Synthesizing Solid Particle Textures via a Visual-Hull Algorithm

Hsing-Ching Chang, Chuan-Kai Yang, Jia-Wei Chiou, and Shih-Hsien Liu

Department of Information Management,

National Taiwan University of Science and Technology,

43, Keelung Road, Section 4, Taipei, 10607, Taiwan

Abstract

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. opens a new door for solid texture synthesis of discrete particles; however, their work leaves two important issues unaddressed First, without the help of stereology, users need to explicitly provide the 3D shapes of target particles for 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 can place these desired 3D particles in space more properly. Additionally we also propose a method to further couple the color and size information of particles to achieve an even better resemblance to the 2D image.

Key words: Texture Synthesis, Solid Texture, Isotropic Texture, Visual Hull, Delauney

1 Introduction

Due to the advances of technologies, modern GPUs have often surpassed CPUs in terms of graphics computing power. For example, today GPUs capable of rendering dozens of millions of smoothly-shaded triangles per second are common. Unfortunately, it is well-known that to achieve photorealism in real time, such rendering power alone is still not enough. One solution to this is to employ a clever idea to *create photorealism from photorealism*, which is the very spirit of *texture mapping*. That is, instead of creating photorealistic images from scratch, we could make use of *real* images to help create the desired photorealism. However, one frequent problem encountered in texture mapping is that the source texture image often comes with a small size/low resolution. As a result, finding methods for generating 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. It is well known that higher dimensional texture mapping, such as 3D texture mapping, can be used to address the distortion problems that are associated with lower dimensional texture mapping, such as 2D texture mapping. One classical example is to perform texture mapping to a sphere, where 3D texture map-

Email address:

D9309104@mail.ntust.edu.tw,ckyang@cs.ntust.edu.tw,

M9309004@mail.ntust.edu,A9515004@mail.ntust.edu.tw (Hsing-Ching Chang, Chuan-Kai Yang, Jia-Wei Chiou, and Shih-Hsien Liu).

ping is usually preferred to its 2D counterpart. Despite the fact that there have been numerous research methodologies that can be used to successfully perform 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, and 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. Additionally, we have further coupled the color and size information of particles to achieve even better results than all existing approaches.

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. The ways to generate *solid texture*, according to Dischler et al. [5], can be classified into three main categories: *procedural solid texturing*, *analytical solid texturing* and *physical simulation*, and among them, the second category is what this work belongs to. 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 at least three different approaches. Ghazanfarpour et al. [6] proposed an automatic solid texture generation algorithm that extracts the spectral information of a given 2D image, and uses it as a basic function, together with a band-pass filtered noise function for perturbation, to synthesize the output solid texture. An anti-aliasing method is also introduced in the paper. They later proposed another method [7] that iteratively modifies a volume initially containing only noise to the volume of desired solid texture volume, by altering the volume content in the spectral domain according to given images from the X, Y and Z axes directions, respectively, while the results

from all directions are averaged and sent for the next iteration until convergence is reached. As the convergence is sometimes difficult to control, Dischler et al. [8] came up with an improved algorithm, which tries to match the target solid texture volume with given 2D images both spatially and spectrally, and the corresponding end results converge much faster. For spectral alignment, assuming the input texture is *stationary*, they extract its sub-images and calculate the main features of the input image in the Fourier domain for coercing similarity. For spatial concordance, they perform histogram equalization adjustments to achieve resemblance.

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. Texture turbulence means that each frame in the target volume is merely a slight perturbation of the initial frame, i.e., the given 2D image. Such a perturbation could help to maintain some global structures that may be presented in the original 2D texture. The problem with texture perturbation is the constancy of the center position on each frame along the Z axis. Therefore, they add the texture growing process so that the center position could be modified on each frame by following a winded path extracted from a 2D turbulence map given by users. Wei [10] proposed to synthesize solid texture from multiple 2D source. The basic idea follows exactly from his former paper [11] (an improved version of an earlier and similar paper by Heeger et. al [12]), where texture synthesis is determined by comparisons of pixel neighborhoods and can be expedited by a tree-structured vector quantization, except that different neighborhood comparisons are performed for different viewing directions. 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*. They also described ways to deform and distribute particles in a controllable manner. 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 a *simulated annealing* approach is applied 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. Moreover, our algorithm for particles' location arrangement is also quite different from theirs.

More recently, Kopf et al. [14] proposed an algorithm that integrates the techniques of *non-parametric texture synthesis* and *global histogram matching* to perform solid texture synthesis. Their algorithm starts from a volume with random values, and then iteratively refines the value of each voxel through matching its neighborhoods' statistics with that of the input image. More specifically, to accommodate the assumption that the input image may be an arbitrary slice through the target volume, the matching process associates three orthogonal and axis-aligned slices with each voxel. Each of the three slices represents a neighborhood that is used for locating the pixel that is with the most similar neighborhood in the input image, and then the voxel value is a weighted average of the values from the three matched pixels. This algorithm, though claimed to be a general solid texture synthesis scheme, i.e., it could deal with textures of both particles or non-particles, suffered from generat-

ing blurry results caused by the averaging operation. A *feature map* is resorted to solve this issue but a success still cannot always be guaranteed. Moreover, by only taking into account three slicing directions, the presented textures may become unwantedly distorted patterns compared with the original input image when applied on some 3D meshes. Takayama et al. [15] generalized the idea of *patch-based* 2D texture synthesis to 3D using *tetrahedral meshes*, user-defined tensor fields are used to guide the orientations of the synthesized textures. However, it is more suitable for the synthesis of *anisotropic* textures, which are different from the target textures of ours.

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 corresponding 2D images. Let us start with a system overview which explains what the major steps are, and how these steps are linked together.

3.1 Overview

Fig. 1 depicts an overview of our system. First, we start with the segmentation of the given 2D image, so that the associated information of the target particles

can be analyzed and collected for later synthesis use. Note that particles with too small sizes are filtered to get rid of possible noise presented in the input image. Second, we build a 2D histogram of the input image according to the areas of particles. Third, by making use of the contours extracted from the previous stage, we synthesize a single particle through the *visual hull* algorithm. Finally, according to the information collected from the input image, we scale the sizes of a particle non-uniformly along each dimension, assign a color for it, and arrange a spatial location for it, so that the statistics shown in 3D will match what we observed in the given 2D image. The execution of the last two steps forms a loop, that is, particles are generated one by one until the 3D histogram equals to the 2D histogram or the specified overall *crowdedness* is reached. We explain each of these steps in more detail in the ensuing subsections.

