Interactive texture design and synthesis from mesh sketches

Article *in* Frontiers of Computer Science (electronic) · April 2014

DOI: 10.1007/s11704-014-3285-5

5 authors, including:

Lili Wang
beihang univeristy
47 PUBLICATIONS **169** CITATIONS
SEE PROFILE

Qinglin Qi
Beihang University (BUAA)
35 PUBLICATIONS **5,714** CITATIONS
SEE PROFILE

Wei Ke
Macao Polytechnic University
79 PUBLICATIONS **450** CITATIONS
SEE PROFILE

Front.Comput.Sci.
DOI

RESEARCH ARTICLE

Interactive Texture Design and Synthesis from Mesh Sketches

Lili WANG (✉)[1],Qinglin QI [1], Yi CHEN [1], Wei KE [2], Aimin HAO [1]

1 State Key Laboratory of Virtual Reality Technology and Systems, School of Computer Science & Engineering, Beihang University, Beijing 100191, China
2 Macao Polytechnic Institute, Macao, China

Abstract Geometry mesh introduces user control into texture synthesis and editing, and brings more variations in the synthesized results. But still two problems related remain in need of better solutions. One problem is generating the meshes with desired size and pattern efficiently from easier user inputs. The other problem is improving the quality of synthesized results with mesh information. We present a new two-step texture design and synthesis method that addresses these two problems. Besides example texture, a small piece of mesh sketch drawn by hand or detected from example texture is input to our algorithm. And then a mesh synthesis method of geometry space is provided to avoid optimizations cell by cell. Distance and orientation features are introduced to improve the quality of mesh rasterization. Results show that with our method, users can design and synthesize textures from mesh sketches easily and interactively.

Keywords Texture Synthesis, Texture Design, Feature-based Method, Structural Texture

1 Introduction

Texture mapping is a technique that applies raster images on the surface to enhance appearance of 3D model without adding extra geometry details. It is a uniquely powerful method in interactive computer graphics. Research efforts of texture synthesis have produced the methods that generate high-resolution texture algorithmically. Texture synthesis methods can be classified as procedural or sample-based methods. Procedural methods use, for example, turbulence or Perlin noise functions to generate textures that have repetitive patterns or self-similarities. Sample-based methods, like ours, assemble textures from modified versions of input patches. The quality of texture synthesis, especially for the structural texture, still has some space to improve. Geometry information based texture synthesis and editing use geometry mesh to represent the shape and distribution of the texture elements, which can provide more cues to guide texture synthesis in the same or similar patterns of the inputs, and produce some good results.

However, some problems related to geometry mesh based texture synthesis and editing methods remain in need of better solutions. One is the complex user inputs. For synthesizing texture with the same or similar patterns as the examples, the user is required to provide example textures for elements, mesh patches for shape and arrangements, and a large initial mesh for the optimization step to remove the noticeable repetitions. Many other mesh patches are also required if we want to obtain variable synthesized results. The second problem is that the regular pixel-based neighborhood matching and sub-texture methods in mesh rasterization do not consider the distance and orientation cues that the mesh provides, so usually they have high cost of the best matching, which introduces the loss of texture details and unacceptable artifacts in vision.

To addresses these two problems, we propose a two-step method to help user synthesize and design the texture more easily. The first step of our texture synthesis method is mesh synthesis, which uses a small piece of mesh sketch as input and generates a large texture mesh with the similar features

Received August 09, 2013; accepted month dd, yyyy

E-mail: *wanglily@buaa.edu.cn*

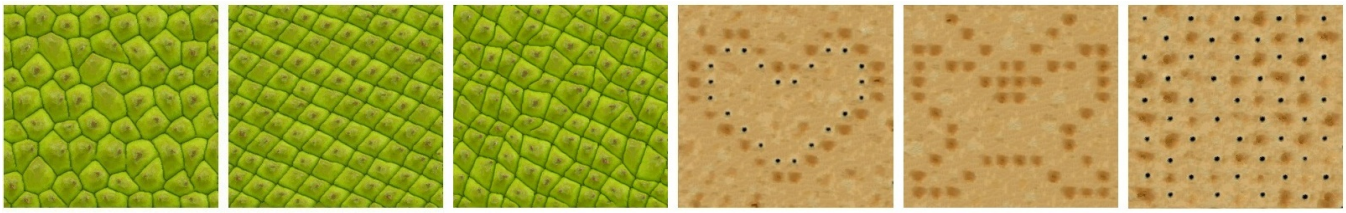


Fig. 1: *The synthesized textures of Jackfruit (left 3 images) and Cracker (right 3 images) using the method proposed in this paper. A heart and a butterfly patterns are added on the texture meshes to introduce diversifications of texture design and synthesis.*

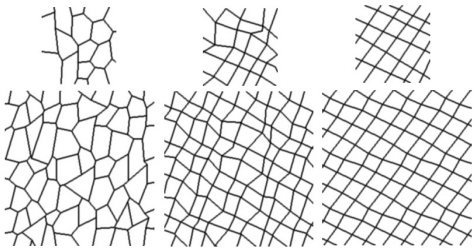


Fig. 2: *Texture meshes(bottom) synthesized from the input mesh sketches (top).*

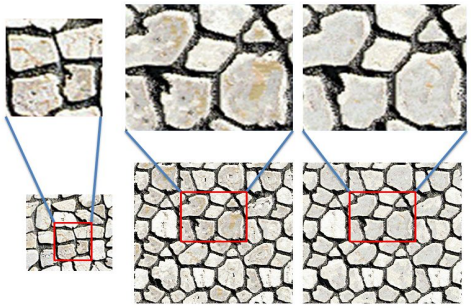


Fig. 3: *The result synthesized with orientation features (middle) keeps the similar details as the input example (left), the details in the results without orientation features are blur (right).*

and patterns as output. In this paper, mesh sketch refers to the mesh drawn by hand; texture mesh refers to edge and vertex based geometry mesh, and it will introduce some constrains in the later process. The second step of our method is mesh rasterization, which synthesizes patches with shape and distribution cues provided by the texture mesh. Because these synthesized patches are filled into cells of the texture mesh, as same as rastering triangle, we call this step as mesh rasterization.

Besides example texture, only a small piece of mesh sketch drawn by hand is needed. User can also modify the sketch on-line, and get many different results interactively. Figure 1 shows the textures synthesized with two example textures: Jackfruit and Cracker. 3 images on right also show the texture diversification generated with some extra patterns of mesh. We avoid complex user inputs by a mesh synthesis step. A seamless mesh patch is constructed from the mesh sketch, and then is tiled to generate a large texture mesh. A parallel refining is used to remove the repetitions of the large mesh. Figure 2 shows three texture meshes (bottom) generated from the small input sketches drawn by hand (top).

