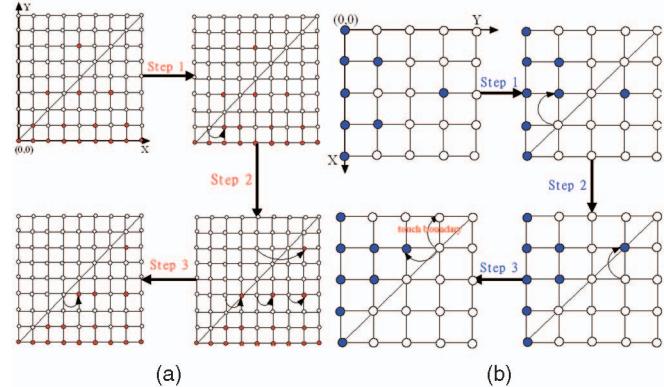


Fig. 7. Three major steps to generate triangulation vertices in a *corner block*. (a) Selected vertices of $\Gamma_{k_1=3}$. (b) Selected vertices of $\Gamma_{k_2=2}$.

- 2.2. If $(N_\ell \in \text{even})$, then

$$L_x^{(\ell+1,i)} = L_x^{(\ell,(i+1))}, \forall i \in [1, (N_\ell - 1)].$$

- 3.1. If (selected vertex is on the main diagonal) $L_x^{(\ell,i)} = L_x^{(\ell,i)} + 1, \forall i > 1$.
- 3.2. If (selected vertex touches the boundary of the block) $L_y^{(\ell,i)} = L_y^{(\ell,i)} - 1, \forall i > 1$.

In a similar manner, the vertices of Γ_{k_2} can be found in the upper triangular region of this block, as shown in Fig. 7b. Either $k_1 = 1$ or $k_2 = 1$ is a special case. For example, when $k_1 = 1$, a vertex $L_{(1,1)}^{(1,1)}$ of Γ_{k_1} is selected on the main diagonal of this 3×3 block in the step 1. We need to reselect a new one for $L_{(1,1)}^{(1,1)}$. According to the above rules, we cannot find a qualified one for it from the integer grids. For simplicity, we get this $L_{(1,1)}^{(1,1)}$ by subsampling at $(1, 0.5)$ location of this block. Similarly, when $k_2 = 1$, we get $L_{(1,1)}^{(1,1)}$ of Γ_{k_2} at $(1, 0.5)$ location.

5.3 Triangulating Zipper Blocks

In the following, let $L_{\Gamma}^{(\ell,i)}$ and $L_{k_i}^{(\ell,i)}$ denote that it is the i th vertex on the ℓ th level of Γ and Γ_{k_i} , respectively. Both s

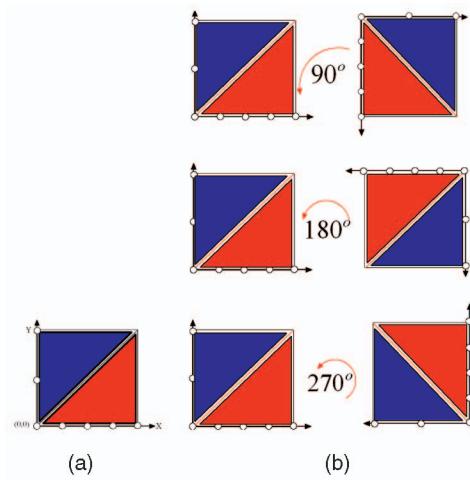


Fig. 8. Two triangulations Γ_{k_1} and Γ_{k_2} with $k_1 \geq k_2$ are built in red and blue areas.

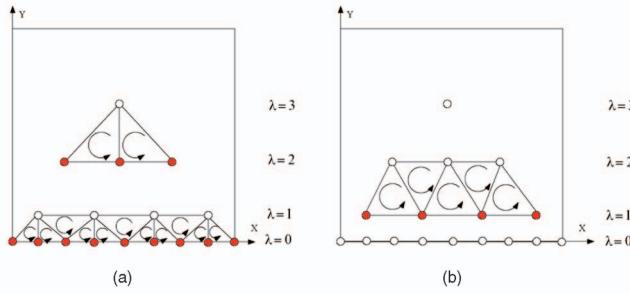


Fig. 9. Triangle strips between levels ℓ and $\ell+1$. Arrow heads indicate triangle orientations in generated triangle strips. (a) $N_\ell \in \text{odd}$. (b) $N_\ell \in \text{even}$.

and g denote samples located on the upper right and upper left corners of a zipper block.

5.3.1 Triangulating an Edge Block

We have obtained all Γ triangulation vertices in Section 5.2.2. If $N_\ell \in \text{odd}$, each vertex on the $(\ell+1)$ th level corresponds to three vertices on the ℓ th level. We can use either a set of triangle strips or fans for generating triangulation between these two levels (Fig. 9a). For fast rendering, we adopt [9] to build a triangle strip with repeated vertices between these two levels in the following pseudocode. Although repeated vertices result in degenerated triangles (i.e., zero area), this approach generates the same rendering appearance as that by either a set of triangle strips or fans.

```
Begin(triangle strip){
   $L_\Gamma^{(\ell, N_\ell-1)}, L_\Gamma^{(\ell, N_\ell)}, L_\Gamma^{(\ell, N_\ell-1)}, L_\Gamma^{(\ell+1, (N_\ell-1)/2)};$ 
  /*  $L_\Gamma^{(\ell, N_\ell-1)}$  is a repeated vertex */
  if( $N_\ell == 3$ )  $L_\Gamma^{(\ell, N_\ell-2)}$ 
  for( $i = (N_\ell - 2); i > 1; i = i - 2$ ) {
     $L_\Gamma^{(\ell, i)}, L_\Gamma^{(\ell+1, (i-1)/2)}, L_\Gamma^{(\ell, i-1)};$ 
    if( $(i - 2) > 1$ )
       $L_\Gamma^{(\ell+1, (i-1)/2)};$ 
    /*  $L_\Gamma^{(\ell+1, (i-1)/2)}$  is a repeated vertex*/
  }
  else
     $L_\Gamma^{(\ell, i-2)};$ 
}
}end(triangle strip);
```

If $N_\ell \in \text{even}$, each vertex on the $(\ell+1)$ th level corresponds to two vertices on the ℓ th level. The following is the pseudocode to build a triangle strip between these two levels (Fig. 9b):

```
Begin(triangle strip){
  for( $i = N_\ell; i > 0; i --$ ) {
     $L_\Gamma^{(\ell, i)};$ 
    if( $i > 1$ )  $L_\Gamma^{(\ell+1, i-1)};$ 
  }
}end(triangle strip);
```

Finally, an extra triangle will be formed by $(L_{(x,y)}^{(\ell=k,1)}, s, g)$ to stitch an adjacent 2×2 block on the top border. Fig. 10 shows several examples of triangulating edge blocks with $1 \leq k \leq 3$.