3.2 Segmentation of the 2D input image

The segmentation of the given 2D image into a set of 2D shapes by the ensuing procedures. We first perform the segmentation manually through *Photoshop* by marking the contours that define all the particles in the 2D image. We opt for manual segmentation because input images are often with a very low resolution, e.g., 128×128, thus making potential serious *aliasing* effects in later processing stage. During the segmentation process, we could enhance the extracted particle contours by fitting them with *spline* curves. Second, to get rid of noise or other undesired artifacts, particles whose 2D areas smaller than a threshold are filtered. Third, each extracted 2D contour, together with its bounding-box dimension and color, are stored for later analysis and synthesis stages. After the non-background pixels are identified, all the remaining pixels are treated as background pixels, and

their colors are averaged into one color to be used for coloring the background voxels during later synthesis stage.

3.3 Histogram analysis of particle attributes

The first information to be collected, after the previous segmentation stage, is the size of particles. For each particle being segmented from the input image, we first measure its area, in terms of pixels. A 2D area histogram can thus be built, and is divided into 10 bins in total. In addition, we also need to record the height and width of the 2D bounding-box of each particle, as such information will be used during later particle synthesis stage. One might argue that due to various orientations that a particle may possess, this bounding-box measurement could become very imprecise. A good example to support such an argument could be constructed as follows. Imagine there is a circle with its center co-locating with the origin of a coordinate system. Let us elongate it along the X-axis direction to make it an ellipse, and measure its bounding-box, in terms of width and height, along the axis-aligned directions, as shown in Fig. 2(a). Now we could rotate it 45 degrees counterclockwise, and measure its axis-aligned bounding-box again, as shown in Fig. 2(b). Apparently the former will show a tighter bounding-box than the latter. However, as far as our implementation is concerned, *currently we resort to the normal axis-aligned bounding-box finding process*, i.e., we do not take into account the orientation issues, as this does not lead to any problematical results, not to mention that we will take extra procedures to probe for an accurate estimation of particle placement, and thus could potentially counteract this side effect.

Notice that if a more accurate estimation of particle size, or equivalently, a tighter bounding-box, is desired, one could proceed according to following method. For a

given 2D contour, each time we could rotate 1 degree and measure its corresponding axis-aligned 2D bounding-box, and eventually find out the angle at which the 2D contour assumes the tightest bounding-box. Here the comparison metric could be the sum of the width and height for a particular bounding-box. Other metrics are also possible, such as counting the percentage of pixels occupied by a 2D contour, divided by the bounding-box area. We must point out that even *without this bounding-box tightening process, our current synthesis process could still offer satisfactory results*, especially with notable orientation diversity. In other words, a tighter bounding-box may benefit the estimation of particles' sizes, but at the same time lead to a more uniform or axis-aligned orientation distribution, thus limiting the orientation variety.

During the particle synthesis process, the 3D histogram distribution of particles, in terms of their volumes, should match with the 2D histogram to achieve the desired resemblance. In addition to size information, color information is collected in the segmentation stage as well, and all such information, 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 one by one. We first show how to synthesize one particle, and then we will explain how to adjust the sizes of the generated particle along different dimensions. We will describe how to place the generated particles in the global volume later on. Regarding single particle's synthesis, Dischler et al. in [13] proposed 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 than our method to be proposed. With only one 2D image given, we choose to make a seemingly bold assumption that the target particle shape is *isotropic*; that is, the shape does not differ much when viewed from different orientations. We later will discuss 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 particles' 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*, and then use the idea of *visual hulls* [16–18] to reconstruct the 3D shape of a particle.

Although simple, there are still two implementation issues of visual hulls that are worth mentioning. First, to simplify the process, we only apply parallel projections. It should be understandable as we may have anyway used the cross sections from different particles to approximate the shape of one particle, so there is not much point of applying perspective projections to further increase the modeling precision. Second, to produce a particle, usually a number 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 a 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.

After the determination of L, the construction of a single particle is ready to be performed. 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, and a dozen of randomly chosen *normalized* 2D contours are then used to "carve" the 3D shape of a target particle from several arbitrary directions. We implement this modeling process by first aligning the center of the cubic local volume with the center of the 2D image, whose content is the 2D contour to be used in the visual hull process. And both centers are also co-located with the origin of the coordinate system, while both the X and Y axes of the image and all the axes of the cubic local volume are also aligned with the coordinate system. To simulate the effect of having a given 2D contour as the cross section, we randomly rotate the image along the X, Y and Z axes by an arbitrary angle while keeping the local volume intact, project all the voxels of the local volume onto the rotated image, and assign the background color to the voxels falling outside of the 2D contour to indicate their non-existence after the intersection.

Through numerous experiments, we have found that the number of cross sections is better to be set around 3 to 5, so that the produced particles could present desired shape variety, without all becoming approximately rounded. In our implementation, we chose 3 cross sections as they are good enough for creating satisfactory results for all input images. 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 is 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. One way is to apply the idea described in [1], where the sizes of particles are analyzed in detail. Here we resort to a simpler approach which in practice performs satisfactorily well. The fundamental issue boils down to finding a 3D counterpart of a 2D bounding-box, so that a non-uniform 3D scaling could be applied to resize the particle just synthesized, and a straightforward answer is a 3D bounding-box. Recall that we have previously collected 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 2D particle size distribution, now we turn the size picking process into a *hierarchical picking process* by the following approach, as shown in Fig. 4.

At the first level, we randomly picked a $(width, height)$ pair recorded previously, while each such pair is associated with a probability according to its frequency of appearance. The next step, applied 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

such 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. For example, in this figure, imagine we randomly select $(60, 50)$ as the initial $(width, height)$ pair. Now according to the existing 2D bounding-box sizes, there are four sizes, namely $(60, 50)$, $(60, 50)$, $(50, 70)$, and $(60, 80)$, are available for setting the desired $depth$ value, as they overlap with the initial $60, 50)$ pair with at least one value (width or height). The remaining (non-overlapped) value, marked in underlines, are the potential candidates for the depth value; that is, the depth value could be 60, 50, 70, or 80, thus making the altogether $(width, height, depth)$ a completely determined 3D bounding-box size for a particle to be created.