For the second problem, we introduce the distance and orientation features to give constrains in mesh rasterization. Distance feature indicates how far the pixel is from the edges of the mesh, which will help to maintain arrangement and basic shape for texture elements of inputs. Orientation feature extends the neighborhood searching from fixed orientation domain to changeable one, which can reduce the neighborhood matching cost, and avoid details loss in the result texture. Figure 3 shows comparison among the input example texture (left), the results with (middle) and without using the orientation features (right).We also refer the reader to the accompanying video.

The method we propose in this paper provides user an easy and flexible way to design and synthesize textures with the patterns in the user's mind. Comparing with the previous methods, our method has the following characters:

1. Besides example texture, user can provide mesh sketch drawn by hand to constrain shape and distribution of synthesized texture elements.
2. The mesh synthesis step is very fast, user can modify the sketch and have a result texture mesh in real-time.
3. In the mesh rasterization step, we consider the distance and orientation features of the pixel, which help the synthesized results maintain more details of the example texture.

2 Related Work

We target design and synthesis of structural textures from input examples and patterns interactively, and the most

related work include feature-based texture synthesis, element arrangement, patch tiling and real-time texture synthesis. For more example-based texture synthesis methods, one can check a state-of-the-art report in [1].

2.1 Feature-based texture synthesis

Features in example textures always contain the shape, scale, orientation and color of the elements, so a lot of researches focus on feature-guide texture synthesis.

Feature-based method [2] extracts a small feature map containing some easy-detect structural features such as curves and ridges from the example, and then uses the feature matching and deformation to build a new big feature map, which is used to guide texture synthesis. For near-regular texture, user-assisted lattice is extracted from example texture according to the regular tile, and geometric deformation field is constructed to provide the geometry data to lighting and shape deformation [3]. Features are also used into texture diversification. Feature-based warping and blending introduce the shape variations of texture elements, and generate more complex textures with diversifications [4]. Multiple feature maps are detected from the example textures [5], and matched with warping function to construct a simplicial complex for morphable interpolation. In [6], the features morph using a level set advection approach, and constrain the synthesis of texture with the material. Compare to the a global morphing transformation on the entire feature map, the method in [7] interpolates the features in the local regions of feature maps, and then synthesize the texture with blend patches. Moreover, the texture feature can introduce not only the spatial variations of texture, but also the temporal ones. The features from highly dynamic fluid phenomena are captured, and used to generate the dynamic textures in videos [8]. Another variation of feature-based texture synthesis is constrained texture synthesis. The most reprehensive work is Bala's method [9]. Energy minimization is used to introduce constrains from both examples and maps provided by the users to the synthesis process. The details of the result texture preserve good structure from the input constrains, but it is not a real-time method.

Texture splicing [10] is the work closed to our method. It extracted elements from the example texture, and constructed displacement of distribution from both example texture and new patterns to indicate the arrangement of elements. The results of this method are very similar to ours, but it had dozens of seconds pre-computation time to create candidate sets for example textures. Our method is a real-time one, which doesn't require any pre-computation.

For texture with structural features, Risser [11] introduced a multi-scale descriptor, and brought many diversifications of the results while maintain global structures. Kim [12] observed the symmetry properties of the example texture, and build symmetry representations of them, which can be used to transfer and process the patterns in textures.

Our method is also a feature-based texture design and synthesis method. Compare to the methods above,it avoids complex user inputs, and maintains good features both in the example texture and the mesh sketch. The extra patterns added in the texture mesh make the texture design more flexible.

2.2 Element arrangement

Feature map includes not only the shape, scale, orientation and color information, but also arrangement cues for the texture elements. The density and distribution of the texture elements are considered into feature map generation in [4], so the synthesized texture with this method is suited to mimic the variations in the nature. The method in [6] introduces material axis that works with feature map to guide element arrangement.

Compare with the feature map, control map gives more powerful constrains to element arrangement in higher level. An input control map is used to indicate the arrangement in some previous work [13–15]. Recently a texture synthesis method using the multiple layers of control maps constructed automatically from example texture is introduced in [16].

There are still some other methods to guide the element arrangement. A local growth based procedural modeling system is proposed to arrange the texture elements in [17]. It focused on how to generate the arrangement with the similar pattern of input reference, and considered less on the pixel-level synthesis. The orientations of ridges on the 3D surfaces also give some cues to the patterns of arrangement in [18]. In [19], the author demonstrates the results using flow fields into controllable synthesis of textures. More complex arrangement based physical rules for discrete 3D elements was introduce in [20], which considered the size, shape, etc. properties of the individual elements, and gave more reasonable layout of the elements.

In our method, the input mesh sketch can be used to indicate and diversify the arrangement of the elements.

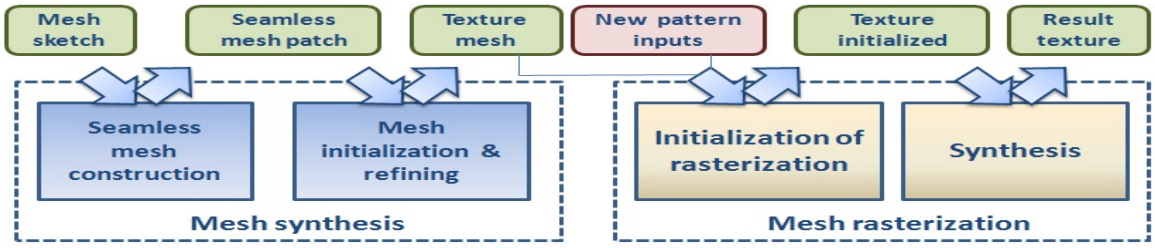


Fig. 4: *Overview of the mesh sketches based texture synthesis pipeline.*

2.3 Patch tiling

The main problem of patch-based texture synthesis is removing the seams in tiling. A small set of Wang Tiles are constructed from examples, and then the continuous textures are created with these tiles stochastically [21]. Graphcut uses error function to find the best boundary cut path with least error in the overlapped patches, and fits the patches seamlessly [14, 15]. For the irregularly shaped patches, cutting and stitching patches to generate new texture images with minimized visual discontinuity are discussed in [22]. Irregular patch growing on the 3D surface with the optimization provides another solution to the seamless tiling [23]. Kun Zhou etc. [24] proposed mesh quilting for tiling 3D texture patches, and used triangle based mesh to provide more general and flexible deformations. While our goal is 2D texture synthesis and editing, our method exploits edge based 2D mesh tiling and deformation because of its simplicity, which only occupies very small part of the computation in our method (<5%). Lasram [25] accelerated the patch-based texture synthesis. The method starts from the results with poor quality, and refine them with selecting and stitching the patches in parallel. Compare to our method, this method has good performance, but it doesn't allow the more intersections by users in the synthesis.