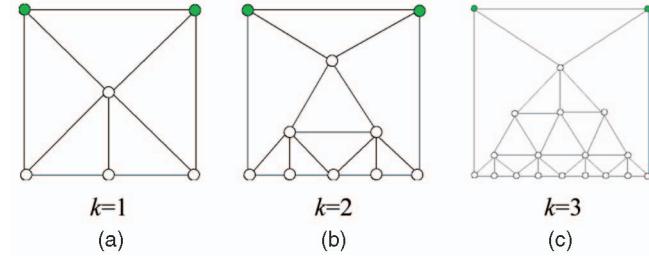


Fig. 10. Triangulations of edge blocks with $1 \leq k \leq 3$. Both samples s and g are illustrated as green points. (a) $k = 1$. (b) $k = 2$. (c) $k = 3$.

5.3.2 Triangulating a Corner Block

We have obtained all vertices of triangulations Γ_{k_1} and Γ_{k_2} in Section 5.2.3. In Fig. 11a, we build Γ_{k_1} and Γ_{k_2} independently using the same method for triangulating an edge block. Then, we stitch Γ_{k_1} and Γ_{k_2} within a corner block in two steps. First, we build two red triangles (Fig. 11b) using two vertex-tuples $(L_{k_1}^{(0,1)}, L_{k_1}^{(1,1)}, L_{k_2}^{(1,2^{k_2}-1)})$ and $(L_{k_1}^{(k_1,1)}, s, L_{k_2}^{(k_2,1)})$. Second, we triangulate the blue area (Fig. 11b) using one of the following four possible configurations, as illustrated in Fig. 12:

1. $k_1 = k_2 = 1$. This is a special case, since $L_{k_1}^{(1,1)} = L_{k_2}^{(1,1)}$. In this case, we build triangle strips by traveling vertices in the following order: $g, L_{k_2}^{(1,1)}, s, L_{k_1}^{(0,3)}$.
2. $k_1 = k_2 + 1$. The blue area becomes a trapezoidlike shape. Some vertices of Γ_{k_1} are on its lower base including $L_{k_1}^{(\ell,1)}$ with $1 \leq \ell \leq k_1$. Similarly, some vertices of Γ_{k_2} are on its upper base, including $L_{k_2}^{(\ell, N_\ell)}$ with $1 \leq \ell \leq k_2$, where N_ℓ is the number of vertices on the ℓ th level of Γ_{k_2} . Since $k_1 = k_2 + 1$, we build triangle strips to triangulate this trapezoid by traveling vertices in the following order (i.e., similar to that in Fig. 9b and please refer to its pseudocode): $L_{k_1}^{(k_1,1)} L_{k_2}^{(k_2, N_{k_2})} L_{k_1}^{(k_1-1,1)} L_{k_2}^{(k_2-1, N_{k_2})}, \dots, L_{k_2}^{(1, N_{k_2})} L_{k_1}^{(1,1)}$.
3. $k_1 = k_2 \neq 1$. The blue area becomes a rectanglelike shape. Since $k_1 = k_2 \neq 1$, we build triangle strips to triangulate this rectangle by traveling vertices in the following order:

$$L_{k_1}^{(k_1,1)} L_{k_2}^{(k_2, N_{k_2})}, L_{k_1}^{(k_1-1,1)} L_{k_2}^{(k_2-1, N_{k_2})}, \dots, L_{k_1}^{(1,1)}, L_{k_2}^{(1, N_{k_2})}.$$

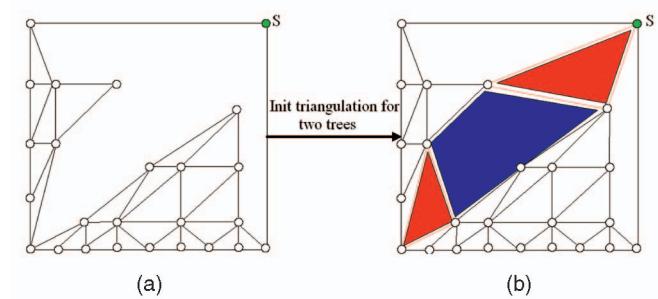


Fig. 11. (a) Build Γ_{k_1} and Γ_{k_2} independently. (b) Generate triangles to stitch Γ_{k_1} and Γ_{k_2} .

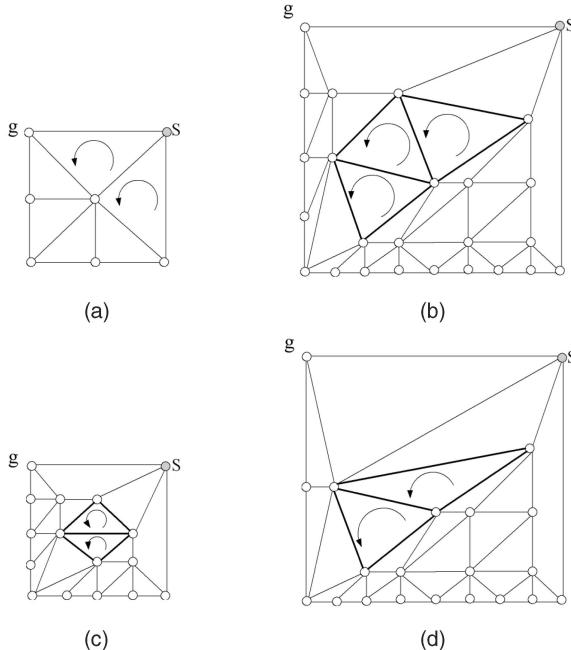


Fig. 12. Configurations of the inner triangulation in a *corner block*. (a) $k_1 = k_2 = 1$. (b) $k_1 = 3, k_2 = 2$. (c) $k_1 = k_2 = 2$. (d) $k_1 = 3, k_2 = 1$.