3.6 Color assignment

Rather than applying a histogram-based approach for color assignment, we have come up with a more successful approach than all existing approaches. Recall 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. However, it often occurs that a pixel value on the residual image, originally belonging to one particle with a certain color, could be perturbed onto another particle with a different color, thus causing undesired disturbance. An example of such is shown in Fig. 11(a), where one could see that some dark brown particles get accidentally brightened

during the perturbation process. To have a better result, we address this issue by resorting to a finer *granularity* on distinguishing particles one by one, so that a particle's size and its color perturbation could be correlated more properly. More specifically, we generate the residual image for each and every particle, so that at run time, as a $(width, height)$ pair is chosen, the corresponding residual image is retrieved for adding random noise onto the synthesized 3D particle in a manner similar to its 2D counterpart. In case there exists multiple such pairs, we choose an arbitrary one from it and its associated residual image accordingly. All the background voxels, treated as a special type of (potentially disconnected) particle as a whole, are dealt with in a similar fashion. One should note that in the case where there exists specific relationship between particle size and color, or even noise, our approach will definitely provide better quality than that of others.

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, and 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 could vary significantly. Second and most importantly, as a particle can have an 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 particles 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 can 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. Basically 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* [19–21]. It can be shown that given 3D points located in the 3D space, we could first find the Delauney tetrahedralization of these points, and for each tetrahedron in this tetrahedralization, we then determine its *circumsphere*, i.e., the sphere that passes the four vertices of this tetrahedron. The *circumcenter* and *circumradius* of this sphere can also be decided accordingly. Assuming the circumcenter co-locates with the center of a voxel, the circumradius is exactly the spaciousness value that we want to assign to the voxel, otherwise the circumcenter could be rounded to its nearest voxel, and the corresponding radius is deducted by one to take care of the roundoff error. Note that with this tetrahedralization approach, only the voxels that are centers for some circumspheres are associated with values while others are not. This causes no problem as placing particles at the voxels other than those circumcenter positions will not yield better results. Therefore we can simply assign the spaciousness value to those voxels that are not circumcenters. It is widely known that to build the 3D Voronoi diagram for N points requires $\Theta(N^2)$ time complexity in the worst case, however, there are at most n^3 voxels in the global volume if n is the side length, thus making the total time complexity of $\Theta(n^6)$ in the worst case. Moreover, a particle is a group of points, which seem to further complicate the whole process. Luckily, it does not take long for us to figure out that we need to run the Delauney

tetrahedralization only for those voxels on the boundary of a particle, thus greatly reducing the involved complexity.

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. Fig. 5 demonstrates such a location assignment process, where a voxel is denoted by $< r, *\alpha >$. Here r represents the spaciousness value, while $*\alpha$ is a pointer to a node that could be found in the corresponding queue of a particular spaciousness. Note that a node is consisted of the coordinate information of the associated voxel, denoted as (x, y, z), and two pointers, denoted as $*p(revious)$ and $*n(ext)$, for being used in the linked-list structure. 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.

For obtaining the spaciousness value for each voxel, we have implemented two approaches. The first, and more brute-force like approach, which will be referred to as *brute force* hereafter, works as follows. Just like what we mentioned in the previous implementation, we built 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 voxels near the boundary of the global volume, are assigned to the queue of $\sqrt{3}D/2$ spaciousness, while the remaining 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 second approach, which will be referred to as *discrete Voronoi* hereafter, is to build a *discrete Voronoi* diagram by observing the fact that all coordinates of the voxels can be treated as integers, thus greatly simplifying the implementation complexity. Fig. 6 lists the pseudo code, which is executed each time after a particle is newly generated and to be placed into the global volume. In this pseudo code, the function of $update_spaciousness(v, s)$ updates the spaciousness value of voxel v to be s, while $update_linked_list(v)$ updates the linked-list associated with voxel v, as shown in Fig. 5, after voxel $v's$ spaciousness value gets changed. The basic idea is to initially set all the spaciousness values for every voxel on the newly created particle to be zero, and then gradually move outwardly through their adjacent (26) neighbors to compare the old spaciousness values with the existing ones and

perform updates only when necessary. Those voxels whose spaciousness values get updated are treated as seeds for enclosing more voxels for potential further updates afterward.

Fig. 7 gives a 2D example to further demonstrate the basic idea behind the *discrete Voronoi* algorithm that we employed. In Fig. 7(a), we assume that there are initially two particles, as big as one voxel, and marked by black. The spaciousness values for all voxels are therefore defined in such a way that the farther it is a voxel to the two particles, the larger spaciousness value it gets assigned. Fig. 7(b) shows the immediate spaciousness status just after another new particle, marked by purple, is inserted. Fig. 7(c) demonstrates how the spaciousness values of the neighboring voxels for the purple particle get affected in the next stage. And the final resulting spaciousness values for all the voxels are shown in Fig. 7(d).

3.8 Histogram Matching and Estimation of Crowdedness

After describing each step in detail, we are now ready to illustrate the whole process of solid texture synthesis. As the goal is to generate a 3D solid texture which is similar to the 2D input image, we opt for a *histogram matching* approach. We start by synthesizing particles following the order from the bin corresponding to the largest area to the bin corresponding to the smallest area. This is due to the fact that placing small particles first could easily make the vacant space fragmented, thus greatly lowering the chances of accommodating large particles to be placed later. For each bin, which represents 2D particles whose areas fall between a specific range, we randomly pick a $(width, height)$ pair and perform the resizing and placement processes. However, there still exists a 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. In that case, as the same area could be obtained from different combinations of width and height, we randomly pick the next possible $(width, height)$ pair for a re-trial. If all the combinations belonging to the current bin have been tried, we simply give up and go for the next bin for ensuing synthesis process. Note that according to all our testing inputs, such a case does happen, but with a relatively low frequency, and therefore does not offset the resemblance of both histograms very much.