There are two steps in our method that use patches. One is the construction of the texture mesh. A large seamless texture mesh is generated by tiling a 2D mesh patch with considering of a matching cost function and deformation of the edges. The other is initialization of mesh rasterization, in which the texture patches in the example texture are warped or scaled to fit the cell of the texture mesh.

2.4 Real-time texture synthesis

Texture synthesis is a time consume task in computer graphics, and researchers tried to present some real-time solutions for it. One idea is using pre-computation. Analysis on the example texture is executed in the pre-processing phrase off line, and the neighborhood relations are stored into a jump map, which will be indexed rapidly in the run-time phrase for texture synthesis [26].

Another option for reducing the time cost is patch-based method. Patch-based sampling with a local Markov random field is a faster method than pixel-based method, and generates high quality results [27]. Lefebvre [28] proposed parallel method to synthesize the texture in real-time, but it needs a few minutes to preprocess the example texture sized with 64*64, or 128*128. Coordinate jitter is used to introduce some diversification of the texture, which makes the synthesis controllable. While the interactions provided by this method aren't intuitive enough for users such as texture designers to use.

The most related work of our method is real-time texture synthesis with image-mesh analogies [29]. Both these two methods use 2D meshes to indicate the distribution of texture elements, and raster the meshes to generate the output texture with GPU. But our method has two advantages over Dischler's method. 1) We avoid the strict user input of periodic arrangement map by a mesh synthesis step, which allow more flexible user interactions to edit textures. Moreover, our parallel mesh synthesis removes two iterative procedures in texture-mesh optimization, and is more efficient. 2) In mesh rasterization, distance and orientation feature maps are added to preserve the details of the example texture.

There exist some schemes to synthesize meshes, such as Barla's method in [30]. It analyzed input stroke patterns, and then synthesized from elements to patterns, also added some variations. It took 5 to 10 seconds to synthesize 2D patterns. In our method, we tile the seamless mesh patches found from mesh sketches, and then remove repetitions in the mesh according to mesh refining. Compare to Barla's method, our texture mesh synthesis is more efficient due to patch tiling (Ours only takes 3ms).

The goal of our method is to design and synthesize the textures with the examples and the input mesh sketches interactively. Thus we cannot rely on the pre-computation off line. We combine the patch-based idea with parallel

computation on the geometry meshes, and achieve interactive frame rates.

3 The algorithm

In our method, a texture is designed and synthesized with two major steps : mesh synthesis and rasterization. In mesh synthesis, a small piece of mesh sketch either drawn by hand or detected from the example texture is used as input, from which the seamless mesh patch is constructed and tiled repeatedly to form a basic texture mesh with desired size and pattern. We refine this texture mesh with vertex moving to avoid the visible repetitions. Moreover, user can add extra patterns on this mesh, and many interesting results are achieved when the example texture has several texture elements with different appearance. In mesh rasterization, distance feature is determined for each pixel, and then used to guide texture initialization. Orientation feature that indicates the directions of edges is also computed to give constrains for neighborhood matching in the synthesis. We show all these steps in Figure 4.

3.1 Mesh Synthesis

Mesh synthesis is to generate the texture mesh with desired size and pattern from the mesh sketch. Geometry mesh extends the synthesis from image space into geometry space, so some geometry methods for graphics primitive, such as edge matching and mesh deformation, are used in our method. In mesh synthesis, first we select a mesh patch with smaller self-matching cost from the input mesh sketch as a candidate. Second, we deform the edges of the candidate to construct a seamless mesh patch, which is tiled repeatedly to generate a large texture mesh. At last, a refining step is applied to remove the noticeable repetitions in the texture mesh. In this section, we start from some definitions of matching cost, and then explain how to use them in the seamless mesh patch construction. Texture mesh refining is introduced at the last part of this section.

Edge Matching Cost

We refer the segments in mesh as edges. In order to measure how one edge overlap with another edge, an edge matching cost function is introduced. e_1 and e_2 are two edges with the four end points P^1_{in}, P^1_{out}, P^2_{in} and P^2_{out} (Figure 5). We call the red lines that the edges cross over as reference boundaries. Thus, the matching cost $c(e_1, e_2)$ of e_2 and e_1 is defined in Equation 1 when the two reference boundaries are overlapped.

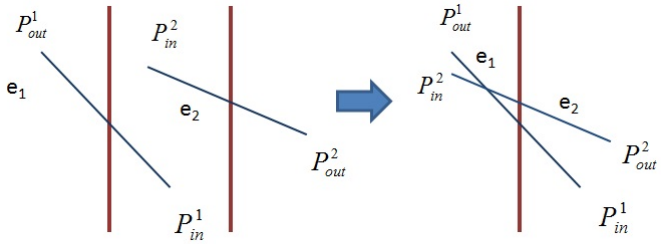


Fig. 5: *Edges matching.*

$$c(e_1, e_2) = (1+ \| \overrightarrow{P^1_{in}P^2_{out}} \|)* \\ (1 + \lambda) * (\| n(\overrightarrow{P^1_{in}P^2_{out}}) - n(\overrightarrow{P^1_{in}P^2_{out}}) \|) \quad (1)$$

where $n(\cdot)$ is the normalize function of the vector, and $\| \cdot \|$ represents the length of the vector. λ is a weight parameter, which indicates how the angle between e_1 to e_2 affects the cost. The matching cost reaches to zero only if e_1 coincides with e_2 entirely.

Boundary Matching

Boundary matching is to find a boundary with the smallest matching cost for a given boundary. The matching cost of two boundaries can be computed with the matching costs of the edges crossing over them.

In figure 6, b_1 and b_2 are a pair of boundaries who have the same length. m edges cross over b_1 while n edges cross over b_2. If m is not equal to n, b_1 doesn't match b_2 , and we set their matching cost to some large values. Otherwise, the matching cost is computed as follow(Equation 2):

$$c(b_1, b_2) = \frac{\sum_{i=1}^{n} c(e^i_1, e^i_2)}{n} \quad (2)$$

Mesh Patch Self-matching

Mesh patch self-matching is to find a mesh patch from the input mesh sketch that matches itself well with smaller cost of matching.