4. $k_1 - k_2 > 1$. Similar to case 2, we first create triangle strips by traveling vertices in the following order $L_{k_1}^{(k_2+1,1)} L_{k_2}^{(k_2,N_{k_2})} L_{k_1}^{(k_2,1)} L_{k_2}^{(k_2-1,N_{k_2})}, \dots, L_{k_2}^{(1,N_{k_2})} L_{k_1}^{(1,1)}$. Since $k_1 - k_2 > 1$, we next create a fan of triangles by traveling vertices in the following order: $L_{k_2}^{(k_2,N_{k_2})}, L_{k_1}^{(k_2+1,1)}, L_{k_1}^{(k_2+2,1)}, \dots, L_{k_1}^{(k_1,1)}$. The origin of this fan is $L_{k_2}^{(k_2,N_{k_2})}$. For case 4, theoretically, the vertex degree of $L_{k_2}^{(k_2,N_{k_2})}$ can be unbounded. However, from our practice, most of this case is with $k_1 = k_2 + 2$, in particular, $k_1 = 3$ and $k_2 = 1$. Therefore, we usually do not create high-valence vertices.

Fig. 12 shows the most frequent configurations of the above four cases. This table can be treated as a lookup table for triangulation. Any configuration with $k_1, k_2 \geq 4$ seems not practical, since it will potentially generate more samples than the original GIM [12]. To prevent this, we can simply enlarge the resolution of G_i . Therefore, for case 4, a high-valence fan at $L_{k_2}^{(k_2,N_{k_2})}$ can be avoided.

5.4 Stitching Adjacent Zipper Blocks

After all zipper blocks are triangulated independently, we need to stitch all adjacent zipper blocks. In terms of their block-to-block adjacency, there are three possibilities: 1) E-to-E, 2) E-to-C (or C-to-E), and 3) E-to-NZ (or NZ-to-E), as shown in Fig. 5, where E denotes a *edge block*, C denotes a *corner block*, and NZ denotes a 2×2 nonzipper-block, respectively. Figs. 13a and 13b show two examples of E-to-E and C-to-E cases. We stitch these two cases using the method for triangulating a corner block. In Fig. 13a, there

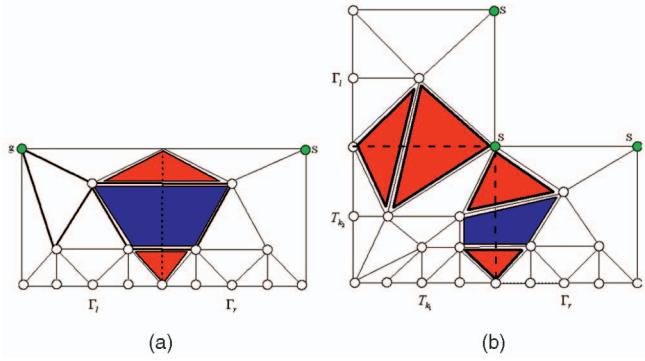


Fig. 13. Stitching adjacent zipper blocks. (a) E-to-E case and (b) C-to-E case.

are two triangulations called Γ_l and Γ_r within the left and right blocks. First, we need to add two edges $(L_l^{(1,N_1)}, L_r^{(1,1)})$ and $(L_l^{(l,1)}, L_r^{(r,1)})$ into $\Gamma_l \cup \Gamma_r$, thereby generating two triangles in the red area. Similar to triangulate the blue area in Fig. 11, we triangulate the blue area in Fig. 13a using Fig. 12. In Fig. 13b, on the left side, it is a *corner block*, and therefore, it has two triangulations Γ_{k_1} and Γ_{k_2} within this block. Then, Fig. 13b can be handled like Fig. 13a. In Fig. 13a, let us assume Γ_l has an NZ neighbor, i.e., a 2×2 block on its left side. We can triangulate this NZ-to-E case by creating a fan of triangles by traveling vertices in the following order: $g, L_l^{(l,1)}, L_l^{(l-1,1)}, \dots, L_l^{(1,1)}$. The origin of this fan is the sample at g . In a symmetric manner, we can handle an E-to-NZ case, too. According to the definition of a *corner block*, both cases C-to-C and C-to-NZ (or NZ-to-C) will not occur. Furthermore, for either C-to-E or E-to-C case, $\Gamma_l = \Gamma_r$. However, for an E-to-E case, l and r can be different.

6 RESULTS AND DISCUSSION

6.1 Results

In Table 1, we experimentally compare distortion rate (PSNR) from geometric reconstruction by AGIM and spherical parameterization [31] and GIM [12]. The input parameterization to our AGIM is in [31]. To fairly conduct this comparison, our initial virtual GIMs, G_i are all tessellated at 17×17 for all cases in this table. Remarkably, the number of samples for AGIM is close to or lower than other methods, but we efficiently reduce reconstruction error and therefore get higher PSNR than others. Fig. 18 shows AGIMs, original model and reconstructed model for the gargoyle, horse, and dinosaur examples. Fig. 14 shows a close look to reconstructed models comparing AGIM and [31]. Results show that AGIM can adaptively adjust GIM such that reconstructed models can better approximate the original models than [31]. From the above results, we can see our AGIM can benefit other parameterization methods such as [31]. The PSNRs of some results (horse and gargoyle in Table 1) are very close to MCGIM [34]. For achieving better PSNR, we must sacrifice the data storage, even if it is bigger than GIM.