The final issue is regarding the stopping criteria, that is, when the whole solid texture synthesis should stop. As just mentioned previously, one straightforward solution is to perform the synthesis process bin after bin, according to the referred 2D histogram, until all bins have been examined. Another option is to estimate the *crowdedness* of the 3D global volume, and as long as the crowdedness of 3D synthesized texture is more or less equal to that of the 2D image, the synthesis could terminate. One natural definition of 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, 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 new particles should not stop until the desired crowdedness is reached. Currently we adopt both approaches, i.e., as either one constraint is met, the synthesis process stops.

4 Performance results

In this section, we demonstrate the results using our proposed algorithm, and compared our results with that of others if applicable. 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 five solid texture volumes from five different 2D input images, respectively, and the five input images are shown in Fig. 8.

To show that our algorithm really does a good job on preserving the size distribution, the 2D area histogram of the input images, as well as the 3D volume histogram of the generated solid textures, are shown in Fig. 9 and Fig. 10, respectively. Except in few cases, two histograms demonstrate similar distributions, and the reasons for the minor difference have been explained in Section 3.8. Note that the Y axes for these two figures area are *Ratio* instead of *Area*, *Pixel Numbers*, *Volume* or *Voxel Number*, so that we could make uniform comparisons across input images and out solid textures with different dimensions.

To show that our simplified histogram approach works relatively better, Fig. 11 compares our results with that of others. It is to our notice that for the middle input texture shown in Fig. 8, Jagnow et al.'s approach seems to generate a solid texture with a higher percentage of larger particles, as shown in Fig. 11(a). To be honest, currently we do not have clues on the reasons that cause this, and if we may guess, we suspect that it may be due to the *simulated annealing* process. Another subtle difference is on the handling of color perturbation, as mentioned in Section 3.6. Additionally, Kopf et al.'s approach, produces solid textures with a blurry look, which, as mentioned in the Related Work section, is due to its averaging operation. Our proposed approach, on the contrary, offers not only more clean-cut shapes, but also better resemblance to the input images in terms of color/size distributions.

Table 1 compares the timing results of the two approaches for finding the spaciousness value for each voxel, as described in Section 3.7. As can be shown in this table, *Discrete Voronoi* offers performance as much as 4.5 times faster than the

Brute Force approach.

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.

Fig. 12 and Fig. 13 show the rendered images on multiple models using the five solid textures generated by our system. Note that to further demonstrate our success on synthesizing particle-like solid textures, as shown in Fig. 12(c), we intentionally cut away parts of the generated solid texture volumes to demonstrate the internal texture structures.

5 Conclusions and future Work

We propose a new algorithm for solid texture synthesis of particles. In this algorithm, each single particle is 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*. Particles' colors and sizes are also well correlated so that their relationships in the input image are preserved. 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 algorithm is only suitable for solid textures of particles, as it generates and places particles one by one. Second, our current assumption is that the desired synthesized 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 images for discovering the anisotropic patterns so that more intelligent projections could be performed. Third, 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. One interesting extension of this system is to add texture or finer details into particles, in addition to color, to create a "texture-in-texture" effect, as can be observed from some of the particles in the third texture of Fig. 8.

References

[1] R. Jagnow, J. Dorsey, and H. Rushmeier. Stereological Techniques for Solid Textures. In *SIGGRAPH '2004*, pages 329–335, 2004.

[2] G. Gardner. Simulation of Natural Scene Using Textured Quadric Surfaces. In *SIGGRAPH '1984*, pages 11–20, 1984.

[3] D. Peachey. Solid Texturing on Complex Surfaces. In *SIGGRAPH '1985*, pages 279–286, 1985.

[4] K. Perlin. An Image Synthesizer. In *SIGGRAPH '1985*, pages 287–296, 1985.

[5] J. Dischler and D. Ghazanfarpour. A Survey of 3d Texturing. *Computers & Graphics*, 25(1):135–151, 2001.

[6] D. Ghazanfarpour and J. Dischler. Spectral Analysis for Automatic 3-d Texture Generation. *Computers & Graphics*, 19(3):413–422, 1995.

[7] D. Ghazanfarpour and J. Dischler. Generation of 3d Texture Using Multiple 2d Models Analysis. *Computer Graphics Forum*, 15(3):311–323, 1996.

[8] J. Dischler, D. Ghazanfarpour, and R. Freydier. Anisotropic Solid Texture Synthesis Using Orthogonal 2d Views. *Computer Graphics Forum*, 17(3):87–96, 1998.

[9] Y. Chen and H. H.S. Ip. Texture Evolution: 3d Texture Synthesis from Single 2d Growable Texture Pattern. *The Visual Computer*, 20(10):650–664, 2004.

[10] L. Wei. Texture Synthesis from Multiple Sources. In *SIGGRAPH 2003 Sketches & Applications*, 2003.

[11] L. Wei and M. Levoy. Fast Texture Synthesis Using Tree-Structured Vector Quantization. In *SIGGRAPH '2000*, pages 479–488, 2000.

[12] D. J. Heeger and J. R. Bergen. Pyramid-Based Texture Analysis/Synthesis. In *SIGGRAPH '1995*, pages 229–238, 1995.

[13] J. Dischler and D. Ghazanfarpour. Interactive Image-Based Modeling of Macrostructured Textures. *Computers & Graphics*, 19(1):66–74, 1999.

[14] J. Kopf, C. Fu, D. Cohen-Or, O. Deussen, D. Lischinski, and T. Wong. Solid Texture Synthesis from 2d Exemplars. In *SIGGRAPH '2007*, 2007.

[15] K. Takayama, M. Okabe, T. Ijiri, and T. Igarashi. Lapped Solid Texture: Filling a model with anisotropic textures. In *SIGGRAPH '2008*, 2008.

[16] A. Laurentini. The Visual Hull Concept for Silhouette-Based Image Understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, 1994.

[17] S. Petitjean. A Computational Geometric Approach to Visual Hulls. *International Journal of Computational Geometry & Applications*, 8(4):407–436, 1998.

[18] A. Laurentini. The Visual Hull of Curved Objects. In *ICCV '1999*, pages 356–361, 1999.

[19] G. M. Voronoi. 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.*, 133:97–178, 1907.

[20] G. M. Voronoi. Nouvelles Applications des Paramètres Continus à la Théorie des Formes Quadratiques. Deuxième Mémoire: Recherches sur les Paralléllòedres Primitifs. *J. Reine Angew Math.*, 134:198–287, 1908.

