Patch-based Synthesis of Geometry Textures with Point-set Surfaces

François Duranleau Pierre Poulin

Dép. d'informatique et de recherche opérationnelle
Université de Montréal

Abstract

As high-quality geometrical models become necessary for realistic applications, the creation of sophisticated surface details quickly becomes a crucial bottleneck to modeling. Geometry texture synthesis can alleviate this problem. We propose to combine geometry texture synthesis with point-set surfaces. Point-set surfaces form a powerful and flexible representation to encode intricate surface details. Our algorithm incrementally builds up a final geometry texture by fitting patches from an initial geometry texture, according to a distance field-based metric applied to a point neighborhood. An automatic point pairing scheme is used to warp the most similar patch with a thin-plate spline interpolation to make it concordant with its neighborhood. The point-set representation frees us from coping with explicit connectivity, while offering trivial manipulation for cutting, merging, and fitting portions of a surface. An appropriate blending corrects for any remaining small texture gaps. Experimental results are provided to illustrate the generality and the efficiency of our approach.

1 Introduction

Textures in computer graphics appear mostly in the form of details, whether as color or geometry alterations, on the surface of an object. These details are traditionally represented as 2D images (not necessarily color) mapped on a surface, and often associated with patterns such as cloth, wood, bricks, etc. Unfortunately covering an entire surface with a smaller texture using regular tiling results in objectionable visual patterns.

Texture synthesis attempts instead to generate from an original texture a new texture of arbitrary size with similar visual properties. Much work has been devoted to color textures, as indicated by extensive surveys [18, 13]. The most suitable textures for such generation processes respond to Markov random fields (MRFs) criteria, meaning they exhibit locality and stationarity properties. This class of textures represents a wide range of texture types.

This paper proposes to extend the texture generation process to geometry textures in order to produce adapted surface details, a very tedious task to perform manually.

Other researchers have addressed this problem. One simple solution applies standard color texture synthesis to height maps [28, 30], bounded height being substituted for color. However, this limits the geometric complexity of the surface details. It is also possible to apply texture synthesis to volumetric data (voxels with distance and/or density information) [15, 5]. This can represent arbitrary geometry, but may be costly in memory and slow to process. Geometry images [16, 19] also form a straightforward application of texture synthesis. Although they can represent more complex geometry than height maps, they cannot represent disconnected components.

Extending texture synthesis to surfacic representations has only recently been investigated. These approaches can potentially represent arbitrary topology and usually have a much smaller memory footprint than volumetric data. Zelinka and Garland [31] use graph-cut techniques on meshes to synthesize textures on one object from textures on another object. Closely related to our approach, Zhou *et al.* [32] also use meshes.

A point-set surface offers an interesting alternative representation to meshes. It proved a valuable tool for geometry processing and modeling (see [12] for a survey), as it requires no structures to explicitly maintain topology, and it allows direct processing of scanned data without explicit surface reconstruction.

We thus propose to exploit point-set surfaces and patch-based sampling schemes [17, 8, 14, 7, 29] in a general geometry texture synthesis approach. In

order to focus on the many elements affecting the synthesis process, we will limit our experiments to generation over a plane.

The rest of the paper proceeds as follows. Related work is presented in Section 2, our geometry texture representation is defined in Section 3, and our synthesis algorithm is first described in Section 4, and then detailed in Section 5. Our results are analyzed in Section 6, and a conclusion completes the discussion in Section 7.

2 Related Work

Modeling by example is not a concept exclusive to textures. Sloan *et al.* [24] presented a system to generate new shapes from a set of shapes with some interpolation scheme. Funkhouser *et al.* [9] developed a system to build piecemeal objects from different parts of a shape database. They retrieve different parts based on their similarity with a target shape. That similarity is evaluated by comparing distance fields, an idea we also exploit. Pauly *et al.* [22] also use search in a shape database to repair 3D scans. All these systems are essentially considered as macro-synthesis, as they build up geometry by *parts*, such as handles, arms, feet, etc. (except for [22]).

More closely related to our approach, Sharf *et al.* [23] and Park *et al.* [20] use principles of texture synthesis to repair 3D scans, including surface details. However, they aim at repairing models by filling holes, and not generating geometry textures. They also advocate the use of point-based representations.

3 Texture Representation

To synthesize geometry textures of arbitrary topology, we need more than height fields. The idea of our representation is akin to generalized displacement maps [26], and is quite similar in concept to the textures of Zhou *et al.* [32].

