

Geometry Synthesis

Ares Lagae Olivier Dumont Philip Dutré

Report CW 381, March 2004

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Geometry Synthesis

Ares Lagae Olivier Dumont Philip Dutré

Report CW 381, March 2004

Department of Computer Science, K.U.Leuven

Abstract

Inspired by texture synthesis techniques, we present in this paper a method for *geometry synthesis*. Given an example of input geometry, we synthesize new output geometry that is perceived similar to the input geometry, but at the same time differs in its local appearance. We assume our input geometry satisfies the constraints of a Markov Random Field model, and represent the input by a hierarchical distance field. This allows us to perform fast matching queries between a target distance field that is partially synthesized, and the input distance field. Once the target distance field is completed, we copy the original corresponding geometry elements to our synthesized result. We show that automatically generating geometry by example can be achieved within reasonable computing times, and is able to produce convincing results.

Keywords : geometry synthesis, texture synthesis, distance fields, modeling, graphics

CR Subject Classification : I.3.5.1: Computing Methodologies - Computer Graphics - Computational Geometry and Object Modeling

Geometry Synthesis

Ares Lagae

Olivier Dumont

Philip Dutré

Department of Computer Science
Katholieke Universiteit Leuven *

Abstract

Inspired by texture synthesis techniques, we present in this paper a method for *geometry synthesis*. Given an example of input geometry, we synthesize new output geometry that is perceived similar to the input geometry, but at the same time differs in its local appearance. We assume our input geometry satisfies the constraints of a Markov Random Field model, and represent the input by a hierarchical distance field. This allows us to perform fast matching queries between a target distance field that is partially synthesized, and the input distance field. Once the target distance field is completed, we copy the original corresponding geometry elements to our synthesized result. We show that automatically generating geometry by example can be achieved within reasonable computing times, and is able to produce convincing results.

CR Categories: I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling

Keywords: geometry synthesis, texture synthesis, distance fields, modeling, graphics

1 Introduction

Geometry synthesis relates to geometry modeling much like texture synthesis relates to texture modeling. Textures can be created by hand, generated procedurally, or synthesized from existing textures. Likewise, geometry can be modeled by hand, and specific types of geometry can be generated procedurally. However, there are currently no techniques to generate geometry *by example*.

The problem of geometry synthesis can be stated as follows: given an example of input geometry, synthesize new geometry that, when perceived by a human observer, appears *similar* to the input geometry. Figure 1 shows an example. To formalize this concept of similarity, we impose the Markov Random Field (MRF) model on our geometry. This model assumes that the geometry is the realization of a local and stationary stochastic process. This means that the geometry at each location is characterized by the geometry in a relatively small neighborhood around that location (locality), and that this characterization is the same for each location on the geometry (stationarity). The synthesized geometry is then perceived similar to the input geometry if they both seem to be generated by the same stochastic process.

Our technique is a two-phase process. In the *analysis phase*, we compute the regularly sampled distance field of the input geometry, and organize its hierarchical representation in a search tree. In the *synthesis phase*, we use a tree search algorithm to search for the best match for a partially synthesized neighborhood of distance field samples, and replace the unsynthesized samples with those of the best match. In parallel, we construct the final synthesized geometry.

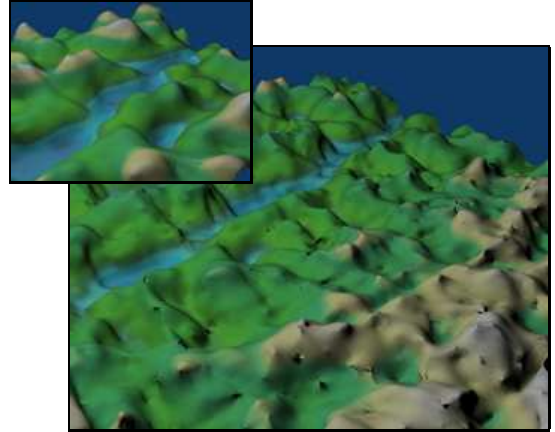


Figure 1: The terrain geometry in the large image (TERRAIN1) was synthesized from the example geometry shown in the top left corner.

