A New Algorithm for Solid Texture Synthesis

Jia-Wei Chiou and Chuan-Kai Yang

National Taiwan University of Science and Technology, Taipei, 106, Taiwan, ROC

Abstract. Despite the tremendous rendering power offered by modern GPUs, real-time and photo-realistic rendering is still often out of reach of traditional polygonal-based rendering. Thanks to the invention of texture mapping, a scene with a moderate number of triangles could be readily and vividly rendered by nowadays popular and inexpensive graphics cards. However, as a desired texture often comes with a very limited size, the technique of texture synthesis, i.e., synthesizing a larger texture from an originally smaller texture, has become a popular research topic in recent years. Numerous techniques have been proposed to successfully synthesizing 2D textures in terms of quality and performance. 3D or solid texture synthesis, on the other hand, remains relatively unexplored due to its higher complexity. There are several types of existing algorithms for solid texture synthesis, and among them, the outstanding work by Jagnow et al. [1] opens a new door for solid texture synthesis of discrete particles; however, their work did not address two important issues, thus leaving room for further improvement. First, without the help of stereology, users need to explicitly provide the 3D shapes of target particles for synthesis, and this is especially true when oftentimes only a 2D texture image is available for solid texture synthesis. Second, the locations and orientations of the 3D particles are resolved by a *simulated annealing* method, which is intrinsically a non-deterministic approach, and thus the optimality is not always guaranteed. To solve the shape problem, we propose a simple algorithm that applies the idea of *visual hulls* to approximate the shapes of 3D particles when only a 2D image is given; to solve the location and orientation problem, we design a deterministic algorithm that could place these desired 3D particles in space more properly. Most importantly, there is no need for user's intervention for both algorithms. We have successfully implemented the proposed algorithm and the experimental results are also presented for comparisons with previous results and also for the proof of our concepts.

1 Introduction

One frequent problem encountered in texture mapping is that the texture image source often comes with a very limited size. As a result, how to generate a larger texture from a given smaller texture, has become one of the most important problems in the field of *texture synthesis*.

Textures can be two-dimensional (2D), three-dimensional (3D), and four or even higher-dimensional. Despite the fact that there have been numerous researches successfully performing texture synthesis, in terms of quality and efficiency, most of the works concentrate on 2D texture synthesis, while 3D texture, or solid texture synthesis, receives relatively less attention. The scarcity of related papers is mainly attributed to the much higher complexity involved in solid texture synthesis. Among the existing approaches for solid texture synthesis, the synthesis of discrete particles has attracted our attention, as it is an area that so far has been even less explored. The pioneering work in this specific direction, done by Jagnow et al. [1] especially caught our eyes for its great outcome, but it still leaves two important issues that have not been fully addressed. The first issue concerns the 3D shapes of target synthesized particles. According to the paper, these shapes could be derived by applying *stereology*, otherwise the provision of 3D particles is required. However, as often times only one 2D image is available for texture synthesis, the application of stereology may be difficult. Instead, we propose a simple algorithm that could approximately construct the shapes of desired 3D particles through the concept of *visual hull*, assuming the synthesized 3D particles are *iso-tropic*, i.e., bearing similar cross sections from every viewing direction. The second issue is regarding the placement of these 3D particles. In their paper, this issue is solved by a *simulated annealing* approach, where all the particles were initially put into the volume, then their locations or even orientations could be gradually adjusted to avoid collisions. Rather than using such a soft optimization technique, where the optimal solution can not always be guaranteed, we develop a simple algorithm that could deterministically and appropriately place the particles in the output texture volume.

The rest of the paper is organized as the following. Section 2 reviews some of the literature related to this study. Section 3 details how we synthesize solid texture of particles. Section 4 presents the experimental results produced by our system, while section 5 concludes our work and hints several potential future research directions.