We define our input geometry texture $\mathcal{T}_{in}$ with the following elements:

- an axis-aligned bounding box $\mathcal{B}_{\mathcal{T}_{in}}$ defining the dimensions of the geometry texture.
- arbitrary manifold uniformly sampled point-set surfaces residing within $\mathcal{B}_{\mathcal{T}_{in}}$. This geometry must be defined beyond the limits of the bounding box in order to avoid ill-defined border conditions. We use the surface definition of Adamson and Alexa [2].
- an axis-aligned reference plane defining the orientation of the texture. The *height* of a texture is defined by the dimension of the bounding box perpendicular to this plane.

Figures 4, 5, 7, and 8 show examples of input geometry textures.

Our synthesis method is patch-based. A *patch* $P \subset \mathcal{T}_{in}$ is defined as the geometry inside an axis-aligned box of *height* equal to the texture's height. As for standard patch-based texture synthesis methods, the patch size should be chosen according to the size and distribution of features in $\mathcal{T}_{in}$. It should be large enough to encompass at least a texture element (*e.g.*, a bump in Figure 4) plus some neighborhood. If the patch is too small, the elements' shape and distribution may not be well preserved. If too large, there may be less variation in the output texture.

4 Synthesis Algorithm

The general process of our synthesis algorithm is similar to [8, 17]. It starts with a seed patch and then proceeds incrementally in scan-line order. Since the synthesis is performed along the texture's reference plane, the generation process is basically 2D. Thus, when we speak of a region, we actually refer to the volume defined by an area in the reference plane extruded to the full height of the texture.

At a given iteration k, we gather a point-set $\mathcal{N}_k$ (the neighborhood) around the region where a new patch will be placed. We then gather a set of patches matching $\mathcal{N}_k$ and randomly select one. The selected patch P_k is then translated to its destination position, transformed to better match $\mathcal{N}_k$, and merged with $\mathcal{T}_{out}$. For reasons explained later (Section 5.1), the patch has to be augmented with an extra region E_k extending out of P_k in the general synthesis direction. Figure 1 illustrates the terminology describing different regions.

Our synthesis algorithm thus proceeds as follows:

1. Randomly or manually select a seed patch P_0 from $\mathcal{T}_{in}$ and paste it at the lower left corner of $\mathcal{T}_{out}$ together with E_0. Set iteration counter $k = 1$.

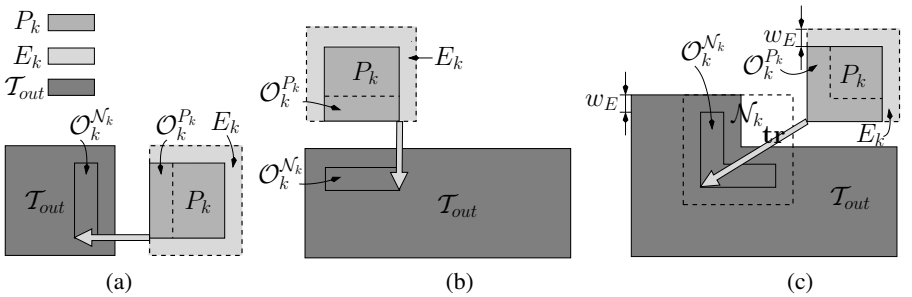

Figure 1: Synthesis process: (a) Start with an initial patch and proceed in scanline order. (b) A new patch starts a new scanline. (c) The general case. The dashed square over $\mathcal{T}_{out}$ represents the gathering region.

2. Find the set of candidate patches

$$\Psi_k = \{P \subset \mathcal{T}_{in} \mid d(\mathcal{O}_k^{\mathcal{N}_k}, \mathcal{O}_k^P) < d_{max}\}$$

where

$$d_{max} = (1+\epsilon) \min_{P \subset \mathcal{T}_{in}} d(\mathcal{O}_k^{\mathcal{N}_k}, \mathcal{O}_k^P).$$

$\mathcal{O}_k^*$ is the overlapping region of interest at iteration k in $\mathcal{N}_k$ or any patch (see Figure 1), d is a comparison metric between two overlapping regions, and ϵ is the relative error tolerance on d (details in Section 5.1).