[21] M. D. Berg, M. V. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer, second edition, 2000.

Table 1

Timing result comparisons of the two approaches for finding the spaciousness *value mentioned in Section 3.7 for five input textures. Numbers are in minutes.*

	Texture1	Texture2	Texture3	Texture4	Texture5
Brute Force	9	175	532	8	22
Discrete Voronoi	2	77	176	2	5

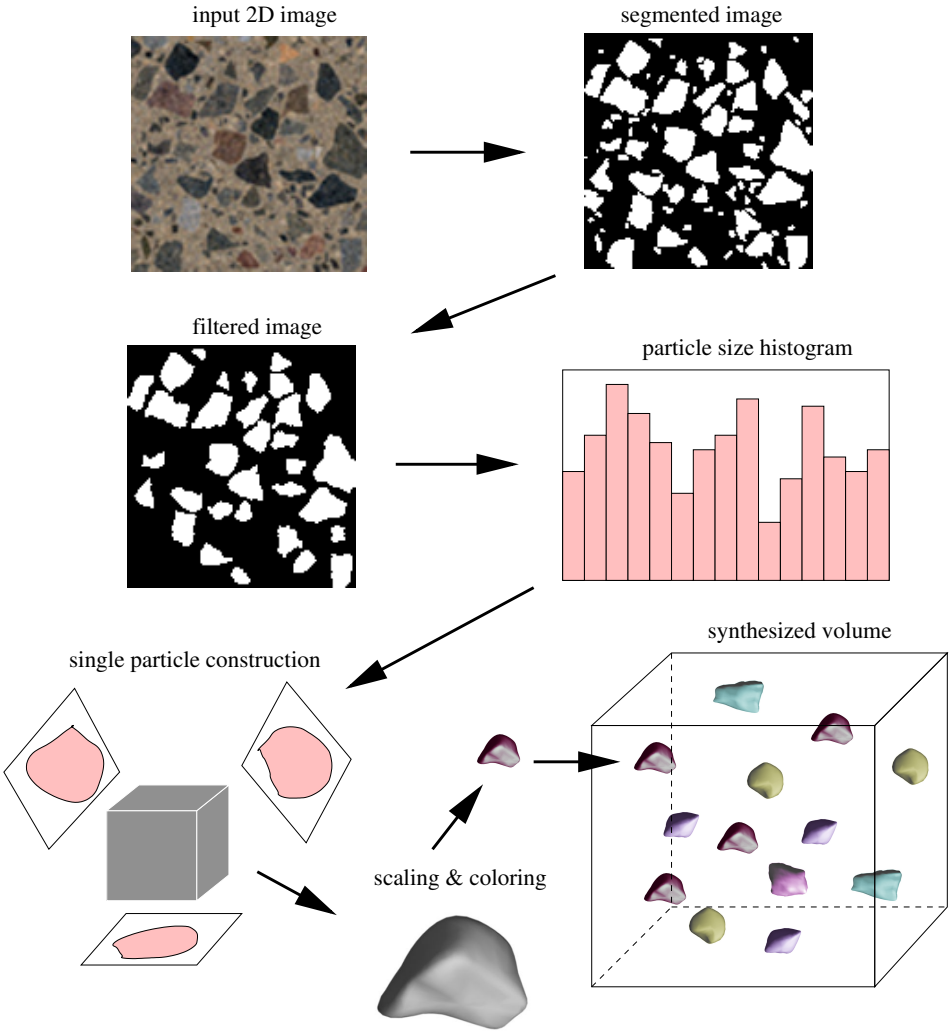
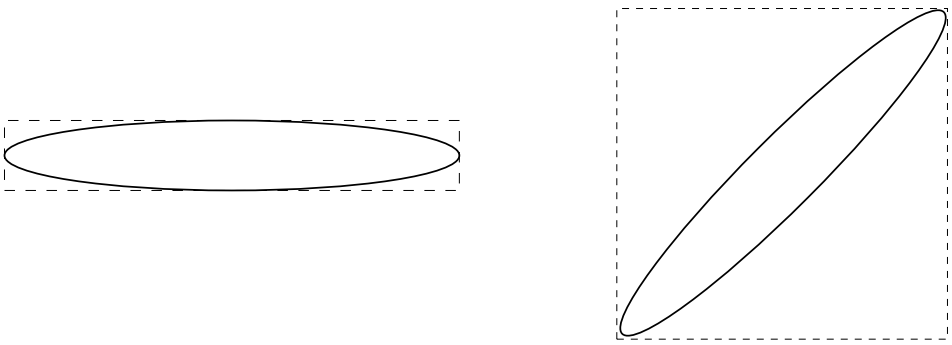


Fig. 1. An overview of the system used in this work.



(a) (b)

Fig. 2. Bounding-boxes (shown in dashed lines) for (a): an ellipse, and (b): its rotated counterpart.

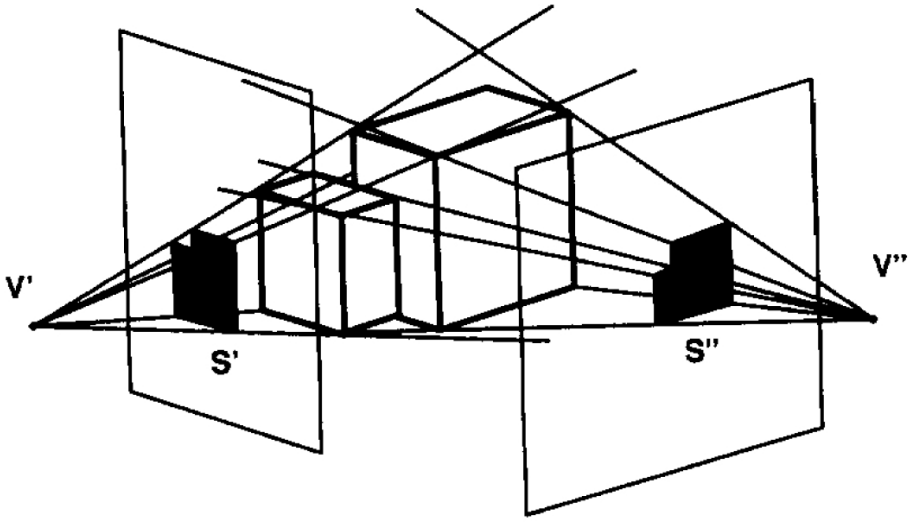


Fig. 3. The concept of *visual hull*, cited from [16], where the central shape is "carved" through two different views V' and V'' with two corresponding contours S' and S''.