2 Related Work

The concept of solid texture was first given by Gardner et al. [2], but the term *solid texture* was formally introduced by Peachy [3] and Perlin [4] in 1985. As pointed out in the introduction section, there have been numerous researches on texture synthesis. However, due to the involved complexity, only relatively few of them are on solid texture synthesis. In terms of completeness, the review done by Dischler et al. [5] does a good job, therefore we will only concentrate on the literature that is highly related to solid texture synthesis.

In terms of spectral analysis for solid texture synthesis, there exist several approaches, such as Ghazanfarpour et al. [6], [7], and Dischler et al. [8]. Our work, on the contrary, employs a spatial analysis approach. With regard to this direction, Chen et al. [9] introduced a 3D texture synthesis method that is based on texture turbulence and texture growing. Wei [10] proposed to synthesize solid texture from multiple 2D source. The basic idea follows exactly from his former paper [11], which in turn is an improved version of an earlier and similar paper by Heeger et. al [12]. Although both approaches operate in the spatial domain, they are not suitable for dealing with solid texture of particles, as the integrity of particles may get destroyed during the synthesis process.

To be able to synthesize solid texture of particles, Dischler et al. [13] proposed a method, operates in the spatial domain, to shape particles using the idea of *generalized cylinders*. Perhaps the most similar approach to ours is the one proposed by Jagnow et al [1]. In the paper, they first showed how to perform an accurate estimation of 3D particle distribution from a 2D image, so that later particles of different sizes can be generated and arranged accordingly. Next they demonstrated how a *simulated annealing* approach is used to solve the location arrangement problem when all synthesized particles are to be put within the target texture volume. Finally they addressed the issue of adding variant colors and details on the synthesized particles. Like [1], our method also operates in the spatial domain and our target solid texture is for discrete particles, but unlike their approach to depend on *stereology* or on the provision of 3D particles for later synthesis, we try to build particles solely from the given 2D image(s). Moreover, our algorithm for particles' location arrangement is also quite different from theirs.

3 3D Volume Texture Synthesis

We present the core materials of this work in this section. To ease the discussion, we hereby distinguish two terms: *local volume*, and *global volume*. The first term refers to the crude volume from which a single particle is synthesized, while the second term the target volume of solid texture where all the generated particles to be placed into. Given one or more 2D images, our task is to synthesize the global volume of solid texture of discrete particles, where the cross sections of these discrete particles will look like what appear in the input 2D images. Let us start with an system overview which explains what the major steps are, and how these steps are linked together.

3.1 Overview

Figure 1 depicts an overview of our system. First, we start with the segmentation of the given one of more 2D images, so that the associated information of the target particles can be analyzed and collected for later synthesis use. Second, particles with too small sizes are filtered to get rid of possible noise presented in the input image(s). Third, according to the contours extracted from the previous stage, we could synthesize a single particle by applying the *visual hull* algorithm. Fourth, according to the information collected from the input image(s), we could scale the sizes of a particle non-uniformly along each dimension, assign the color of a particle, and arrange a spatial location for a particle, so that the statistics shown in 3D will match what we observed in the given 2D images. The execution of the third and the fourth steps forms a loop, that is, particles are generated one by one until the desired number of particles or the specified overall *crowdedness* is reached.

3.2 Segmentation of 2D Input Images

The segmentation process extracts a set of 2D shapes from a given one of more 2D images. Currently this process is done manually for the following reasons. First, some particles may possess textures which make segmentation more difficult. Second, for the sake of robustness in the *visual hull* process, particles whose sizes are too small should be eliminated. Third, some particles which intersect with the image boundary should also be removed, so that they will not create sharp boundary during the visual hull process.