If the mesh patch is tiled repeatedly, its right boundary will match its left one, the top boundary will match the bottom one, and vice versa. Thus, the cost function of patch self matching can be written as Equation 3:

$$c(P) = c(b_{left}, b_{right}) + c(b_{top}, b_{bottom}) \quad (3)$$

where b_{left}, b_{right},b_{top} and b_{bottom} are the four boundaries of the patch P.

Smaller matching cost means less seam in the result mesh when we tile a patch repeatedly. We call the patch that has zero self matching cost as seamless patch.

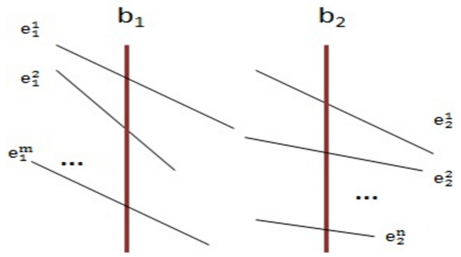


Fig. 6: *Boundary matching.*

Seamless Mesh Patch Construction

The seamless mesh patch is the basic unit to generate the texture mesh for the output texture, and can be constructed in two steps: candidate selection and edge deformation.

The candidate patch P_c is a piece of mesh cut from the input sketch with smaller self matching cost. We crop several rectangle patches from the sketch, and refer them as P_i ($i = 1$ to n). S is a set containing all P_i. Then, the self matching costs of all P_i are calculated. The one with smallest matching cost is used as the candidate for constructing the seamless patch with Equation 4 .

$$P_c = \arg\min_i c(P_i), \qquad P_i \in S \tag{4}$$

In most cases, tiling the candidate directly cannot generate a seamless texture mesh, i.e. the candidate mesh need to be deformed before it is applied to tiling. The vertices related to the candidate can be divided into two groups: one is for the vertices of edges crossing over the boundaries of the candidate (red vertices are outside of the candidate, and purple ones are inside, see in Figure 7), and the other is for the vertices that are inside of the candidate(navy ones). For the vertices in the first group, we connect them with the corresponding vertices on the edges they match, and then move them half distances on the way to the corresponding vertices. The positions of the vertices in the second group are also updated to avoid artifacts. The displacement dsp of the vertex v in the second group can be computed as follow (Equation 5):

$$dsp(v) = \frac{\sum_k \omega_k dsp(v_{in}^k)}{\sum_k \omega_k}, \qquad v_{in}^k \in V_{in} \tag{5}$$

$$\omega_k = \frac{1}{\| v - v_{in}^k \|} \tag{6}$$

where V_{in} is a set that contains all the purple vertices v_{in}^k of the candidate, ω_k is an inverse of the distance between v and v_{in}^k, and used as the weight for the displacement $dsp(v_{in}^k)$ of v_{in}^k.

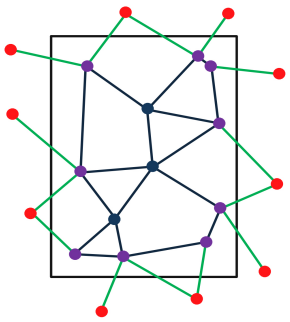


Fig. 7: *A mesh patch: green segments are the edges crossing over the boundaries of the patch, red and purple vertices are the end points of green edges, the navy vertices are inside of the patch.*

After mesh deformation, the seamless mesh patch is tiled to generate a large texture mesh for the output texture.

Texture mesh refining

There are a lot of repetitions in the texture mesh because there is only one patch for tiling. Moving the vertices in the texture mesh with considering the structure of the sketch can remove most repetitions in the texture mesh and preserve the local features of user input.

Several non-overlapped round disks are selected randomly from the input mesh sketch. We refer the set containing all vertices q_i inside the disk Dr as S. Then we put the disks on the texture mesh, and the vertices in the texture mesh touched by the disks will be moved. For a given vertex p touched by Dr, its displacement is determined by Equation 7.

$$dsp(p) = w * \frac{\sum_i w_i * \overrightarrow{pq_i}}{\sum_i w_i} + R, q_i \in S'(p, r) \tag{7}$$

$$w_i = \frac{1}{\| \overrightarrow{pq_i} \|} \tag{8}$$

where $S'(p, r)$ is a sub-set of S, and only includes q_i inside the neighborhood of p with r radius. w_i is weight that is inversely proportional to the distance from p to q_i. w is a scale factor of the displacement. R is a very small displacement generated randomly.

Several iterations of mesh refining can be applied for the more natural texture mesh.

3.2 Mesh Rasterization

Distance and orientation maps

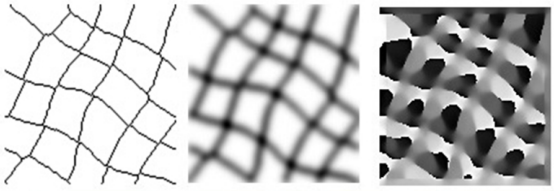


Fig. 8: *The distance map (middle) and the orientation map (right) constructed from the texture mesh (left).*

Distance and orientation features determined from the texture mesh introduce some constrains in mesh rasterization for better results. Each pixel in the output texture has its own distance and orientation features, so the distance and orientation maps for the output texture are constructed. We raster the texture mesh into a two-value bit mask C with 1 and 0, 1 means edges and 0 means background. C is as the same size as that of the output image.

For every pixel in the output texture, its distance feature represents the distance from the pixel to the nearby edges in the corresponding location of the texture mesh. The distance feature $D(p)$ of a given pixel p can be determined by the equations below (Equation 9, 10):

$$D(p) = \frac{\sum_i g(\| \overrightarrow{pp_i} \|) C(p_i)}{\sum_i g(\| \overrightarrow{pp_i} \|)}, \qquad p_i \in N(p) \tag{9}$$

$$g(d) = \frac{1}{\sqrt{2\pi\delta^2}} e^{-d^2/(2\delta^2)} \tag{10}$$

where N is the set containing all the pixel p_i in the neighborhood of p, $g(d)$ is the Gaussian function for d. δ is a parameter of Gaussian function.