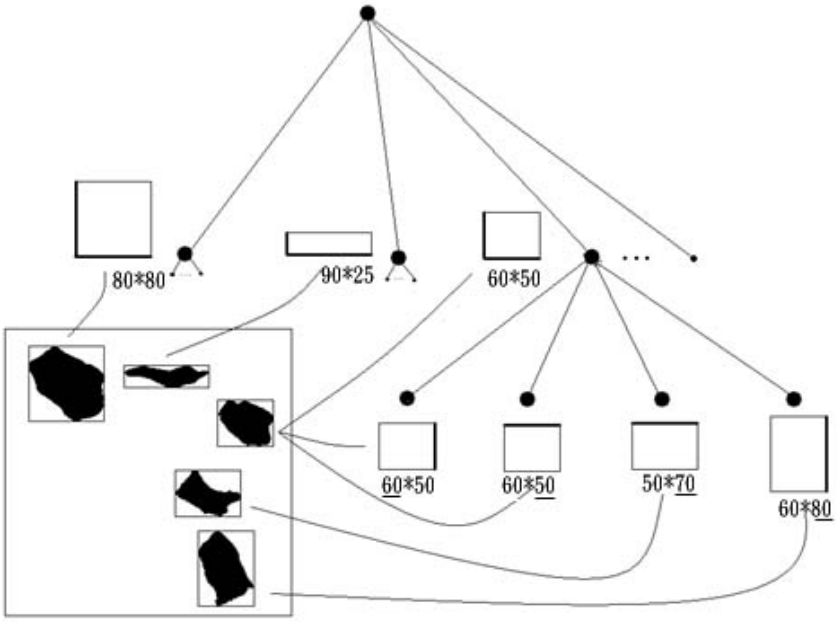


Fig. 4. The process of size assignment.

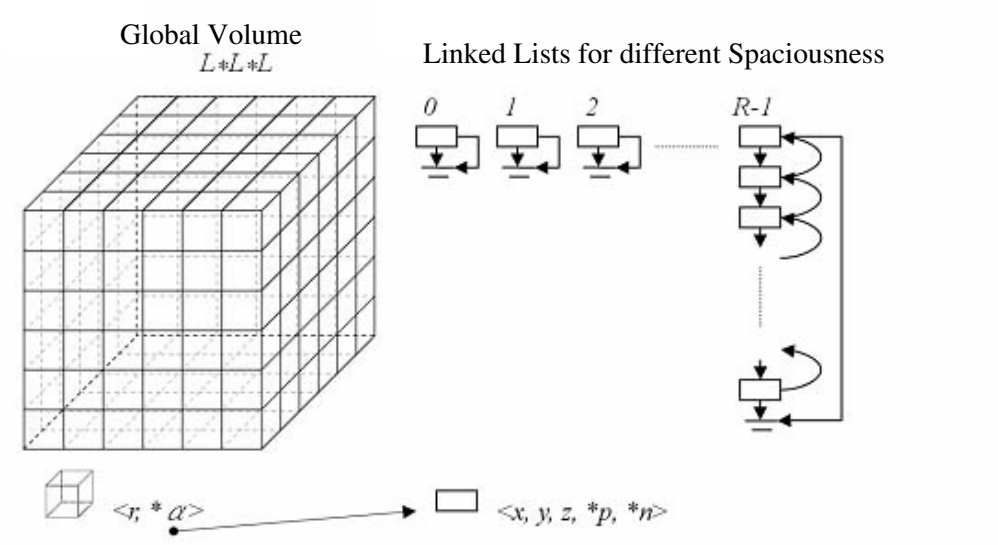


Fig. 5. The process of location assignment, where each voxel is associated with its information of spaciousness (r), coordinate (x, y, z), and pointers ($*p(revious), *n(ext)$), and is placed in the queue with the corresponding spaciousness.

```
//executed each time when a particle is generated and to be placed into the global volume

queue.empty()

for every voxel inside the newly inserted particle
     queue.enqueue(voxel)
     update_spaciousness(voxel, 0)
     update_linked_list(voxel)
end for

while(queue is not empty)
    voxel = queue.dequeue()
    for all 26 neighbors of voxel
        if(spaciousness(neighbor) > spaciousness(voxel) + 1)
            queue.enqueue(neighbor)
            update_spaciousness(neighbor, spaciousness(voxel) + 1)
            update_linked_list(neighbor)
        end if
    end for
end while
```

Fig. 6. *The pseudo code for updating the* spaciousness *values for the involved voxels each time after the insertion of a particle.*

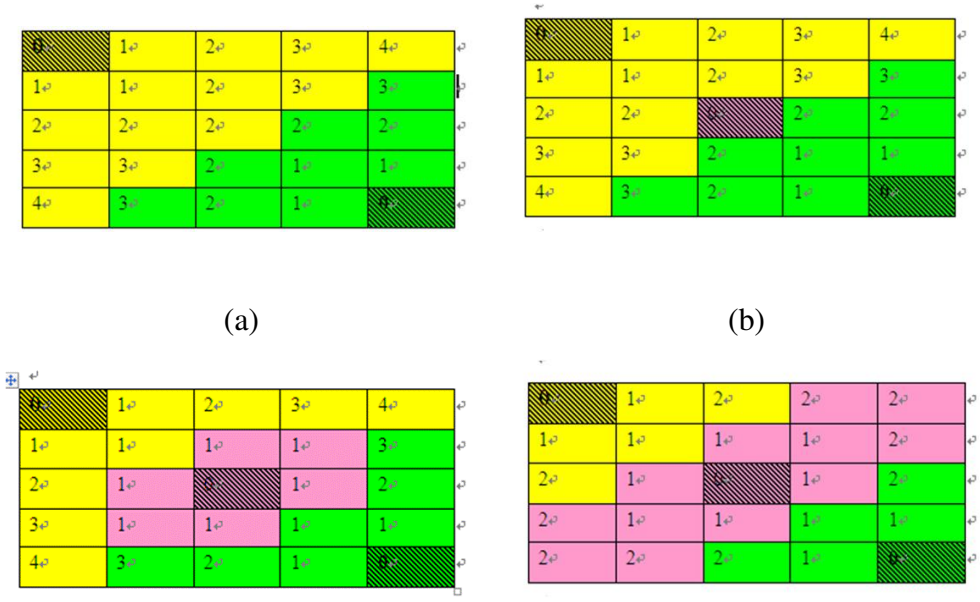


Fig. 7. A 2D example to demonstrate the spaciousness calculation process.