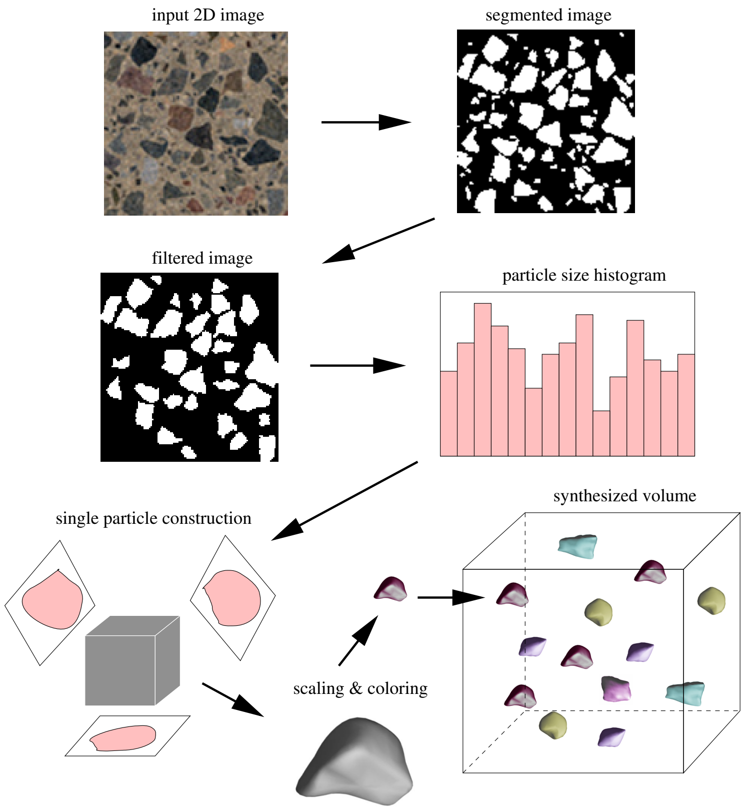

Fig. 1. An overview of the system used in this work

3.3 Histogram Analysis of Particle Attributes

The first information to be collected, after the previous segmentation stage, is the size of particles. We measure the height and width of the 2D bounding-box of a particle, and build the associated histogram. In addition, color information is collected in this stage as well. After the analysis of size and color information, the corresponding histograms, together with the 2D contours are stored so that the arrangement of locations and colors could be determined accordingly later on.

3.4 Synthesis of a Single Particle

The next step is to synthesize particles. We first show how to synthesize one particle, and we will explain how to adjust the sizes of a generated particle along different dimensions. To address this, Dischler et al. proposed in [13] to perform the synthesis of a particle through *generalized cylinders*, i.e., a base figure followed by a moving path. This technique, though proven to be effective in some cases, still shows orientation preference, and the modeling process itself is more complicated then our method to be proposed. With only one image given, we choose to make a seemingly bold assumption that target particle shape is *isotropic*; that is, the shape does not differ much when viewed for different orientations. We later will show how this assumption may be modified when more than one 2D images are given and when anisotropic solid texture is desired. The assumption comes from the observation that if particles of different sizes and different orientations are to be placed randomly without collisions with others, then the 2D contours that we found from one slice simply represent cross sections of these particles along arbitrary directions. This seems to be reasonable as the shapes of particle's cross sections in general do not show too much variation. We therefore randomly *select a subset of the 2D contours collected in the previous segmentation stage and treat them as the cross sections of a single particle from different view angles*, then use the idea of *visual hulls* [14,15,16] to reconstruct the 3D shape of a particle.

Although simple, there is still one implementation issue of visual hulls that is worth mentioning. In order to produce a smooth particle, usually at least a dozen of cross sections are needed for projection, and the result is the intersection of all these projections. However, imagine there is one 2D contour used for projection that is significantly smaller than others, then the size of the resulting particle will be reduced rapidly due to intersection. This essentially means that it is more difficult to control the size of generated particle if the involved 2D contours present non-negligible variations in terms of width or height. We therefore choose to "decouple" this factor from the synthesis of a single particle. More specifically, after recording the original size or bounding-box information of all the 2D contours for future use, we identify the largest width and largest height from these 2D bounding-boxes. Assume the larger one of the two is L, we then stretch all the 2D bounding-boxes into *square images* with the side length L. We will refer to this stretching process as a *normalization* process.