Orientation feature represents the direction that the pixel in the output texture pointing to its nearest edge, which locates at the corresponding place in the texture mesh. The orientation feature $\overrightarrow{V(p)}$ for a pixel p is computed as follow (Equation 11, 12):

$$\overrightarrow{V(p)} = \frac{\sum_i w(p_i)\overrightarrow{pp_i}}{\sum_i w(p_i)}, \qquad p_i \in N(p) \tag{11}$$

$$w(p_i) = \frac{D(p_i)}{\| \overrightarrow{pp_i} \|} \tag{12}$$

where $\overrightarrow{pp_i}$ represents the vector from p to p_i, and w is a weight, which can be computed as Equation 12.

Figure 8 shows the distance map (middle) and orientation map (right) computed from the texture mesh (left) with our method.

Patch Initialization

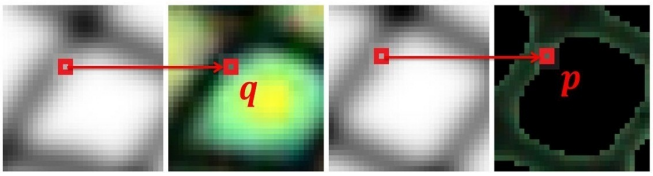


Fig. 9: *Edge initialization, from left to right: the distance map for the example texture, the example texture, the distance map for the result texture, the result texture with edges initialized.*

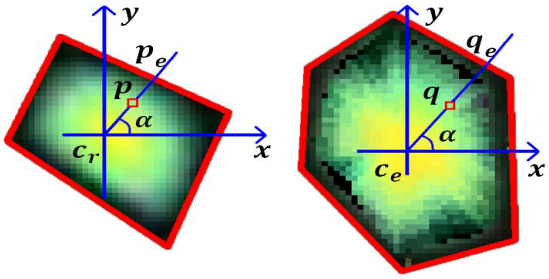


Fig. 10: *Radial warping for patch initialization*

Initialization is to improve the quality of the synthesis. The pixels in the output texture corresponding to the 0-value region and non-zero region (edge region) of the distance map are initialized separately.

The distance map of the example texture is computed first. Then, for a given pixel p of the output texture corresponding to the non-zero value pixel in its distance map, the pixel q in the example texture with the most similar distance feature is found, then q's color will be used to initialize p (See Equation 13 and Figure 9).

$$q = \arg\min_i (D(p) - D(q_i)) \tag{13}$$

In order to obtain a better initialization for the pixels of the output texture corresponding to 0-value region in its distance map, we classify the example textures into two categories, which will be initialized with different methods. Elements in the textures of the first category are not sensitive to deformation, i.e. the elements can be stretched to fit the shape of the texture mesh. While the elements in the textures of the second category have obvious structural features,and we scale these elements overall to preserve their features.

We refer texture patches as pixel sets in both the example texture and the output texture, which are surrounded by the edges of the corresponding meshes. For a given patch P_r in the output texture, a patch P_e from the example texture is found randomly. Radial warping is used to initialize P_r for the textures in the first category. The centers c_r and c_e

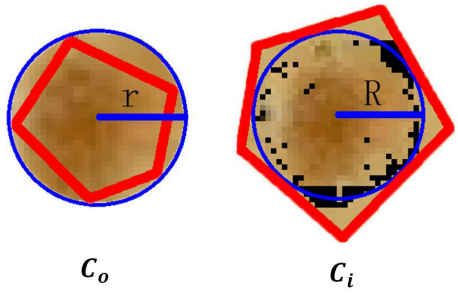


Fig. 11: *Overall scaling for patch initialization*

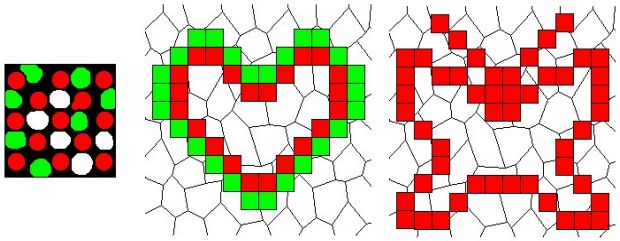


Fig. 12: *Color map for the example (left), for the result texture with heart pattern (middle), for the result texture with butterfly pattern (right).*

of P_r and P_e are aligned together. For each pixel p in P_r corresponding to the 0-value region in the distance map, we connect p with c_r, and then extend it along the vector $\overrightarrow{c_r p}$ to intersect with the edge in the corresponding mesh. If the intersection is represented as p_e, the ratio $\|c_r p\|/\|c_r p_e\|$ and the direction of $\overrightarrow{c_r p}$ are used to index color of q from the example texture as in Figure 10.

For the texture in the second category, a circumcircle C_o of P_r and an incircle C_i of P_e are constructed. We refer R and r as the radius of C_o and C_i. The overall scaling with the coefficient r/R is used to initialize the pixels in P_r corresponding to 0-value region in its distance map (See in Figure 11).

Although our method can also integrate more complex initialization, we adapted these radial warping based methods due to their fast computation. Moreover, the color map as in Figure 12 can be used to generate many more interesting results. Color map (left) is a map that use different colors to indicate the elements with different features of the example texture. Users can add extra patterns to the texture mesh (middle and right) by setting the colors. In patch initialization, for a given a P_r with a mesh color, only P_e with the same color in color map can be copied or warped.

Pixel synthesis

Pixel-based synthesis is adapted to refine the results of initialization. In regular neighborhood matching algorithm, the neighborhood of a pixel is constructed with a fixed orientation as in Figure 13 (middle). But for structural texture, the appearance of patch is often related to the shape of its texture mesh. Because the orientation feature for each pixel represents the direction pointing to its nearest edge in the mesh, we encode this feature into neighborhood construction. First, for the pixels in both the example texture and the output texture, the square neighborhoods oriented according to the orientation vectors in their orientation feature map are built (left and right). Then, the least cost

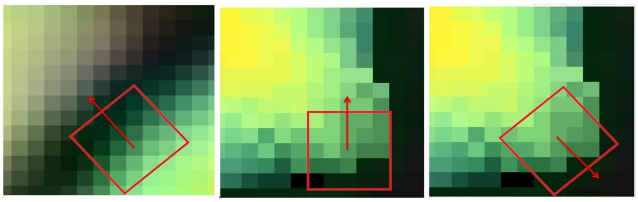


Fig. 13: *The neighborhood with changeable orientation for the example texture (left) and the matching neighborhood in our method (right) compare to regular neighborhood searched (middle).*

matching based on these changeable orientation neighborhood is used into the synthesis.

In order to accelerate the synthesis, the matching areas in the example texture are localized for edges region and 0-value region respectively. The iterative process of synthesis can also be used to improve the quality of the output texture.