As mentioned earlier, the MCGIM [34] can achieve very low distortion for parameterizing models. It is not very

TABLE 1
Reconstruction Error (PSNR) and the Number of Samples Using Different Reconstruction Methods

PSNR	Geometry Images [12]	GIM from Praun et al. [31]	AGIM from [31]	PolyCube [37]	AGIM from [37]
horse	74.3(66,049)	76.9(66,049)	84.9(46,240)	82.22(137,218)	87.3(54,510)
bunny	78.2(66,049)	79.8(66,049)	85.3(36,992)	74.53(43,010)	86.7(41,078)
venus	80.7(66,049)	83.4(66,049)	83.7(63,580)	85.25(73,730)	86.7(37,345)
dinosaur	74.8(66,049)	73.6(66,049)	86.9(56,644)	89.31(116,738)	89.78(49,539)
gargoyle	77.0(66,049)	79.2(66,049)	86.9(70,516)	75.93(92,162)	88.3(56,951)

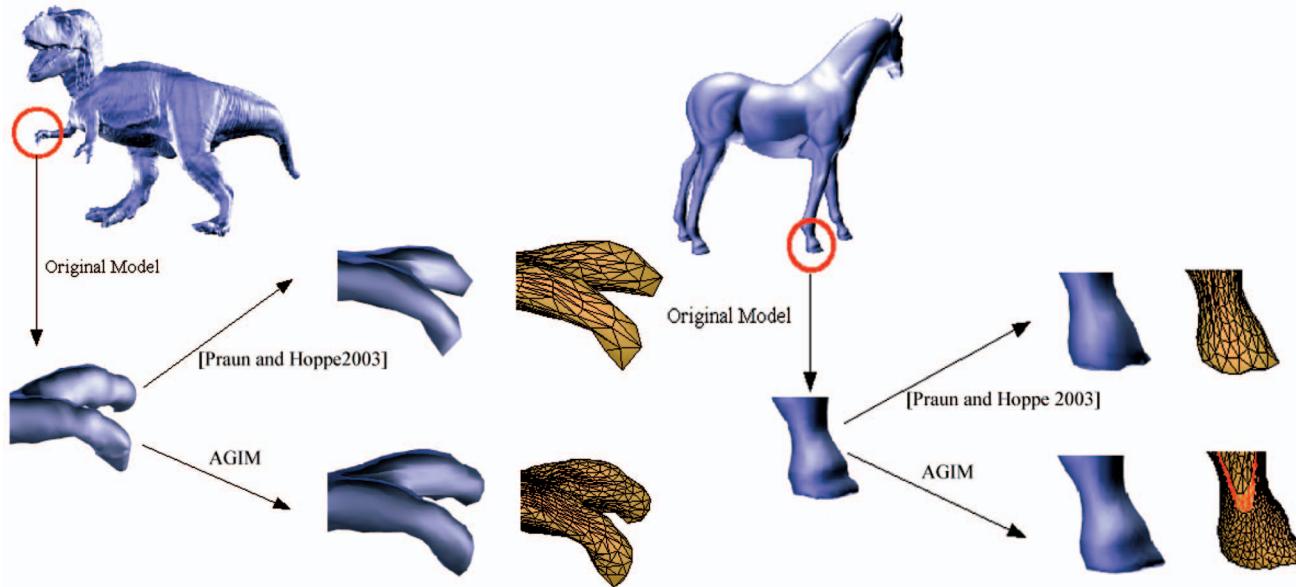


Fig. 14. A close look at reconstructed models comparing AGIM and Spherical Parameterization and Remeshing [31].

surprising to expect that the MCGIM can achieve better results than AGIM in this respect. Even if we increase the sampling rates of AGIMs to those of MCGIM, our resampled regular AGIMs have lower PSNR than MCGIMs or at most have comparable PSNR to that of MCGIMs. The gain of PSNR is bounded by the quality of input parameterization. However, the AGIM still shows some potential advantages over MCGIM as follows: First, the AGIM retain more advantages of the original GIM than MCGIM, as discussed in Section 1. Table 2 shows another advantage than [34] regarding the packing efficiency. The AGIM shows much higher packing efficiency than [34]. Our results are even better than those of a recent work called RMCGIM [3] (about 80 percent was achieved at this work). For example, the packing efficiency for horse and gargoyle is 75.6 percent and 72.7 percent using MCGIM and is 81.7 percent and 83.6 percent using RMCGIM, respectively. Fig. 15 shows another interesting comparison between two methods regarding the valences of vertices. In this figure, we directly excerpt the bunny example in their paper [34] to compare with our results. There are many high-valence

vertices (more than eight) observed in their zippered regions. In our case, we create the worst case on purpose to the bunny example. However, using Fig. 12, the AGIM, at most, creates 8-valence vertices for this worst case.

The AGIM is suitable for global parameterization techniques based on quadrilateral complexes. In this paper, we first manually built PolyCube-based [37] quadrilateral complexes and then used the PolyCube method to parameterize the surface onto the quadrilateral complexes. Fig. 19 shows several models and their quadrilateral complexes. Finally, for each quad of a quadrilateral complex, we used an adaptive resolution of GIMs to approximate surface parameterized on this quad. To stitch adjacent GIMs with varying sizes, we used the proposed zippering algorithm. Fig. 19 also shows the

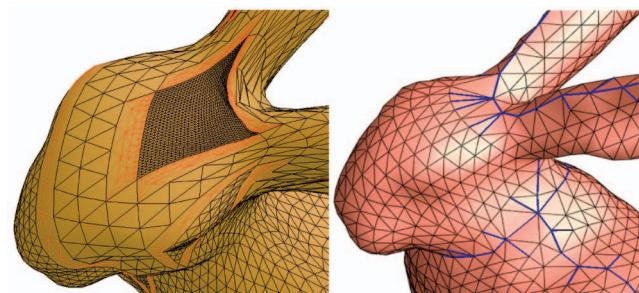


Fig. 15. A close look at reconstructed models comparing AGIM and Sander et al. [34].

TABLE 2
AGIM Packing Efficiency

horse	bunny	venus	dinosaur	gargoyle
90.2%	88.0%	93.7%	83.6%	90.8%

TABLE 3
Reconstruction Error (PSNR) Comparison Using Different Methods

PSNR(no. of vertices)	gargoyle	horse	dragon	feline
MCGIM [34]	85.2 (58,769)	85.3 (55,329)	79.4 (52,059)	82.5 (55,329)
PolyCube [37]	75.93(92,162)	82.22(137,218)	80.56(139,264)	79.25(235,518)
AGIM from [37]	88.3 (56,951)	87.3 (54,510)	81.3 (51,798)	84.2 (52,904)