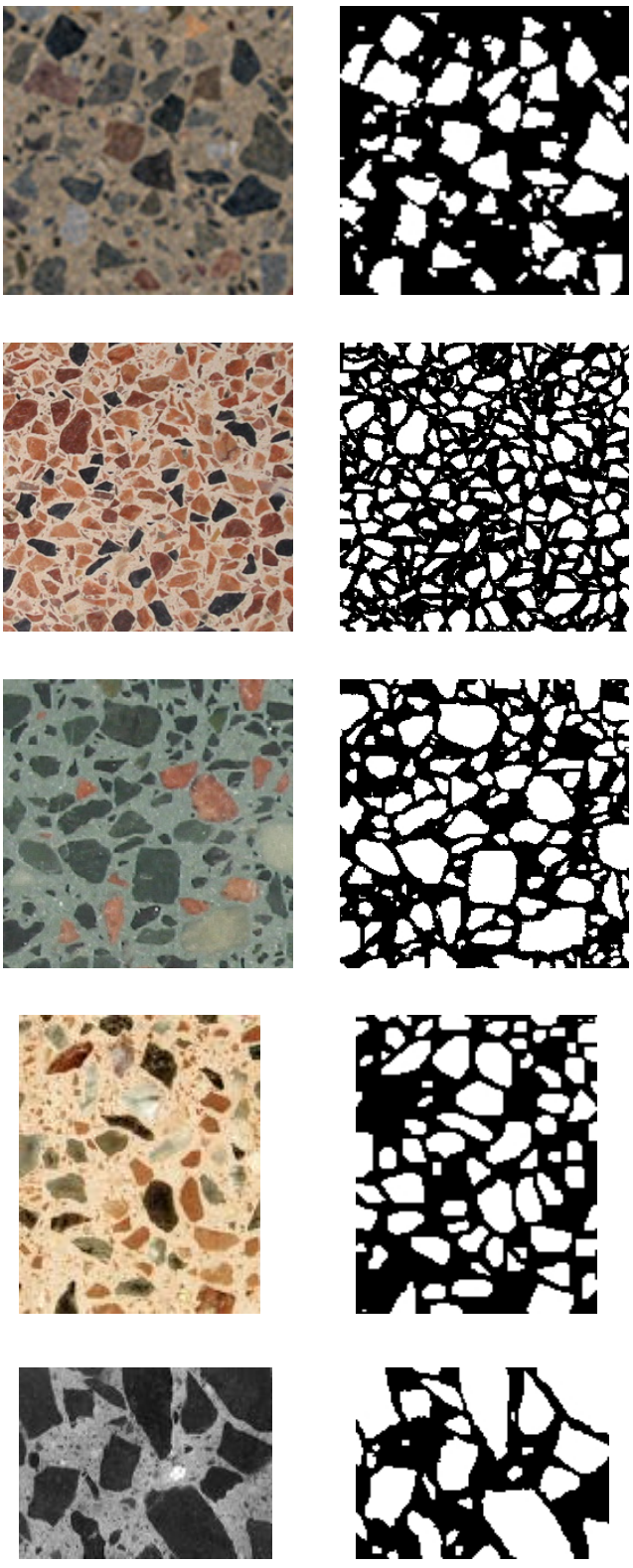


Fig. 8. (a): five input images used for the experiments, (b): the corresponding 2D particle's masks extracted from the five input images, and used for later synthesis process.

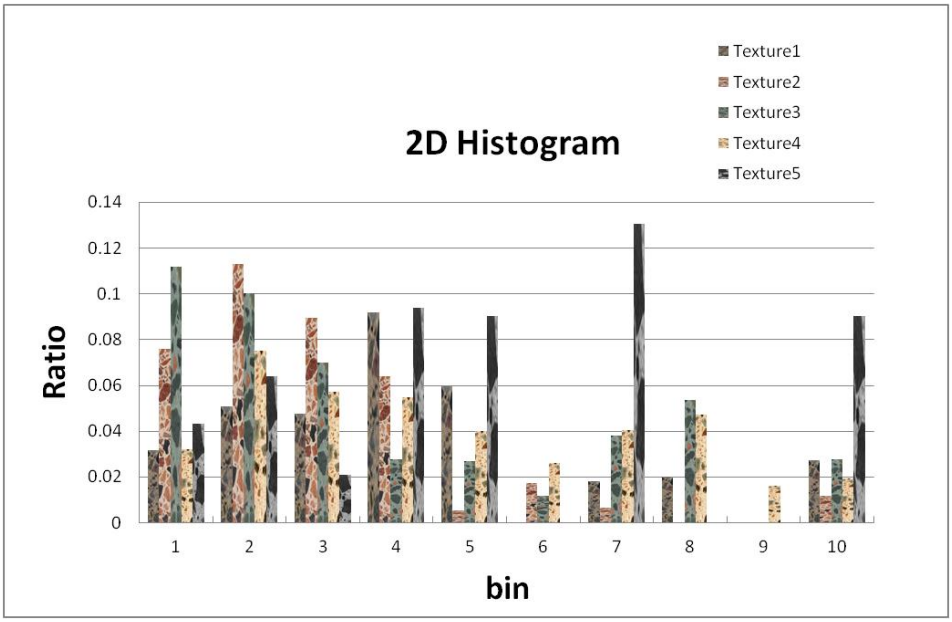


Fig. 9. Histogram of the areas of the 2D particles in the five input images.