3. Randomly select a patch P_k from Ψ_k. Set $\mathcal{F}_k = \mathbf{tr}(P_k \cup E_k)$, where the function $\mathbf{tr}$ translates the union to its destination position, and let $\mathcal{B}_k$ be the bounding box of $\mathcal{F}_k$.
4. Warp $\mathcal{F}_k$ to make it concordant with $\mathcal{N}_k$, and let $\mathcal{F}'_k$ be the result (details in Section 5.2).
5. Blend the geometries of $\mathcal{N}_k$ and $\mathcal{F}'_k$ into $\mathcal{F}''_k$ (details in Section 5.3).
6. Remove all points of $\mathcal{T}_{out}$ contained inside $\mathcal{B}_k$, and copy all samples from $\mathcal{F}''_k$ inside $\mathcal{B}_k$ into $\mathcal{T}_{out}$. Set $k = k + 1$.
7. Repeat steps 2 to 6 until $\mathcal{T}_{out}$ is filled.
8. Remove a region w_E wide along the boundaries of $\mathcal{T}_{out}$.

During the synthesis, we need to dynamically add and remove samples from $\mathcal{T}_{out}$, as well as locate neighbors. In order to achieve good performance for these operations, $\mathcal{T}_{out}$ is encoded as a hashed regular grid of buckets of points.

5 Processing a Patch

Several steps in the above algorithm involve searching and pasting a new patch during the synthesis process. In this section, we provide details for three basic steps: searching, warping, and blending.

5.1 Finding a Similar Patch

Step 2 of our algorithm creates the set Ψ_k of best candidate patches, according to a patch similarity metric d. As a large number of patches need to be evaluated, d needs to be fast to compute.

Zhou *et al.* [32] compute a similarity cost based on vertices' projection distances and normal differences. However, such a cost function is expensive to evaluate, particularly with point-sets.

Another commonly used similarity metric definition is the Euclidean distance, which is fast to evaluate, but would require to map our point-sets to an Euclidean space. Park *et al.* [20] and Zelinka and Garland [31] require local parameterization to evaluate feature vectors, which limits the local topology (no disconnected components). Sharf *et al.* [23] locally fit implicit functions to point-sets and take a few samples of signed distance and gradients of those functions. In our case, we need a denser sampling to allow more complex geometry inside our patches. We therefore opted to use signed distance fields [15, 9] together with a gradient field. The fields are computed only inside the overlapping regions $\mathcal{O}_k^*$. Let $\mathcal{D}(\mathcal{S})$ be the distance field of a given point-set $\mathcal{S}$, extended with the corresponding gradient field. By interpreting those fields as single high-dimensional vectors, our distance metric d is thus defined as:

$$d(\mathcal{O}_k^{\mathcal{N}_k}, \mathcal{O}_k^P) = \|\mathcal{D}(\mathcal{O}_k^{\mathcal{N}_k}) - \mathcal{D}(\mathcal{O}_k^P)\|^2$$

where $\|\cdot\|$ is the Euclidean norm, and $P \subset \mathcal{T}_{in}$.

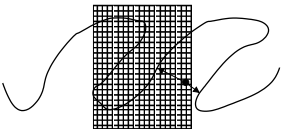

Figure 2: Error in distance computation: the long arrow indicates the computed distance inside the box, but the short arrow indicates the correct distance.

$\mathcal{D}(S)$ is computed by projecting points of a regular grid on the surface(s) of $\mathcal{S}$ using the quasi-orthogonal projection operator of Adamson and Alexa [2]. To determine the distance sign, the operator should always yield consistent normals, and therefore we require explicit normals for our point-sets. The user-defined grid resolution should be large enough to roughly capture the geometric details, yet because the field is used only for comparison and not for reconstruction, this resolution does not need to be very dense.

Only $\mathcal{D}(\mathcal{O}_k^{\mathcal{N}_k})$ has to be computed on the fly. For all $P \subset \mathcal{T}_{in}$, $\mathcal{D}(\mathcal{O}_k^P)$ is extracted from $\mathcal{D}(\mathcal{T}_{in})$. The resolution of $\mathcal{D}(\mathcal{T}_{in})$ discretizes the set of all possible patches.

Some care needs to be taken to accurately compute distances and gradients near a field's borders. Only considering the geometry within the field's extents may result in false distance values, as illustrated in Figure 2. To circumvent this problem, we allow projections to fall on the geometry outside the field's extents. Therefore geometry must extend $\mathcal{B}_{\mathcal{T}_{in}}$, as described in Section 3. This is also why we paste an extra region E_k together with P_k. It provides a padding region around $\mathcal{O}_k^{\mathcal{N}_k}$, as illustrated in Figure 1. We identify the width of those extra regions as w_E. It is set accordingly with the type and density of features in $\mathcal{T}_{in}$. In our examples, it varied between $\frac{1}{4}$ and $\frac{1}{3}$ of a patch's size. Larger proportions tend to only slow down computations.