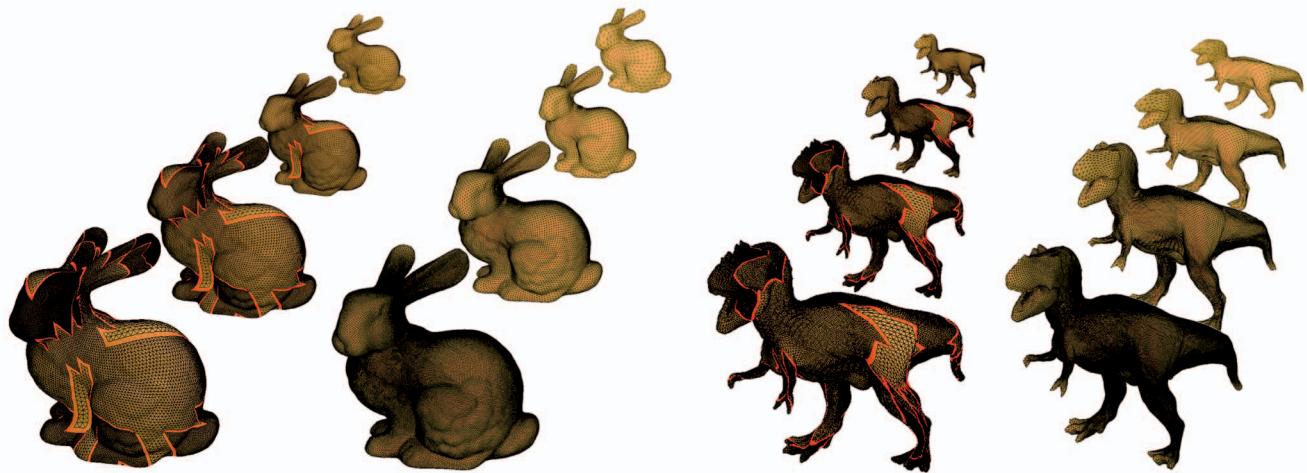


Fig. 16. View-independent LOD comparison between AGIM and Gu et al. [12]. For AGIM LOD, we do not show zippered regions (red) for the most distant one in this figure, since it becomes G_i after this distance, i.e., no need to zipper. If further simplification is needed, we can simplify it similar to Gu et al. [12].

reconstruction results based on PolyCube with AGIMs. Table 3 shows a PSNR comparison with MCGIM for these examples. Using the approximate sampling rates, our method can outperform state-of-the-art MCGIM in terms of PSNR. In this table, we also show PSNR for PolyCube that serves as the input to our AGIM. Our results show that the PolyCube with AGIM can also achieve significant PSNR gain over the PolyCube. For completeness, in Table 1, we show PSNRs for PolyCube with AGIM to experimentally compare with [12], [31], and PolyCube without AGIM. These results show that PolyCube with AGIM can significantly outperform other methods.

Next, the levels of details (LOD) can be easily implemented for AGIM. Fig. 16 shows two view-independent LOD examples. We adapted [26] for our AGIM to generate LODs in these figures. For comparison, we also show LOD

results of GIMs [12]. Our LOD results appear better than that in [12] in their visual appearance. In these two examples, we simply used the distance from the eye to the center of a model as a metric in order to switch LODs. In addition, another advantage of AGIM can allow selective refinements on GIMs instead of globally uniform refinements in [12]. Therefore, it is very straightforward to implement view-dependent LODs using AGIM. In this case, our zippering algorithm can still dynamically stitch GIMs tessellated at different rates; so, no crack occurs. In contrast, it is not easy to have this advantage using MCGIM [34]. For a view-dependent LOD example in Fig. 17, whenever an adaptive GIM is out of view-frustum, its resolution is switched to the lowest resolution in our implementation (see our accompanying video). For this purpose, we test if its corresponding quad of the quadrilateral complex is out



Fig. 17. A view-dependent LOD example using PolyCube with AGIM. Whenever a quad is out of view-frustum, its corresponding AGIM will be switched to its lowest resolution.

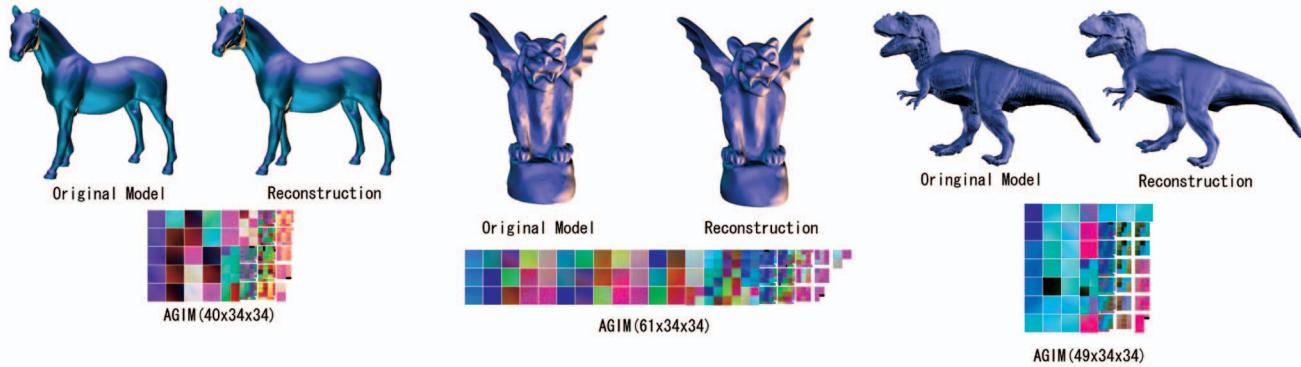


Fig. 18. Results for horse, gargoyle, and dinosaur examples using AGIM.