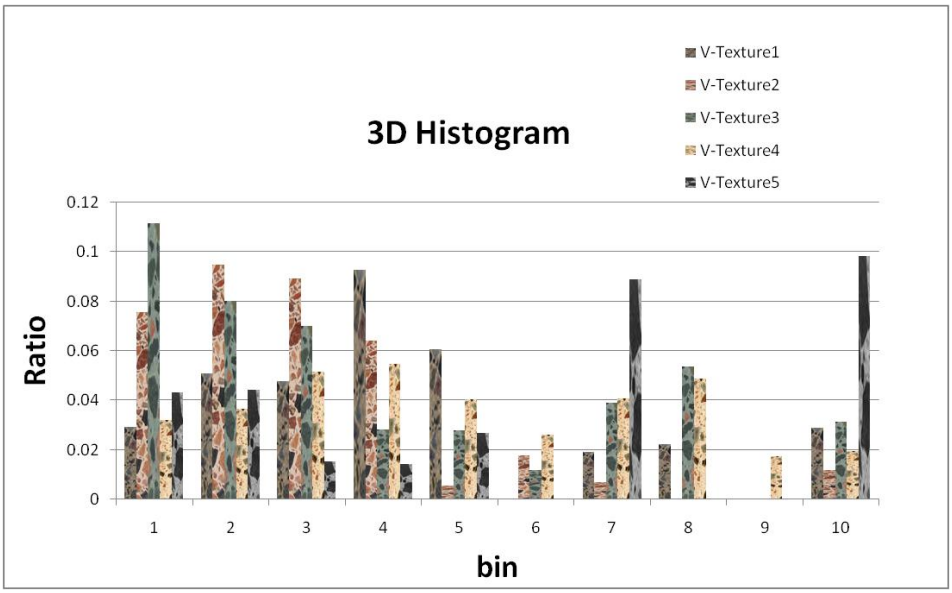


Fig. 10. Histogram of the volumes of the 3D particles in the synthesized volumes.

(a) (b) (c) (d)

Fig. 11. Comparison of Jagnow et al.'s and Kopf et al.'s solid texture results with ours for two of the five input textures shown in Fig. 8. (a) Results from Jagnow et al., (b) Results from Kopf et al., and (c) Results from our proposed methods. (d) The corresponding 2D input textures. Note that the results of (a) and (b) are cited from [14].

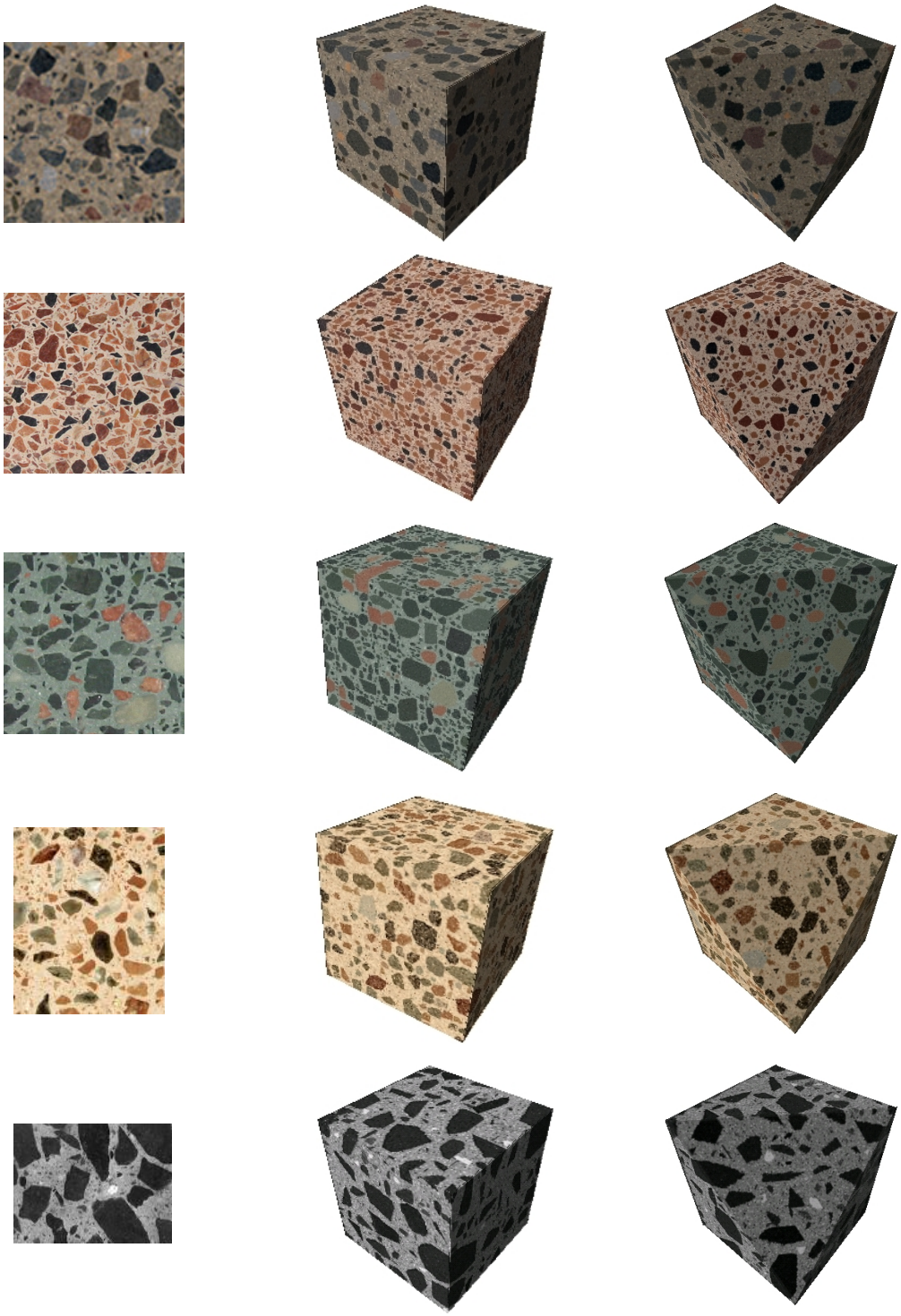


Fig. 12. (a): five input images used for the experiments, (b): the generated texture volumes, and (c): the corresponding results of (b) with parts of the volumes being sliced away to show the internal texture structures.

Fig. 13. More generated results. (a): five input images used for the experiments, (b) and (c): the horse and bunny models solid-textured by five textures generated from the input textures in(a).