A metric over an Euclidean space also offers the possibility to use search acceleration structures. Two popular and efficient methods are approximate nearest neighbor search (ANN) [3] and tree-structured vector quantization (TSVQ) [10], both of which have been used successfully for texture synthesis [17, 27, 11, 5]. However, the high dimensionality of our search spaces makes ANN about the same as an exhaustive search. We could make use of TSVQ by trading quality for speed, however, we have found that even by searching exhaustively, the search step is not a bottleneck of our synthesis method (see Table 1). We can help to make the search faster by unpacking all $\mathcal{D}(P), P \in \mathcal{T}_{in}$ as a precomputation step. This can be memory-intensive, but for all our test cases, it rarely takes more than 100-200 MB (with single precision floating point scalars).

5.2 Warping

After step 3 of our synthesis algorithm, a chosen patch P_k usually does not fit perfectly $\mathcal{N}_k$. For color texture synthesis, blending [17], finding an optimal seam [8, 14], or both methods are applied. Blending geometries proves much more difficult than blending colors, however.

Sharf *et al.* [23] use a method akin to ICP [4, 6], extended with a non-rigid quadratic warp. This warp proved not powerful enough for our needs. Wu and Yu [29] use feature correspondence to warp the patch with a thin-plate spline interpolation. We have opted for a similar approach. Using thin-plate splines involves finding a set of matching pairs of points from source and target surfaces. Wu and Yu [29] automatically match points based on proximity and tangents of features points. Dealing with geometry is more complicated. Most notably, we have to be careful that neighboring matching pairs do not cross each other to avoid local surface flips.

Let $S = \{s_1, ..., s_m\}$ be the source point-set and $Q = \{q_1, ..., q_n\}$ be the target point-set, $n \leq m$. Let W_Φ be the warping function for a given pairing Φ of points. We are looking for a pairing $\tilde{\Phi}$ from S to Q that would minimize a shape similarity metric between the surfaces defined respectively by $W_{\tilde{\Phi}}(S)$ and Q.

Finding such a pairing is difficult. We thus approximate it by minimizing a cost function for a given pairing Φ as

$$C(\Phi) = \sum_{(s,q)\in\Phi} [c(s,q) + g(s,q,\Phi)] \qquad (1)$$

where c is a cost function evaluating the quality of a given pair (s,q), $s \in S$, $q \in Q$, and g evaluates the coherence of (s,q) and its neighbors in Φ.

In our case, we define c as a function of the distance between two points, the difference of normals, and the difference in local surface variation [21],

which gives a rough estimate of the local shape. The importance of each aspect of c may vary from one geometry texture to another, so they are weighted with user-defined values as follows:

$$c(s,q) = \omega_p \|\mathbf{p}_s - \mathbf{p}_q\|_2 + \frac{\omega_n}{2}(1 - \mathbf{n}_s^{\mathrm{T}}\mathbf{n}_q) + \omega_v(v_s - v_q)^2 \quad (2)$$

where $\mathbf{p}_*$, $\mathbf{n}_*$, and v_* denote respectively the position, normal, and surface variation of a point, and w_p, w_n, and w_v their respective weights. If any of the three terms in Equation 2 is above its specific user-defined threshold, then we set $c(s,q) = \infty$. This eliminates bad pairings, such as two points with opposite normals.

The function g evaluates the difference in length and direction between a pair (s,q) and all its neighbors. Each aspect is computed and weighted as follows:

$$g(s,q,\Phi) = \omega_l \big| \|\mathbf{p}_s - \mathbf{p}_q\| - f_l(q,\Phi) \big| + \frac{\omega_d}{2}\left(1 - f_d(q,\Phi)^{\mathrm{T}} \frac{\mathbf{p}_s - \mathbf{p}_q}{\|\mathbf{p}_s - \mathbf{p}_q\|}\right)$$

where summing for all $(t,r) \in \Phi$

$$f_l(q,\Phi) = \frac{\sum \|\mathbf{p}_t - \mathbf{p}_r\| \theta(\|\mathbf{p}_q - \mathbf{p}_r\|)}{\sum \theta(\|\mathbf{p}_q - \mathbf{p}_r\|)}$$

$$f_d(q,\Phi) = \frac{\sum (\mathbf{p}_t - \mathbf{p}_r) \theta(\|\mathbf{p}_q - \mathbf{p}_r\|)}{\big\| \sum (\mathbf{p}_t - \mathbf{p}_r) \theta(\|\mathbf{p}_q - \mathbf{p}_r\|) \big\|}$$

and where θ is a monotonically decreasing function (a Gaussian, in our implementation). Weighting is computed from distances in Q rather than S, simply because $n \leq m$, which makes it more convenient to build pairings by matching elements of Q to elements of S.