After the determination of L, the construction of a single particle is as follows. We start with a cubic volume whose side length is L, and the voxels of this local volume are initialized to be the color that we desire, as will be described later, a dozen of randomly chosen *normalized* 2D contours are then used to "carve" the 3D shape of a target particle from an arbitrary direction.

Note that after the synthesis is done, the bounding-box of the synthesized particle is measured again for a more accurate estimation and for later processing, as the newly formed particle may have a quite different bounding-box dimension from that of the initial local volume, due to the effect of multiple cross-section projections.

3.5 Size Assignment

The aforementioned procedure can be used to generate a single particle. The next step is to scale it non-uniformly along each dimension so that the overall size distribution of all particles could match what we observe from the given 2D image(s). One way is to apply the idea described in [1], where the size and density of particles are analyzed in detail. Here we resort to a simpler approach which in practice performs satisfactorily well. Recall that we have previously built the histogram information from the width and height information of the bounding-boxes of the extracted 2D contours, so each bounding-box in effect "pairs up" its width and height values. To respect the original distribution, now the size picking process has become a *hierarchical picking process*. At the first level, we randomly picked a $(width, height)$ pair from the histogram recorded previously, while each such pair is associated with a probability according to its frequency of appearance. The next step, happened at the second level, is to pick a $depth$ value from the chosen pair's associated distribution. Note that due to the bounding box pairing process, there may exist several sizes that have been paired with either the chosen width or height value, and from their occurring frequency the probability of each size being selected can then be determined, thus obtaining totally three values as a result. These three values are then used to scale the 3D particle constructed in the previous phase accordingly. Notice that there still exists the possibility that a newly generated particle with the chosen width, height and depth cannot be accommodated into the global volume due to the occupancy of particles produced earlier. To address this, we dynamically adjust the distribution so the probability of larger sizes will become smaller and smaller.

3.6 Color Assignment

The color assignment can be done similarly. Here we adopt the idea of Jagnow et al.'s idea in [1], where the average colors for 2D particles, as well as for the background pixels are first calculated. 2D particles and background pixels are then colored by their averaged colors. A residual image is formed by subtracting the averaged-color image from the original input image. During the color assignment process, pixel values on the residual image can then be used to perturb the previously assigned color of a background or non-background voxel.

3.7 Location Assignment

Once the size and color assignment of a particle are settled, the location assignment comes next. One possible solution for particle arrangement is to generate all particles at once, then place them within the global volume, just like Jagnow et al.'s approach [1]. There are, however, two drawbacks with their approach. First, sometimes it may not be easy to determine the number of particles to be placed into the global volume, given the fact that particles' sizes and orientations may vary significantly. Second and most importantly, as particles may have arbitrary shape, to find the best placement for all particles simultaneously is more difficult than the well known *bin packing* problem, which is already a *NP complete* problem. Therefore in [1], a *simulated annealing* approach is used instead to probe for a sub-optimal solution.

A potentially less optimized but more efficient approach is to generate and place a particle one by one, while at each time we always try to find the best spot for arranging a particle just produced. To expedite the process of searching for the best spot, we could associate each voxel with a value, which represents the maximal radius that a sphere centered at this voxel can have without touching any other non-background voxels. Essentially this value indicates the *spaciousness* around a given voxel. One way to speed up the calculation of spaciousness of each voxel is through the use of *Delauney tetrahedralization*, a 3D generalization of *Delauney triangulation*, which is closely related to *Voronoi diagram* [17,18,19].

After each voxel being assigned its spaciousness value, and according to this value, each voxel will be distributed to the corresponding queue of the same spaciousness. When a newly generated particle comes, we applied the policy of *best fit* borrowed from *operating systems*, to locate the queue with the best spaciousness, and then randomly pick a voxel location from the queue, for arranging the given particle, i.e., the particle's center will be put at the selected spot. Once a particle is added, the Delauney tetrahedralization is incrementally computed and the spaciousness of those affected voxels will be updated, i.e., deleted from their original queues and inserted to different queues.