4 Results and discussion

In this section we discuss the quality of the textures synthesized by our method, report performance measurements, and discuss the limitations of our method.

4.1 Quality

First, we have tested our algorithm to find some candidates of seamless patches from the user input sketches. Figure 14 shows 9 seamless candidates (in the black rectangles) cropped from 9 different sketches drawn by hand, which indicate if length of mesh edge is 1/3 less than the size of mesh sketch, the candidates have reasonable sizes to construct the seamless patches.

Some parameter variations introduce the diversifications of the texture mesh. The most important one is r in Equation 7. It refers to the radius of the neighborhood for the vertex of texture mesh. The other vertices inside of this neighborhood influence the displacement computation of the vertex in the

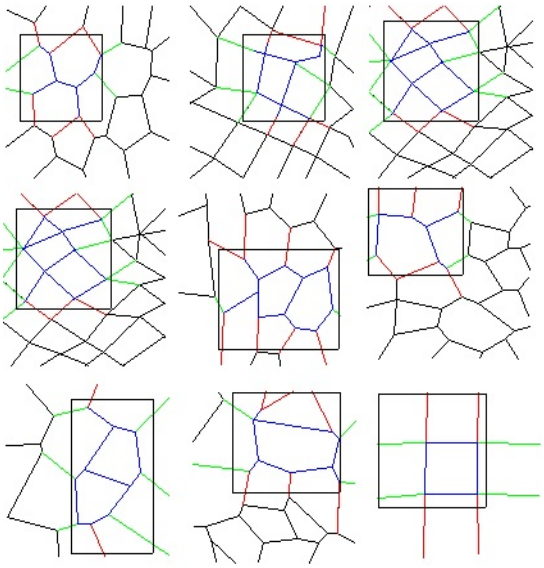


Fig. 14: *the candidates of the seamless patches.*

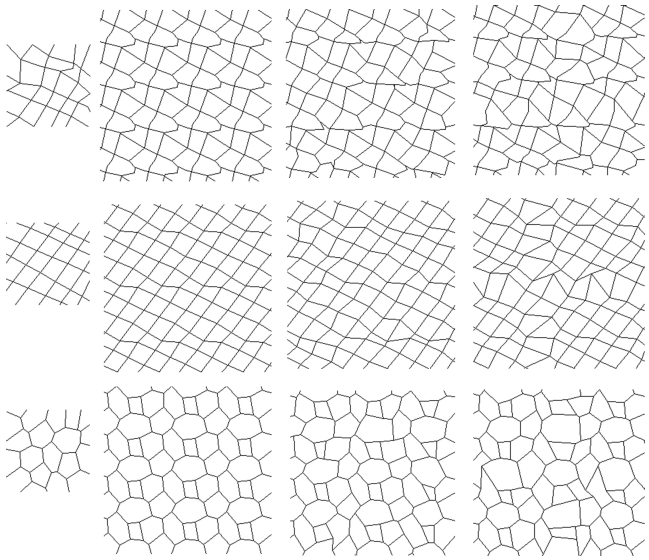


Fig. 15: *Texture meshes generated with the parameter r variation.(a)Input mesh sketch,(b)(c) and (d)Texture mesh generated with r = 5,15,25.*

mesh refining step. Using smaller r causes more regular texture mesh, while using larger r leads to more natural texture mesh (See Figure 15).

Then we have applied our technique to generate several textures: Jackfruit (Figure 1, left), Cracker (right), Pineapple (Figure 16, row 2), Bricks (row 3), Mosaic (row 4), Grid (row 5) and Standard (row 6). For each texture in Figure 16, 3 mesh sketches(row 1) are used to guide the synthesis. The results demonstrate that our method preserves both the features of element from the example textures and the structures of the input mesh sketches. No obvious artifacts are visible.

The orientation features for the pixels help us to maintain the details in the synthesis. We compare the results with and without considering the orientation features in Figure 3. The result of our method (Figure 3, middle) contains the similar details as the example texture (left). The blurred appearance is showed in the result of the regular neighborhood method (right).

The extra patterns can be added into the texture mesh, which bring more variations into texture design and synthesis. For Cracker in Figure 1 (right), different colors are used in the color map of the example texture (Figure 12 left) to point out the different types of texture elements: the hole, the brown region and the origin region of crackers. User can assign the patterns in the color map of the texture mesh (middle, right), which results in different appearances of the synthesized textures. Another example is showed in Figure 17.

Moreover, our method can apply to not only structural textures, but also non-structural textures. For non-structured textures, the only process to be changed is generating the sample mesh randomly. Figure 18 shows some results for the textures with statistic characters, such as Sands (left) and Marble (right). No visible seams can be seen.

4.2 Performance

Performance was measured on an Intel Core i7-2600 3.4GHz PC with an NVIDIA GeForce GTX 780, 1,280 MB graphics card, 4G RAM. Except mesh synthesis, all steps in our algorithm are implemented on GPU with CUDA 4.0. According to the CUDA thread organization, the computation can be divided into 3 kernels.

Kernel 1 is for distance and orientation maps construction, and edge initialization for each pixel of the result texture; **Kernel 2** is initializing the region in the result texture corresponding to 0-value region in the distance map; **Kernel 3** is synthesizing and refining the pixels in the output texture.

We do the performance test on the textures including Jackfruit, Cracker, Pineapple, Bricks, Mosaic, Grid and Standard. The resolution of the input example is 128*128, and the outputs have two different resolutions: 128*128 and 256*256. For 128*128 case, our method takes about 2ms for mesh synthesis, about 62ms to initialize the result texture and about 64ms to refine the pixels. For 256*256 case, the time cost of each step is: 4ms, 95ms and 257ms. The time