Then, we define the set Υ of all possible pairings

$$\Upsilon = \Big\{ \{(s_{j_i}, q_i) \mid i = 1..n \text{ and } c(s_{j_i}, q_i) < \infty\} \;\Big|\; \{j_1, ..., j_n\} \text{ is a } n\text{−permutation of } \{1..m\} \Big\}.$$

The pairing $\tilde{\Phi}$ we are looking for is therefore

$$\tilde{\Phi} = \arg\min_{\Phi \in \Upsilon} C(\Phi). \quad (3)$$

Solving Equation 3 is a difficult task. We approximate the solution with an iterated greedy algorithm:

1. Set $M = \emptyset$, $\Phi = \emptyset$.
2. For each $q \in Q$:
   2.1. Find $s \in S$ such that $s \notin M$ and $c(s,q)$ is minimal.
   2.2. Set $M = M \cup \{s\}$, $\Phi = \Phi \cup \{(s,q)\}$.
3. Set $M = \emptyset$, $\tilde{\Phi} = \emptyset$.
4. For each $(s,q) \in \Phi$:
   4.1. Find $s \in S$ such that $s \notin M$ and $c(s,q) + g(s,q,\Phi)$ is minimal.
   4.2. Set $M = M \cup \{s\}$, $\tilde{\Phi} = \tilde{\Phi} \cup \{(s,q)\}$.
5. Set $\Phi = \tilde{\Phi}$.
6. Repeat steps 3 to 5 a given number of times.

When the algorithm terminates, the last state of $\tilde{\Phi}$ holds our pairing. $\tilde{\Phi}$ is then used to solve the thin-plate spline linear system [25] to obtain our warping function $W_{\tilde{\Phi}}$.

$W_{\tilde{\Phi}}$ essentially transforms positions, not normals. We could simply recompute the normals after the transformation using the projection operator [2], but we found the following technique more robust, albeit computationally more expensive. Assuming $\tilde{\Phi}$ has no crossings, let $s \in S$, $s' = W_{\tilde{\Phi}}(s)$, $\mathbf{u}_s$ and $\mathbf{v}_s$ be two tangent orthogonal vectors to $\mathbf{n}_s$, and J be the Jacobian matrix of $W_{\tilde{\Phi}}$, then

$$\mathbf{n}_{s'} = \frac{J\mathbf{u}_s \times J\mathbf{v}_s}{\|J\mathbf{u}_s \times J\mathbf{v}_s\|}.$$

Note that J can be analytically computed.

To fit these equations with the terminology of the previous sections, we have $Q \subset \mathcal{N}_k$ and $\mathcal{F}_k \subset S$. Q cannot be the full neighborhood for two reasons: (1) points in the outskirt of $\mathcal{N}_k$ are less reliable, notably for surface variation, and (2) considering all points would lead to huge and costly linear systems to solve, and would also make it difficult to avoid crossing pairs. We have found that simply running an adaptive clustering algorithm [21] in a sub-region of $\mathcal{N}_k$ yields a reasonable set Q. Also, because thin-plate spline interpolation has a global effect, $W_{\tilde{\Phi}}$ may generate deformations that are too strong in non-overlapping regions of $\mathcal{F}_k$. To counter this, we add to $\tilde{\Phi}$ a few extra pairs of points mapping to themselves in corner regions of $\mathcal{F}_k$. For the same reason, we actually have to extend $\mathcal{F}_k$ before warping, to ensure that the regions inside $\mathcal{B}_k$ are fully covered by $W_{\tilde{\Phi}}(\mathcal{F}_k)$. Finally, we have $\mathcal{F}_k \subset S$ because some points in the outskirt of Q may actually have a better match if we extend $\mathcal{F}_k$.