Due to the limitation of time, we did not get the chance to implement the non-trivial 3D Delauney tetrahedralization algorithm, but instead, we employ a more brute-force like approach as follows. Just like what we mentioned in the previous implementation, we could build a number of queues where each queue stores the voxels with the same spaciousness value. To save storage and speedup computation, we make use of the observation on the bounding-box size. From the size histogram collected during the segmentation stage, assume the largest possible side length of a 3D bounding-box is D, it is easy to show that the largest spaciousness value to consider is $\sqrt{3}D/2$. This basically means we only need to build queues of spaciousness up to $\sqrt{3}D/2$. Also due to this limited size, initially all the voxels, except the boundary voxels of the global volume, are assigned to the queue of $\sqrt{3}D/2$ spaciousness, while the boundary voxels to the queues corresponding to their nearest distances to the boundary of the global volume. By applying the same reasoning as for the Delauney tetrahedralization, we know that once a new particle is placed at the best spot with the aforementioned *best fit* algorithm, dealing with only its boundary voxels is sufficient. Therefore we perform the potential update only for those voxels that are lying within the distance of $\sqrt{3}D/2$ of the boundary voxels on the newly added particle. Although such implementation incurs a slightly higher overhead, given the fact that added number of particles may not be huge, and in practice the value of D, compared with final global volume side length, is relatively small, the overall performance is quite acceptable. Note that, although this approach does not measure the spaciousness as accurate as the Delauney tetrahedralization approach does, its exhaustive checking for updates along the boundary of the added particle has made its precision not far from the ground truth; therefore the results it generates is satisfactorily well in practice.

The final issue is the estimation of the crowdedness. One natural definition of the 3D global volume crowdedness is the number of non-background voxels divided by the number of all voxels of the global volume. To simulate the distribution on the given 2D image(s), the crowdedness of the 3D global volume should be equal to the 2D image crowdedness, which is similarly defined as the number of non-background pixels divided by the number of all pixels. To achieve the same level of crowdedness, the synthesis of the new particles should not stop until the desired crowdedness is reached.

4 Performance Results

In this section, we demonstrate the results using our proposed algorithm. All the tests are performed on a Pentium IV 3.0GHz machine with 1GBytes memory running on the Windows XP operating system. We have synthesized three solid texture volumes from three different squared input images, respectively. The three input images are shown side by side in Figure 2. In general, the overall execution time ranges from 30 minutes to 2 hours.

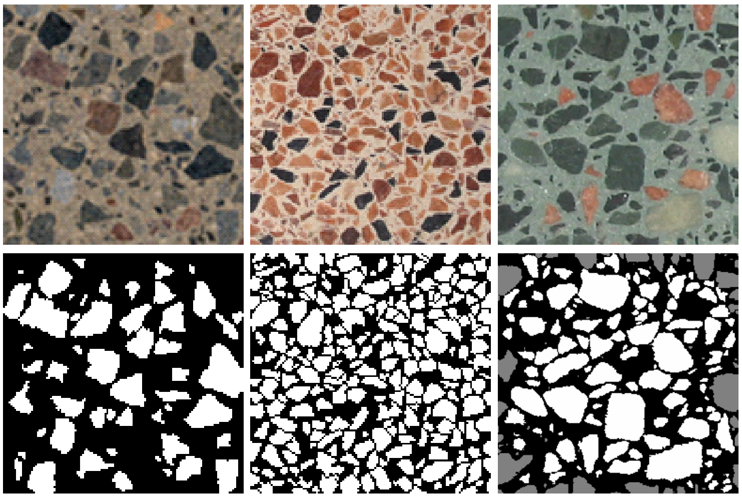

Fig. 2. Upper row: three input images used for the experiments, lower row: the corresponding 2D particle's masks extracted from the three input images, and used for later synthesis process. Note that the particles marked in gray in the third texture are not considered as they intersect with the image boundary.