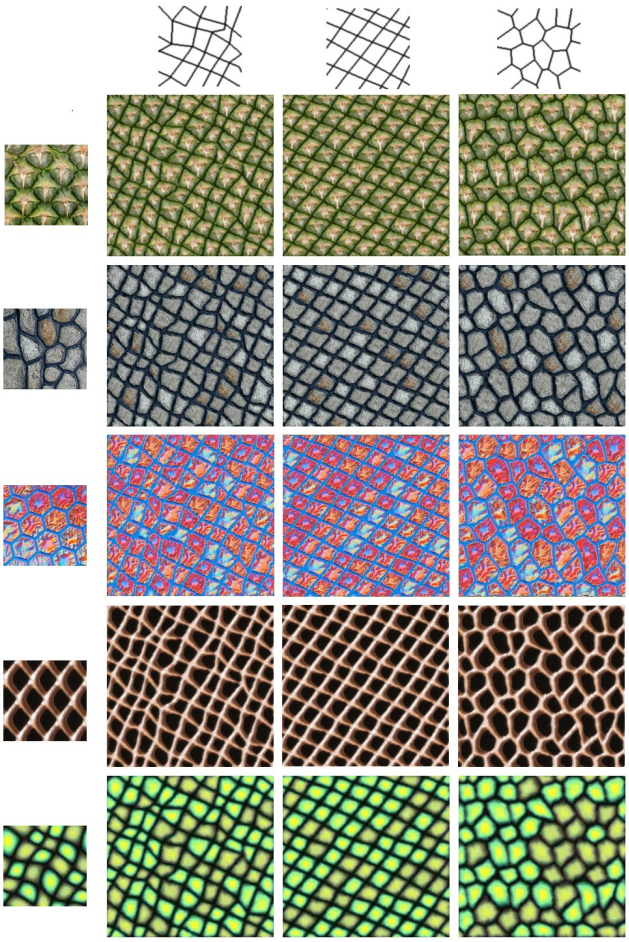


Fig. 16: *Synthesis results for Pineapple(row 2),Bricks (row 3), Mosaic (row 4), Grid (row 5) and Standard (row 6). 3 mesh sketches(row 1) are used in synthesis.*

difference of each step for the different textures is less than 3ms. The results indicate that the performance of our method is not sensitive to different meshes and texture patterns. Thus, our method can be used to design and synthesize the structural texture interactively.

Texture splicing[10] has some similarities with our method. It also generates a new texture with the texture A's elements and B's distribution. However, texture splicing does not allow user interactions in the synthesis process directly. Besides pre-computation part, texture splicing has three steps: mapping creation from one distribution to another, deformation and refinement, which take almost the same computation cost as the mesh rasterization of our method. But foreground background separation in pre-computation part is time consumed. It takes 10 times as long as all other steps of texture splicing. Moreover, the quality of foreground and background detected seriously

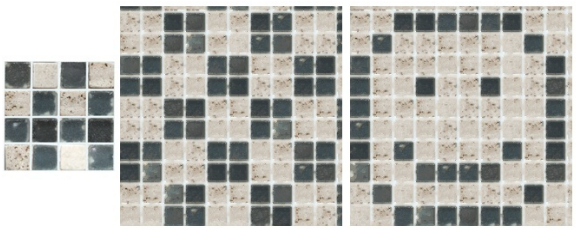


Fig. 17: *Mosaic texture with square (middle) and smile face patterns (right) generated from the input example (left).*

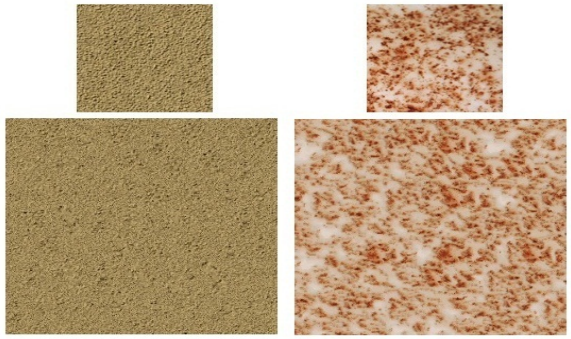


Fig. 18: *Synthesis results for non-structural texture: Sands (left) and Marble (right).*

undermine the quality of the result texture. While, our method uses drawing mesh sketch, a very simple interaction provided by users, to replace the strict detection of texture splicing, and avoids a long pre-computation.

4.3 Limitation

The texture mesh used in our method constrains the locations where the texture elements should be arranged, and for each cell in the mesh, only one element is put. Thus, our method cannot be applied to the texture such as falling leaves, fruits piles because of the overlap between the elements. Another limitation is that for the target texture having curved arranged elements,the quality of synthesis will decrease. This is because the curves are approximated by the segments in both the mesh sketch and the texture mesh, and the texture elements are arranged along the segments, see in Figure 19.

5 Conclusions and Future Work

We present a novel interactive texture design and synthesis method in this paper. Besides example textures, only mesh sketches drawn by hand or detected from the example textures are needed as inputs, which avoids the complex inputs from users. The texture mesh with desired size and

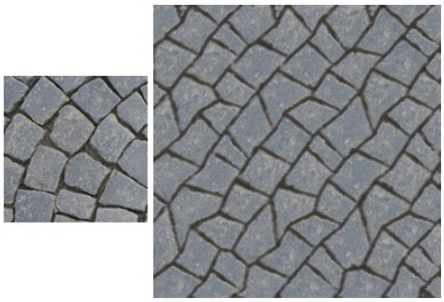


Fig. 19: *The example texture (left) with curved arrangement are compared with the result of our method (right).*

pattern is synthesized from the mesh sketch. Distance and orientation features related to the texture mesh are introduced into the mesh rasterization for reducing the matching cost for pixels. Our method archives interactive design and synthesis because of parallel computation of mesh rasterization.

One possible direction of future work is to extend our method from one example texture and one mesh sketch input to the multiple example textures and mesh sketches, which will bring more variations in texture design and synthesis. Another future work is to alleviate the limitation of curve arrangement discussed above. Curved edges will be considered in the synthesis of texture meshes. After we get the refined texture mesh, the segment edges should be replaced with curve edges, which will introduce the discontinuity between two edges. Bezier or B-spline may be used to smooth the edge pairs.

Acknowledgements This work was supported by the National Natural Science Foundation of China through Projects 61272349, 61190121 and 61190125, by the Macao Science and Technology Development Fund through Project 043/2009/A2, by the National High Technology Research and Development Program of China through 863 Program NO. 2013AA01A604.