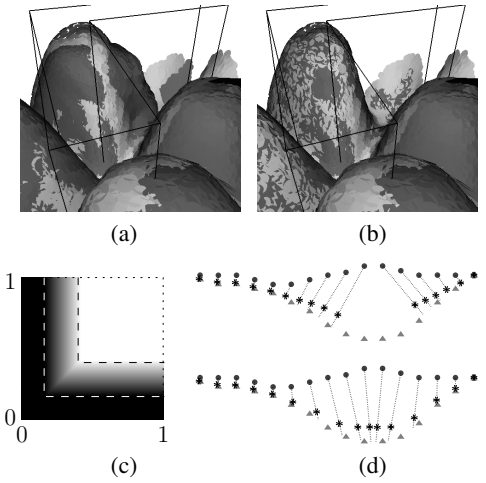

Figure 3: Blending. (a) $\mathcal{N}_k$ (light grey) superimposed with $\mathcal{F}'_k$ (dark grey). The bump in the center shows a remaining mismatch after warping. (b) The result of blending $\mathcal{F}'_k$ with $\mathcal{N}_k$. (c) Parametric value of the interpolation as a function of position inside the region enclosing $\mathcal{N}_k$. The dashed lines match the boundaries of $\mathcal{O}_k^{\mathcal{N}_k}$. (d) Difference between blending with projection (top) and ray intersections (bottom) over a valley.

5.3 Blending

Because the warping from the previous section is not perfect, gaps may still remain (see Figure 3 (a) and (b)). However as these gaps should be small, we can remove them with a simple progressive linear interpolation from $\mathcal{F}'_k$ to $\mathcal{N}_k$. The interpolation is actually done between points (with their normal) of $\mathcal{F}'_k$ and their projection [2] on $\mathcal{N}_k$. The parametric value varies gradually as illustrated in Figure 3 (c).

A small problem arises with the projection when the gap occurs over a valley, as it is more likely to fall on the sides than at the bottom, thus creating a hole in the sampling of the final result. We can improve this situation by using a limited form of ray intersection [1]. We test the intersection of two rays emanating from the point's position in the two directions parallel to the normal. We use the closest intersection. If no intersection is found, we fall back to regular projection. Figure 3 (d) illustrates the effect of this method.

6 Results

Our approach inherits the benefits and also some limitations from both patch-based texture synthesis and point-set surfaces. This section presents some of our results, and discusses their particularities.

The first example starts from a color texture used extensively in texture synthesis, converted into a height field. Figure 4 shows in the top left corner the original height field, with its bounding box in black wire frame, and two views of the synthesized height field generated at four times the original size.

Figure 5 illustrates the flexibility of our synthesis algorithm to different topology in presence of holes in the shape as well as smaller surface details. The patch-based approach gives good results even for larger generated patterns, as illustrated in Figure 6.

Figure 7 has inter-weaved and disconnected threads. The point-set behaves very well here, and the resulting generated pattern retains well-connected threads.

Figure 8 shows more disconnected 3D flowers. The general pattern is well captured by the synthesized flowers, and most individual flowers are adequately reconstructed.

In the previous examples, we applied a relaxation of the point-sets as a post-processing step to improve local sampling density.

None of our examples exhibit very sharp features. This limitation is due to the point-set representation. Indeed, compared to mesh-based approaches [32], a strategy based on or similar to moving-least squares (in our case: [2]) is not well suited to deal with sharp features. The point-set surfaces also require a high sampling density to disambiguate close opposing surfaces. Nevertheless, using this representation can still cover a wide range of geometry textures.

None of our original textures are tileable, which can create difficulties with synthesis techniques. The warping step helps to counter the problem. We give in Table 1 some values used for the different weights from Section 5.2. All examples have $w_p = 1$ and about the same base dimensions for $\mathcal{B}_{\mathcal{T}_{in}}$. Thus, we observe that the larger the features relative to the sampling rate (height field), the more important are the surface variation and normal terms in Equation 2, while smaller features on surfaces require lower weights, but a stronger smooth-

	Point-set size		Weight values				Synthesis time (seconds)				
	$\mathcal{T}_{in}$	$\mathcal{T}_{out}$	w_n	w_v	w_l	w_d	Search	Warp	Blend	Other	Total
Height field	33194	87101	3	30	9	7	6.7	11.2	10.8	2.0	30.7
Chain mail	71454	197282	1	10	8	6	8.5	7.3	18.4	3.8	38.0
Chain mail (large)	71454	2947085	1	10	8	6	147.5	201.3	316.2	45.8	710.8
Weave	43592	126139	2	15	6	5	5.7	7.7	12.0	2.6	28.0
Flowers	37473	68290	2	2	6	6	11.0	14.9	7.4	2.7	36.0

Table 1: User parameters and computation times. The number of input points corresponds to the total number of points, including padding geometry (see Section 3). On average, about 40% of the points lie inside $\mathcal{B}_{\mathcal{T}_{in}}$.

ing with neighboring pairs (chain mail *vs.* weave). Surface variation is less important in the presence of small scale symmetry (flowers).