Figure 3 shows the rendering results of two models with solid texture mapping, where the involved solid textures are synthesized from the first two input textures in Figure 2.

Figure 4 is the rendered image where compare multiple models using different solid textures generated using our system. Note that to facilitate rendering, we apply the transfer functions directly on the particles of synthesis, that is, the voxels of a particle get assigned both RGB colors and opacities immediately after the particle is generated. This is to ease the manipulation of transfer functions when different decisions have to be made on different particles. Alternatively, one could also design a more involved 3D interface to assign transfer functions to particles selectively.

5 Conclusion and Future Work

We propose a new algorithm for solid texture synthesis of particles. In this algorithm, each single particle could be constructed through the *visual hull* approach, and the locations of the generated particles are determined by employing a method that in spirit is similar to *Delauney tetrahedralization*. Performance results are shown to demonstrate the feasibility of our algorithm.

There are, however, still some limitations in our system, and how to go beyond these limitations deserves further study. First, our current assumption is that particle should be more or less *isotropic*, otherwise a *visual hull* approach for constructing the shape of a particle may fail. Remedies for this could be either requiring users to provide more 2D images corresponding to different viewing directions so that projections and involved images are carried out in pairs, or investigating more thoroughly on the given 2D

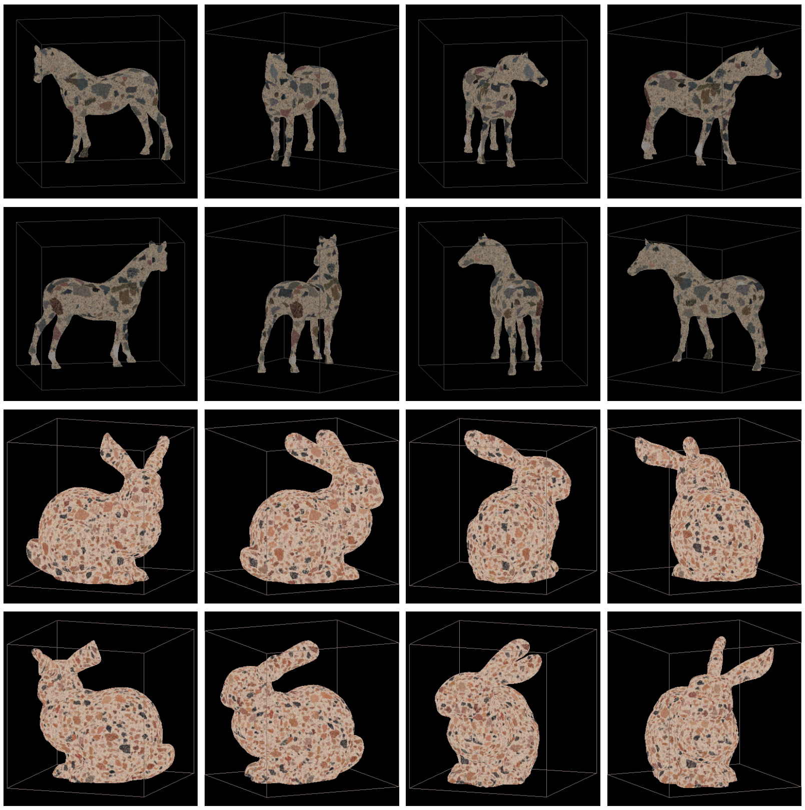

Fig. 3. Top two rows: a horse model rendered from the solid texture corresponding to the first input texture. Bottom two row: a bunny model rendered from the solid texture corresponding to the second input texture.