2 Related Work

Texture synthesis

Numerous approaches have been proposed for texture synthesis. In this overview we limit ourselves to texture synthesis techniques that assume a MRF texture model, since our method for geometry synthesis assumes a MRF geometry model. For a more complete survey of texture synthesis algorithms, we refer to [Efros and Freeman 2001] and [Kwatra et al. 2003].

[Zhu et al. 1998] and [De Bonet 1997] describe seminal work in the area of MRF texture synthesis. [Zhu et al. 1998] synthesize new textures by sampling a probability distribution derived from the input texture. [De Bonet 1997] synthesizes new textures by sampling each spatial frequency band from a density estimator obtained by analyzing the original texture.

[Efros and Leung 1999] propose a non-parametric approach that synthesizes one pixel at a time. A pixel in the synthesized texture is determined by locating a pixel in the input texture with a neighborhood similar to the already synthesized neighborhood. The algorithm is beautiful in its simplicity, and produces decent results, but is very slow because a full search of the input texture is required to synthesize a single pixel. Based on earlier work by [Popat and Picard 1993; Popat and Picard 1997], [Wei and Levoy 2000] significantly accelerate the algorithm of [Efros and Leung 1999] using tree-structured vector quantization. [Hertzmann et al. 2001] build upon the work of [Wei and Levoy 2000]. They combine it with the approach of [Ashikhmin 2001], and generalize this combination to image analogies.

More recently, there has been a shift from pixel-based to patch-based approaches. Patch-based approaches were suggested by a variety of authors, including [Guo et al. 2000], [Harrison 2001] and also [Ashikhmin 2001]. [Guo et al. 2000] use the chaos mosaic to transform a tiling of the input texture to a texture with an even and visually stochastic distribution of the local features of the input

*email:{ares,olivierd,phil}@cs.kuleuven.ac.be

texture. [Harrison 2001] synthesizes textures by adding pixels one at a time. Pixels are chosen by searching the input image for patches that closely match the pixels already present in the output texture. [Ashikhmin 2001] also synthesizes one pixel at a time, but uses segmentation to split the input texture into pieces.

[Liang et al. 2001] formulate the texture synthesis problem as a k -nearest neighbors problem in a high dimensional space of texture patches. They use dimensionality reduction techniques and an approximate nearest neighbor algorithm to synthesize textures in real-time. Also [Harrison 2001] recognizes that nearest neighbor algorithms have difficulties with handling the dimension of the search space required for patch-based texture synthesis. [Efros and Freeman 2001] follow an approach similar to [Liang et al. 2001] and use a minimum error boundary cut to find a good seam between partially overlapping patches. Inspired by this approach, [Kwatra et al. 2003] synthesize a new texture by copying irregularly shaped patches from the input texture to the output texture. They use a graph cut technique to find the best seams. In general, patch-based techniques achieve high quality results.

BTF synthesis

The bidirectional texture function (BTF) is a 6D function that describes textures arising from small scale geometry and spatially variant surface reflection. BTF synthesis can be seen as an intermediate approach between texture synthesis and geometry synthesis. [Tong et al. 2002] did seminal work in the area.

Procedural modeling

The family of geometric techniques most closely related to geometry synthesis is known as procedural modeling. The key difference between procedural modeling and geometry synthesis is that procedural modeling can only be used for a parameterized class of models, whereas geometry synthesis generates geometry *by example*, without imposing a specific model on the input geometry.

Procedural modeling has been successfully applied to problems like urban modeling [Parish and Müller 2001], plant modeling [Prusinkiewicz and Lindenmayer 1996], and the generation of fine geometric detail on objects [Fleischer et al. 1995]. [Greene 1989] introduces a stochastic modeling technique that *grows* models in voxel space, according to rules based on simple relationships like intersection, occlusion and proximity.

Although procedural modeling is a powerful approach, the formulation of a procedure suited to generate a specific class of models can be quite an involved task.