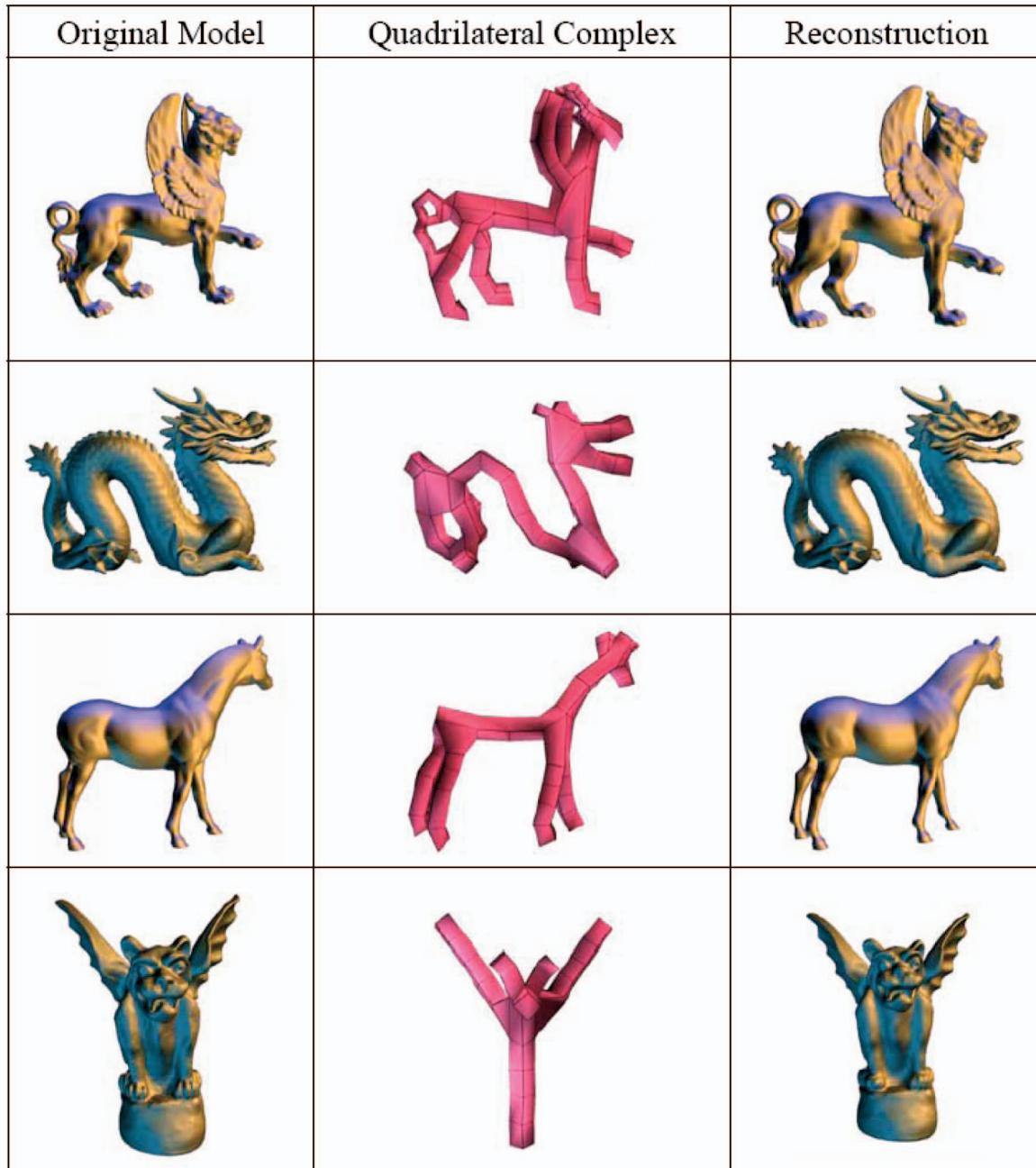


Fig. 19. Results for feline, dragon, horse, and gargoyle using PolyCube [37] with AGIM.

TABLE 4

The Timing Information for View-Dependent LOD Examples
with/without GPU Acceleration

	gargoyle	horse	dragon	feline
vertices	51,376	56,234	77,536	71,336
Charts	90	134	136	230
CPU time	20.29(ms)	22.57(ms)	30.51(ms)	29.30(ms)
GPU time	2.844(ms)	3.061(ms)	3.696(ms)	7.37(ms)
Zippering time	2.28(ms)	1.482(ms)	2.095(ms)	2.161(ms)
View frustum testing	6.602(ms)	6.936(ms)	8.004(ms)	9.510(ms)

Charts: the number of quads for a quadrilateral complex.

of view-frustum or not. Therefore, the cost of this test is also very small (see Table 4).

Finally, there is a trend for modern graphics cards supporting arrays of texture. With this hardware support, we can store each GIM of an AGIM on a texture array. In other words, we will no longer need to pack all GIMs onto a single chart like MCGIM, RMCGIM, etc. In our AGIM implementation, we used an Nvidia G80 graphics card handling arrays of texture directly. Since a texture ID can be allocated to each GIM, each GIM can be independently executed on its own texture memory. Furthermore, the texture coordinate of each GIM can be stored on a Vertex Buffer Object (VBO). Therefore, the multiresolution of each GIM can be efficiently executed on GPU by using a texture array and VBO together. Currently, we used CPU to execute the zippering method. Table 4 shows a detailed timing information and comparison with/without GPU acceleration. With GPU acceleration, our rendering can achieve a real-time performance. When we executed AGIM on CPU alone, we only obtained an interactive frame rate. In this table, the connectivity of zippered regions is computed for each frame, and the timing is obtained from the average of 1,000 running timings for our view-dependent LOD examples.

6.2 Discussion

Experimental results show that AGIM can significantly outperform GIM in terms of PSNR. Furthermore, the PolyCube-based complexes with AGIM can yield a higher PSNR than MCGIM. The GIM is the most extremely regular remeshed representation, and it is easy for hardware-assisted rendering and LOD generation. The proposed AGIM is less extremely regular than GIM. For AGIM, the nonregular triangular areas are on zippering blocks. As shown in Table 5, the number of triangles from these zippering blocks only contributes very small percentages of the total number of triangular meshes, i.e., less than 5 percent. In other words, the AGIM retains more than 95 percent regular representation, and these different resolutions of GIMs can still be highly benefited from hardware-assisted rendering. Therefore, the AGIM can achieve real-time rendering performance, as shown in Table 4. The zippering blocks are rendered by CPU, and they cost only very little CPU time. The GIM can only benefit global view-independent LOD generation. The AGIM can adaptively refine GIMs and zipper them in real time. Figs. 16 and 17 show that the AGIM can benefit both view-independent and view-dependent LOD generations.