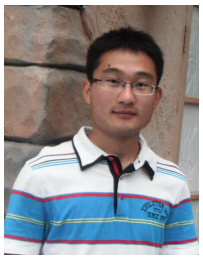
References

1. Wei L Y, Lefebvre S, Kwatra V, Turk G. State of the art in example-based texture synthesis. In: Eurographics '09 State of the Art Report (STARs). 2009, 93–117
2. Wu Q, Yu Y Z. Feature matching and deformation for texture synthesis. ACM Transcations on Graphics, 2004, 23(3): 362–365
3. Liu Y X, Lin W C, Hays J H. Near-regular texture analysis and manipulation. ACM Transcations on Graphics, 2004, 23(3): 368–376
4. Zhang J, Zhou K, Velho L, Guo B, Shum H. Synthesis of progressively-variant textures on arbitrary surfaces. ACM Transcations on Graphics, 2003, 22(3): 295–302
5. Matusik W, Zwicker M, Durand F. Texture design using a simplicial complex of morphable textures. ACM Transactions on Graphics, 2005, 24(3): 787–794
6. Ray N, Levy B, Wang H, Turk G, Vallet B. Material space texturing. Computer Graphics Forum, 2009, 28(6): 1659–1669
7. Ruiters R, Schnabel R, Klein R. Patch-based texture interpolation. Computer Graphics Forum, 2010, 29: 1421–1429
8. Narain R, Kwatra V, Kim T, Lee H, Carlson M, Lin M C. Feature-guided dynamic texture synthesis on continuous flows. In: Eurographics Symposium on Rendering/Eurographics Workshop on Rendering Techniques. 2007, 361–370
9. Ramanarayanan G, Bala K. Constrained texture synthesis via energy minimization. IEEE Transactions on Visualization and Computer Graphics, 2007, 13(1): 167–178
10. Liu J PY. W, Xue S, Tong X, Kang S B, Guo B N. Texture splicing. Computer Graphics Forum, 2009, 28(7): 1907–1915
11. Risser E, Han C, Dahyot R, Grinspun E. Synthesizing structured image hybrids. ACM Transactions on Graphics, 2010, 29(4): 85:1–85:6
12. Kim V G, Lipman Y, Funkhouser T. Symmetry-guided texture synthesis and manipulation. ACM Transactions on Graphics, 2012, 31(3): 22–36
13. Zhou H, Sun J, Turk G, Rehg J M. Terrain synthesis from digital elevation models. IEEE Transactions on Visualization and Computer Graphics, 2007, 13(4): 834–848
14. Kwatra V, Schodl A, Essa I, Turk G, Bobick A. Graphcut textures : Image and video synthesis using graph cuts. ACM Transactions on Graphics, 2003, 22(2): 277–286
15. Wei L Y, Han J W, Zhou K, Bao H J, Guo B N, Shum H Y. Inverse texture synthesis. ACM Transactions on Graphics, 2008, 22(3): 1–9
16. Rosenberger A, Cohen-Or D, Lischinski D. Layered shape synthesis: automatic generation of control maps for non-stationary textures. ACM Transactions on Graphics, 2009, 28(5): 22–30
17. Ijiri T, Mech R, Igarashi T, Millei G. An example-based procedural system for element arrangement. 2008, 27(2): 429–436
18. Xu K, Cohen-Or D, Ju T, Liu L G, Zhang H, Zhou S Z, Xiong Y S. Feature-aligned shape texturing. ACM Transactions on Graphics, 2009, 28(5): 1–7
19. Kwatra V, Essa I A, Bobick A F, Kwatra N. Texture optimization for example-based synthesis. ACM Transactions on Graphics, 2005, 24(3): 795–802
20. Ma C Y, Wei L Y, Tong X. Discrete element textures. ACM Transactions on Graphics, 2011, 30(4): 1–10
21. Cohen M F, Shade J, Hiller S, Deussen O. Wang tiles for image and texture generation. ACM Transactions on Graphics, 2003, 22(3): 287–295
22. Dong F, Lin H, Clapworthy G. Cutting and pasting irregularly shaped patches for texture synthesis. Computer Graphics Forum, 2005, 24(1): 17–26
23. Praun E, Finkelstein A, Hoppe H. Lapped textures. In: Proceeding of the 27th annual conference on Computer graphics and interactive techniques. 2000, 465–470
24. Zhou K, Huang X, Wang X, Tong Y Y, Desbrun M, Guo B N, Shum

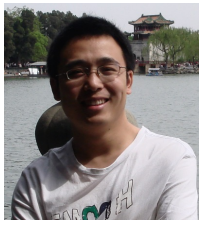
H Y. Mesh quilting for geometric texture synthesis. ACM Transactions on Graphics, 2006, 25(3): 690–697

25. Lasram A, Lefebvre S. Parallel patch-based texture synthesis. In: Proceeding of the 4th ACM SIGGRAPH/Eurographics conference on High-Performance Graphics. 2012, 115–124
26. Zelinka S, Garland M. Towards real-time texture synthesis with the jump map. In: Proceeding of Eurographics Symposium on Rendering/Eurographics Workshop on Rendering Techniques. 2002, 99–104
27. Liang L, Liu C, Xu Y Q, Guo B, Shum H. Real-time texture synthesis by patch-based sampling. ACM Transactions on Graphics, 2001, 20(3): 127–150
28. Lefebvre S, Hoppe H. Parallel controllable texture synthesis. ACM Transactions on Graphics, 2005, 24(3): 777–786
29. Dischler J M, Zara F. Real-time structured texture synthesis and editing using image-mesh analogies. The Visual Computer, 2006, 22(9-11): 926–935
30. Barla P, Breslav S, Thollot J, Sillion F, Markosian L. Stroke pattern analysis and synthesis. Computer Graphics Forum, 2006, 25(3): 690–697

Lili Wang received the Ph.D. degree from Beihang University, Beijing, China, where she works as an assistant professor and an associate professor at School of Computer Science and Engineering from 2005 and 2007. She is also a researcher at the State Key Laboratory of Virtual Reality Technology and Systems. Her interests include the real-time rendering, realistic rendering, global illumination, soft shadow, and texture synthesis.



Qinglin Qi, born in 1989, is Master Candidate in the Computer Science School and the State Key laboratory of Virtual Reality Technology and Systems, Beihang Universitry. His current research interests include computer graphics and texture synthesis.



Yi Chen, born in 1987, is Master graduated from the Computer Science School and the State Key laboratory of Virtual Reality Technology and Systems, Beihang Universitry. His main research interests include 3D graphics rendering, computer graphics and texture synthesis.

Wei Ke is a researcher and lecturer of Macao Polytechnic Institute. He received his PhD from School of Computer Science and Engineering, Beihang University. His research interests include programming languages, formal methods, tool support for object-oriented and component-based systems, and virtual reality applications. His recent research focuses on programming tools, programming environments and collaboration platform architectures.

Aimin Hao is a professor in the Computer Science School and the Associate Director of the State Key laboratory of Virtual Reality Technology and Systems at Beihang University. He got his BS, MS, and PhD in Computer science at Beihang University. His research interests are on virtual reality, computer simulation, computer graphics, geometric modeling, image processing, and computer vision.