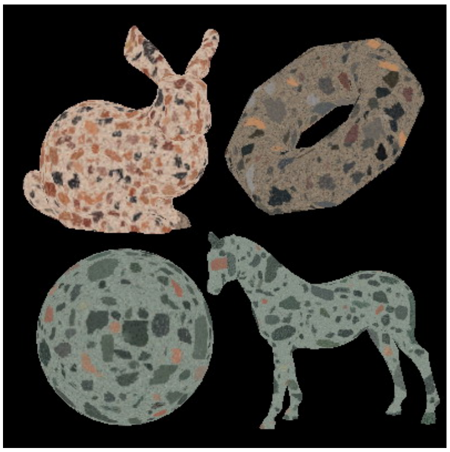

Fig. 4. Four solid-textured models using the texture volumes generated by our system

images for discovering the anisotropic patterns so that more intelligent projections could be performed. Second, it is evident that particles with concavity may not be modeled properly. One solution to this is to perform some deformation process, like what described in [13], and another possibility is to design a convenient interface so that users who are not satisfied with resulting shape synthesized by our system could easily perform their desired modification.

References

1. Jagnow, R., Dorsey, J., Rushmeier, H.: Stereological Techniques for Solid Textures. In: SIGGRAPH '2004. (2004) 329–335
2. Gardner, G.: Simulation of Natural Scene Using Textured Quadric Surfaces. In: SIGGRAPH '1984. (1984) 11–20
3. Peachey, D.: Solid Texturing on Complex Surfaces. In: SIGGRAPH '1985. (1985) 279–286
4. Perlin, K.: An Image Synthesizer. In: SIGGRAPH '1985. (1985) 287–296
5. Dischler, J., Ghazanfarpour, D.: A Survey of 3d Texturing. Computers & Graphics **25** (2001) 135–151
6. Ghazanfarpour, D., Dischler, J.: Spectral Analysis for Automatic 3-d Texture Generation. Computers & Graphics **19** (1995) 413–422
7. Ghazanfarpour, D., Dischler, J.: Generation of 3d Texture Using Multiple 2d Models Analysis. Computer Graphics Forum **15** (1996) 311–323
8. Dischler, J., Ghazanfarpour, D., Freydier, R.: Anisotropic Solid Texture Synthesis Using Orthogonal 2d Views. Computer Graphics Forum **17** (1998) 87–96
9. Chen, Y., Ip, H.H.: Texture Evolution: 3d Texture Synthesis from Single 2d Growable Texture Pattern. The Visual Computer **20** (2004) 650–664
10. Wei, L.: Texture Synthesis from Multiple Sources. In: SIGGRAPH 2003 Sketches & Applications. (2003)
11. Wei, L., Levoy, M.: Fast Texture Synthesis Using Tree-Structured Vector Quantization. In: SIGGRAPH '2000. (2000) 479–488
12. Heeger, D.J., Bergen, J.R.: Pyramid-Based Texture Analysis/Synthesis. In: SIGGRAPH '1995. (1995) 229–238
13. Dischler, J., Ghazanfarpour, D.: Interactive Image-Based Modeling of Macrostructured Textures. Computers & Graphics **19** (1999) 66–74
14. Laurentini, A.: The Visual Hull Concept for Silhouette-Based Image Understanding. IEEE Transactions on Pattern Analysis and Machine Intelligence **16** (1994) 150–162
15. Petitjean, S.: A Computational Geometric Approach to Visual Hulls. International Journal of Computational Geometry & Applications **8** (1998) 407–436
16. Laurentini, A.: The Visual Hull of Curved Objects. In: ICCV '1999. (1999) 356–361
17. Voronoi, G.M.: Nouvelles Applications des Paramètres Continus à la Théorie des Formes Quadratiques. Premier Mémoire: Sur Quelques Propriétés des Formes Quadratiques Postives Parfaites. J. Reine Angew Math. **133** (1907) 97–178
18. Voronoi, G.M.: Nouvelles Applications des Paramètres Continus à la Théorie des Formes Quadratiques. Deuxième Mémoire: Recherches sur les Parallélloèdres Primitifs. J. Reine Angew Math. **134** (1908) 198–287
19. Berg, M.D., Kreveld, M.V., Overmars, M., Schwarzkopf, O.: Computational Geometry, Algorithms and Applications. second edn. Springer (2000)