On the other hand, MCGIM generates irregular charts and discretizes these charts onto the image. The irregularity of

TABLE 5

Percentages of the Total Number of Triangular Meshes on Zippering Regions

horse	bunny	venus	Dinosaur	gargoyle	dragon	feline
2.2%	3.7%	2.1%	4.1%	3.5%	2.3%	2.6%

chart boundaries forces MCGIM to require complicated sampling processes (interior and boundary rasterizations, and nonmanifold dilation) and, therefore, complicates its zippering algorithm. As reported in [34], both sampling and zippering takes less than one minute and are done in preprocessing, i.e., zippering connectivity must be stored in advance, and it cannot be computed on the fly. Therefore, the MCGIM is not suitable for real-time applications requiring adaptive zippering connectivity such as view-dependent LOD generations. Unlike MCGIM, the AGIM does not need to store zippering connectivity in preprocess, and it can adaptively compute zippering connectivity in real time. In addition, for zippering areas, the AGIM only needs to store the adjacency of GIM blocks, and MCGIM needs to store all pixel connectivity along the irregular chart boundaries. The RMCGIM is a variant of MCGIM, and it partitions meshes into rectangular charts. Generally, the RMCGIM has higher distortion than MCGIM. To preserve a one-to-one texel correspondence across chart boundaries, the RMCGIM requires a complicated process to solve the continuity problem on the shared chart boundaries. As reported in [3], this process can be repeated several times so that all patches have necessary resolution to maintain one-to-one texel correspondence. Like GIM, the RMCGIM cannot adaptively refine each chart, since T-junction will occur and its one-to-one texel correspondence across chart boundaries will be lost. In this paper, the AGIM adapts non-power-of-two (i.e., $2^i + 1$) GIMs. Currently, the restriction of textures to power-of-two dimensions has been relaxed on modern graphics hardware. For example, Hoppe et al. in their original GIM [12] and smooth GIM [3] also adapt non-power-of-two (i.e., $2^i + 1$) GIMs with graphics hardware implementation. In addition, non-power-of-two textures have been promoted from the ARB texture non-power-of-two extensions [28].

7 CONCLUSION AND FUTURE WORK

The AGIM is a practical postprocessing technique to reduce the remeshing error of a GIM parameterization. Given existing planar surface parameterizations [12], [31] as input, experimental results show that the PSNR gain can be significant from AGIM. The AGIM is also very suitable for quaddominant parameterization methods. We show an example of a quadlike PolyCube method with AGIM. As a result, we outperform the state-of-the-art MCGIM method in terms of PSNR using the approximate number of samples. In addition, AGIM has the advantage of selective refinements on GIMs, and therefore, it can easily benefit the implementation of view-dependent LODs. We show a real-time dynamic view-dependent LOD rendering performance partially accelerated by modern GPU hardware. Currently, the zippering algorithm is not accelerated by GPU, since this zippering is beyond hardware programmability of current generation (including Geometry Shader). This

disadvantage may potentially hinder the scalability of our method, since the CPU must be bothered to send ad hoc geometry and connectivity for the zippering part, which is different every time as the used resolution changes. Further work with this approach will be explored in the near future. First, we will apply AGIM to other global parameterization methods such as [18] and [19]. We also like to investigate the possibility of new global parameterization method specialized to the need of AGIM or to dynamically modify the input parameterizations to obtain better PSNR. In addition, some further processing on AGIM such as mesh optimization [16] can improve reconstructed quality. However, this is computationally expensive. In this paper, we adaptively refine the quality of AGIM using only Hausdorff distance. In the future, other metrics such as the normal deviations can be used. In particular, for the rendering purposes, normal accuracy will become important. Finally, we will explore other possible AGIM applications such as 3D morphing [22], [24] or constrained texture mapping [23].

ACKNOWLEDGMENTS

The authors would like to thank Praun and Hoppe [31] for providing their open parameterization data for helping with our experiment. They also thank the anonymous reviewers for helping them to improve this paper. This work was supported by the Landmark Program of the NCKU Top University Project under Contract B0008 and was supported in part by the National Science Council under Contracts NSC-95-2221-E-006-193-MY2 and NSC-96-2628-E-006-200-MY3.