The computation times in Table 1 were obtained on a 2.2 GHz AMD Athlon 64 processor. The generated textures have about four times the number of original points, except for the large chain mail. Precomputations on $\mathcal{D}(\mathcal{T}_{in})$ and on the surface variation of $\mathcal{T}_{in}$ take less than 10 seconds.

One can observe that our synthesis algorithm is much faster than volumetric approaches [15, 5] (they report hours of processing time). Under similar conditions (planar generation examples), Zhou *et al.* [32] report times of about three minutes. In our case, we seldom exceed one minute.

Many factors affect synthesis time. The most influential is the number of points in $\mathcal{T}_{in}$. Next comes the number of matching pairs for warping. The size of the thin-plate spline linear system to solve is directly proportional to this number, as is the evaluation of the warping function. The resolution of $\mathcal{D}(\mathcal{T}_{in})$ is another important factor, affecting search, although in all our examples, we used relatively low resolutions, between $51\times51\times12$ (weave) and $84 \times 68 \times 15$ (flowers).

7 Conclusion and Future Work

We have presented a patch-based synthesis algorithm that generates geometry textures using a point-set representation. Pairing, evaluating similarity, pasting, warping, and blending point-sets allow for efficient quality generation, while avoiding the issue of connectivity. The flexibility of the approach is illustrated over a number of different 3D geometric patterns. Mesh-quilting [32] also shares a similar flexibility. However, our method offers an alternative to handle scanned data without the need of conversion to mesh. Simple structures of point-sets also allow fast processing times.

The thin-plate spline interpolation requires quality pairing to yield good results. Finding such a pairing can be sometimes sensitive to the various weights. An in-depth investigation of pairing with different types of geometry should help to produce better results. A higher level surface characterization should allow to provide such insight.

Even though generation has been shown only over a planar surface, extensions to general 3D surfaces appear reasonably simple and promising, both in memory and processing requirements. This will be our next step. We will also investigate how this method behaves for repairing 3D models. Finally, we will apply our method to more sophisticated geometry synthesis, for instance in generating terrains, road maps, and entire environments.

References

[1] A. Adamson and M. Alexa. Approximating and intersecting surfaces from points. In *Eurographics Symp. on Geometry Processing,* 2003, 230–239

[2] A. Adamson and M. Alexa. On normals and projection operators for surfaces defined by point sets. In *Eurographics Symp. on Point-Based Graphics,* 2004, 149–156

[3] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998

[4] P.J. Besl and N.D. McKay. A method for registration of 3-D shapes. *IEEE Trans. on*

Pattern Analysis and Machine Intelligence, 14(2):239–256, 1992

[5] P. Bhat, S. Ingram, and G. Turk. Geometric texture synthesis by example. In *Eurographics Symp. on Geometry Processing,* 2004, 43–46

[6] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *Intl. Journal of Computer Vision and Image Understanding,* 10(3):145–155, 1992

[7] M.F. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for image and texture generation. *ACM Trans. on Graphics,* 22(3):287–294, 2003

[8] A.A. Efros and W.T. Freeman. Image quilting for texture synthesis and transfer. In *ACM SIGGRAPH* 2001, 341–346

[9] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin. Modeling by example. *ACM Trans. on Graphics,* 23(3):652–663, 2004

[10] A. Gersho and R.M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1991

[11] A. Hertzmann, C.E. Jacobs, N. Oliver, B. Curless, and D.H. Salesin. Image analogies. In *ACM SIGGRAPH* 2001, 327–340

[12] L. Kobbelt and M. Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics,* 28(6):801–814, 2004

[13] V. Kwatra. *Example-based Rendering of Textural Phenomena*. PhD thesis, Georgia Institute of Technology, 2005

[14] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Trans. on Graphics,* 22(3):277–286, 2003

[15] A. Lagae, O. Dumont, and P. Dutré. Geometry synthesis by example. In *Shape Modeling International*, 2005, 176–185

[16] Y. Lai, S. Hu, D.X. Gu, and R.R. Martin. Geometric texture synthesis and transfer via geometry images. In *ACM Symp. on Solid and Physical Modeling,* 2005, 15–26

[17] L. Liang, C. Liu, Y. Xu, B. Guo, and H. Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. on Graphics,* 20(3):127–150, 2001

[18] A. Nealen. Hybrid texture synthesis. Master's thesis, Technische Universität Darmstadt, 2003

[19] M.X. Nguyen, X. Yuan, and B. Chen. Geometry completion and detail generation by texture synthesis. *The Visual Computer,* 21(8–10):669–678, 2005