REFERENCES

- [1] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun, "Anisotropic Polygonal Remeshing," *ACM Trans. Graphics*, vol. 22, no. 3, pp. 485-493, 2003.
- [2] N.A. Carr and J.C. Hart, "Meshed Atlases for Real-Time Procedural Solid Texturing," *ACM Trans. Graphics*, vol. 21, no. 2, pp. 106-131, 2002.
- [3] N.A. Carr, J. Hoberock, K. Crane, and J.C. Hart, "Rectangular Multi-Chart Geometry Images," *Proc. Fourth Eurographics Symp. Geometry Processing (SGP '06)*, pp. 181-190, 2006.
- [4] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno, "Planet-Sized Batched Dynamic Adaptive Meshes (P-BDAM)," *Proc. IEEE Visualization '03*, pp. 147-155, Oct. 2003.
- [5] P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: Measuring Error on Simplified Surfaces," *Computer Graphics Forum*, vol. 17, no. 2, pp. 167-174, 1998.
- [6] S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, and J.C. Hart, "Spectral Surface Quadrangulation," *ACM Trans. Graphics*, vol. 25, no. 3, pp. 1057-1066, 2006.
- [7] M.A. Duchaineau, M. Wolinsky, D.E. Sigeti, M.C. Miller, C. Aldrich, and M.B. Mineev-Weinstein, "Roaming Terrain: Real-Time Optimally Adapting Meshes," *IEEE Visualization*, pp. 81-88, 1997.
- [8] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution Analysis of Arbitrary Meshes," *Proc. ACM SIGGRAPH '95*, pp. 173-182, 1995.
- [9] F. Evans, S.S. Skiena, and A. Varshney, "Optimizing Triangle Strips for Fast Rendering," *Proc. IEEE Conf. Visualization*, pp. 319-326, R. Yagel and G.M. Nielson, eds., <http://www.cs.sunysb.edu/stripe/>, 1996.
- [10] M.S. Floater and K. Hormann, *Surface Parameterization: A Tutorial and Survey*, 2005.
- [11] C. Gotsman, X. Gu, and A. Sheffer, "Fundamentals of Spherical Parameterization for 3D Meshes," *ACM Trans. Graphics*, vol. 22, no. 3, pp. 358-363, 2003.
- [12] X. Gu, S.J. Gortler, and H. Hoppe, "Geometry Images," *Proc. ACM SIGGRAPH '02*, pp. 355-361, 2002.
- [13] X. Gu and S.-T. Yau, "Global Conformal Surface Parameterization," *Proc. Eurographics/ACM SIGGRAPH Symp. Geometry Processing (SGP '03)*, pp. 127-137, 2003.
- [14] B. Hernández and I. Rudomin, "Simple Dynamic LOD for Geometry Images," *Proc. Fourth Int'l Conf. Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (GRAPHITE '06)*, pp. 157-163, 2006.
- [15] B.V. Herzen and A.H. Barr, "Accurate Triangulations of Deformed, Intersecting Surfaces," *Proc. ACM SIGGRAPH '87*, pp. 103-110, 1987.
- [16] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization," *Proc. ACM SIGGRAPH '93*, pp. 19-26, 1993.
- [17] J. Ji, E. Wu, S. Li, and X. Liu, "Dynamic LOD on GPU," *Proc. Computer Graphics Int'l (CGI '05)*, pp. 108-114, 2005.
- [18] M. Jin, J. Kim, and X.D. Gu, "Discrete Surface Ricci Flow: Theory and Applications," *Proc. IMA Conf. Math. of Surfaces*, pp. 209-232, 2007.
- [19] M. Jin, F. Luo, and X. Gu, "Computing Surface Hyperbolic Structure and Real Projective Structure," *Proc. ACM Symp. Solid and Physical Modeling (SPM '06)*, pp. 105-116, 2006.
- [20] B.D. Larsen and N.J. Christensen, "Real-Time Terrain Rendering Using Smooth Hardware Optimized Level of Detail," *J. Winter School on Computer Graphics*, Proc. 11th Int'l Conf. Central Europe on Computer Graphics, Visualization and Digital Interactive Media (WSCG '03), vol. 11, no. 2, pp. 282-289, Feb. 2003.
- [21] A.W.F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin, "Maps: Multiresolution Adaptive Parameterization of Surfaces," *Proc. ACM SIGGRAPH '98*, pp. 95-104, 1998.
- [22] T.-Y. Lee and P.-H. Huang, "Fast and Intuitive Metamorphosis of 3D Polyhedral Models Using SMCC Mesh Merging Scheme," *IEEE Trans. Visualization and Computer Graphics*, vol. 9, no. 1, pp. 85-98, Jan.-Mar. 2003.
- [23] T.-Y. Lee, S.-W. Yen, and I.-C. Yeh, "Texture Mapping with Hard Constraints Using Warping Scheme," *IEEE Trans. Visualization and Computer Graphics*, vol. 14, no. 2, pp. 382-395, Mar./Apr. 2008.
- [24] C.-H. Lin and T.-Y. Lee, "Metamorphosis of 3D Polyhedral Models Using Progressive Connectivity Transformations," *IEEE Trans. Visualization and Computer Graphics*, vol. 11, no. 1, pp. 2-12, Jan./Feb. 2005.
- [25] F. Losasso and H. Hoppe, "Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids," *Proc. ACM SIGGRAPH '04*, pp. 769-776, 2004.
- [26] F. Losasso, H. Hoppe, S. Schaefer, and J. Warren, "Smooth Geometry Images," *Proc. Eurographics/ACM SIGGRAPH Symp. Geometry Processing (SGP '03)*, pp. 138-145, 2003.
- [27] M. Marinov and L. Kobbelt, "Direct Anisotropic Quad-Dominant Remeshing," *Proc. 12th Pacific Conf. Computer Graphics and Applications (PG '04)*, pp. 207-216, 2004.
- [28] *What's New in OpenGL 2.1*, OpenGL.org.
- [29] R. Pajarola and E. Gobbetti, "Survey of Semi-Regular Multi-resolution Models for Interactive Terrain Rendering," *Visual Computer*, vol. 23, no. 8, pp. 583-605, 2007.
- [30] G. Peyré and S. Mallat, "Surface Compression with Geometric Bandelets," *ACM Trans. Graphics*, vol. 24, no. 3, pp. 601-608, 2005.
- [31] E. Praun and H. Hoppe, "Spherical Parametrization and Remeshing," *ACM Trans. Graphics*, vol. 22, no. 3, pp. 340-349, 2003.
- [32] B. Purnomo, J.D. Cohen, and S. Kumar, "Seamless Texture Atlases," *Proc. Eurographics/ACM SIGGRAPH Symp. Geometry Processing (SGP '04)*, pp. 65-74, 2004.
- [33] N. Ray, W.C. Li, B. Lévy, A. Sheffer, and P. Alliez, "Periodic Global Parameterization," *ACM Trans. Graphics*, vol. 25, no. 4, pp. 1460-1485, 2006.
- [34] P.V. Sander, Z.J. Wood, S.J. Gortler, J. Snyder, and H. Hoppe, "Multi-Chart Geometry Images," *Proc. Symp. Geometry Processing*, pp. 146-155, 2003.
- [35] J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe, "Inter-Surface Mapping," *Proc. ACM SIGGRAPH '04*, pp. 870-877, 2004.
- [36] D. Steiner and A. Fischer, "Planar Parameterization for Closed Manifold Genus-G Meshes Using Any Type of Positive Weights," *J. Computing and Information Science in Eng.*, vol. 5, no. 2, June 2005.
- [37] M. Tarini, K. Hormann, P. Cignoni, and C. Montani, "Polycube-Maps," *Proc. ACM SIGGRAPH '04*, pp. 853-860, 2004.
- [38] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun, *Designing Quadrangulations with Discrete Harmonic Forms*, 2006.



Chih-Yuan Yao received the BS and MS degrees in computer engineering from the National Cheng-Kung University, Taiwan, in 2002 and 2003, respectively. He is currently working toward the PhD degree at the Department of Computer Science and Information Engineering, National Cheng-Kung University. His research interests is computer graphics.



Tong-Yee Lee received the PhD degree in computer engineering from Washington State University, Pullman, in May 1995. He is currently a professor in the Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan, R.O.C. He served as a member of the international program committees of several conferences including IEEE Visualization, Pacific Graphics, the IEEE Pacific Visualization Symposium, the IEEE-EMBS International Conference on Information Technology and Applications in Biomedicine, the International Conference on Artificial Reality and Telexistence, and the International Conference in Central Europe on Computer Graphics, Visualization, and Computer Vision. He leads the Computer Graphics Group, Visual System Laboratory, National Cheng-Kung University (<http://graphics.csie.ncku.edu.tw/>). His current research interests include computer graphics, nonphotorealistic rendering, image-based rendering, visualization, virtual reality, surgical simulation, medical visualization and medical system, and distributed and collaborative virtual environments. He is an associate editor for the *IEEE Transactions on Information Technology in Biomedicine* from 2007 to 2010. He is also on the editorial advisory board of the *Journal Recent Patents on Engineering*, an editor of the *Journal on Information Science and Engineering* and a region editor of the *Journal of Software Engineering*. He is a member of the IEEE and the ACM.

▷ For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.