[20] S. Park, X. Guo, H. Shin, and H. Qin. Shape and appearance repair for incomplete point surfaces. In *IEEE Int. Conf. on Computer Vision,* 2005, 1260–1267

[21] M. Pauly, M. Gross, and L. Kobbelt. Efficient simplification of point-sampled surfaces. In *IEEE Visualization,* 2002, 163–170

[22] M. Pauly, N.J. Mitra, J. Giesen, M. Gross, and L.J. Guibas. Example-based 3D scan completion. In *Eurographics Symp. on Geometry Processing,* 2005, 23–32

[23] A. Sharf, M. Alexa, and D. Cohen-Or. Context-based surface completion. *ACM Trans. on Graphics,* 23(3):878–887, 2004

[24] P.J. Sloan, C.F. Rose, and M.F. Cohen. Shape by example. In *Symp. on Interactive 3D Graphics*, 2001, 135–143

[25] G. Wahba. *Spline Models for Observational Data*, volume 59 of *CBMS - NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, 1990

[26] X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H. Shum. Generalized displacement maps. In *Eurographics Workshop on Rendering,* 2004, 227–233

[27] L. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *ACM SIGGRAPH* 2000, 479–488

[28] L. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. In *ACM SIGGRAPH* 2001, 355–360

[29] Q. Wu and Y. Yu. Feature matching and deformation for texture synthesis. *ACM Trans. on Graphics,* 23(3):364–367, 2004

[30] L. Ying, A. Hertzmann, H. Biermann, and D. Zorin. Texture and shape synthesis on surfaces. In *Eurographics Workshop on Rendering,* 2001, 301–312

[31] S. Zelinka and M. Garland. Surfacing by numbers. In *Graphics Interface,* 2006, 107–113

[32] K. Zhou, X. Huang, X. Wang, Y. Tong, M. Desbrun, B. Guo, and H. Shum. Mesh quilting for geometric texture synthesis. *ACM Trans. on Graphics,* 25(3):690–697, 2006

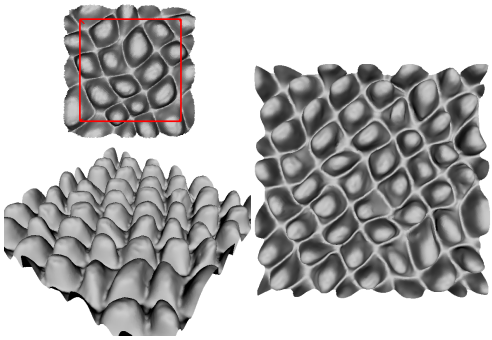

Figure 4: Original and synthesized height field.

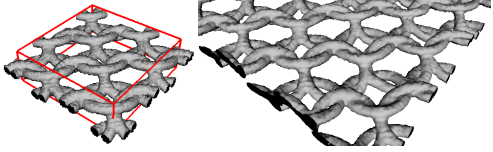

Figure 5: Original and synthesized chain mail with small bumps.

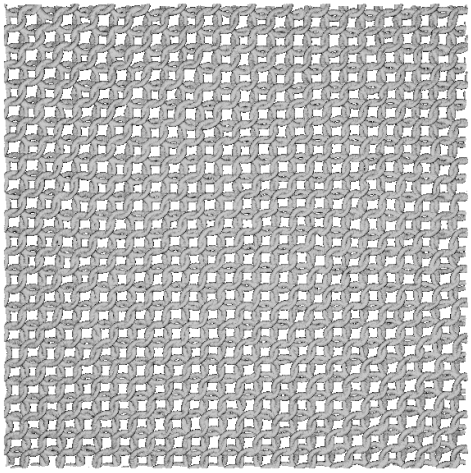

Figure 6: Chain mail generated over a larger area.

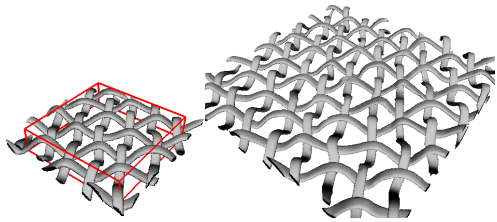

Figure 7: Original and synthesized weaving pattern.

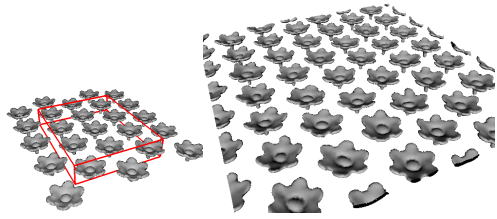

Figure 8: Original and synthesized flowers.