3 Design Decisions & System Overview

Design Decisions

Images and textures are regular 2D arrays of color sample values. The difference between textures and images is that sample values of textures are usually constrained by some model (e.g. a MRF texture model). Intuitively, texture synthesis can be seen as composing a new texture using elements from an input texture.

To extend the ideas of texture synthesis to geometry synthesis, we need a suitable geometry representation. Key ingredients of our method include a hierarchical geometry representation, a definition of geometry similarity, and composing geometric elements. After much consideration, we chose regularly sampled 3D distance fields [Frisken et al. 2000], which are well suited for all of these operations. A distance field is a scalar field that specifies the minimum distance to the surface. Composing polygons is more difficult than composing elements structured in a regular grid. Therefore we do not work directly with polygon meshes. We have also considered

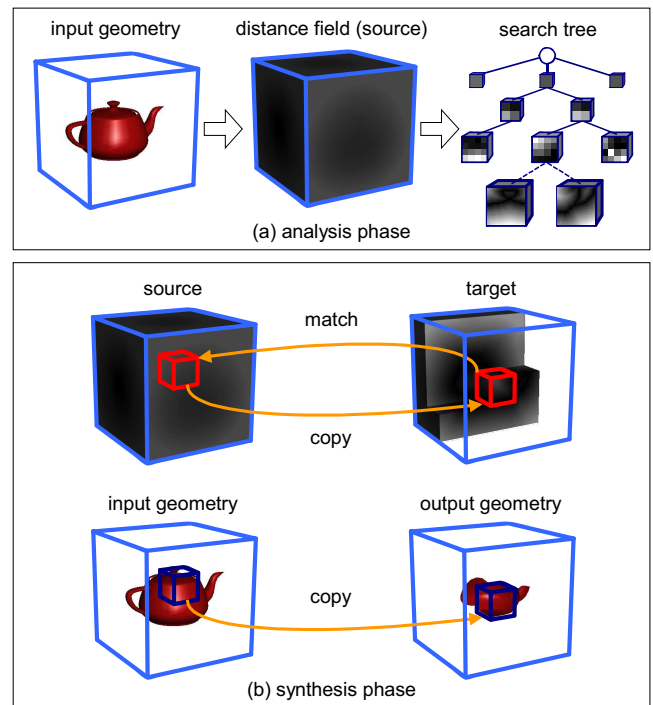


Figure 2: System overview. (a) In the analysis phase, the distance field of the input geometry is computed (source), and a hierarchical representation of the source is organized in a search tree. (b) In the synthesis phase, a new distance field is synthesized (target) by copying the best matching source block for a partially synthesized target block from the source to the target. In parallel, the synthesized geometry is constructed.

using voxel grids, but distance fields store more information at little extra cost, and voxel grids are somewhat awkward to subsample.

Texture synthesis teaches us that algorithms synthesizing one pixel at a time are very limited in the choice of values that can be assigned to a pixel that is being synthesized. Most pixels have their value almost completely determined by what has been synthesized already. A single pixel seems to be too small as a unit of synthesis. This suggests the use of patches rather than individual pixels [Efros and Freeman 2001]. In the context of geometry synthesis, we will use the term *block* rather than patch. A block is defined as a cubic regular 3D array containing samples from a distance field, and the *block size* as the number of samples in each of its dimensions.

At first sight, it seems interesting to follow the approach of [Liang et al. 2001], and formulate the problem of geometry synthesis as a nearest neighbor problem in a high dimensional space. However, for geometry synthesis, the dimensionality of the search space increases to a point where (approximate) nearest neighbor algorithms are not much faster than a simple brute force linear search [Arya et al. 1998]. To tackle this problem, and to make geometry synthesis practical, we have designed a fast geometry matching algorithm, inspired by the A^* algorithm [Hart et al. 1968].

Finally, we have learned from texture synthesis that multiresolution approaches improve the quality of the synthesized result. Therefore, we formulate our geometry synthesis algorithm as a multiresolution synthesis algorithm.

System Overview

Figure 2 shows an overview of our system for geometry synthesis. In the analysis phase, the distance field of the input geometry is computed. We call this distance field the *source*. A hierarchical

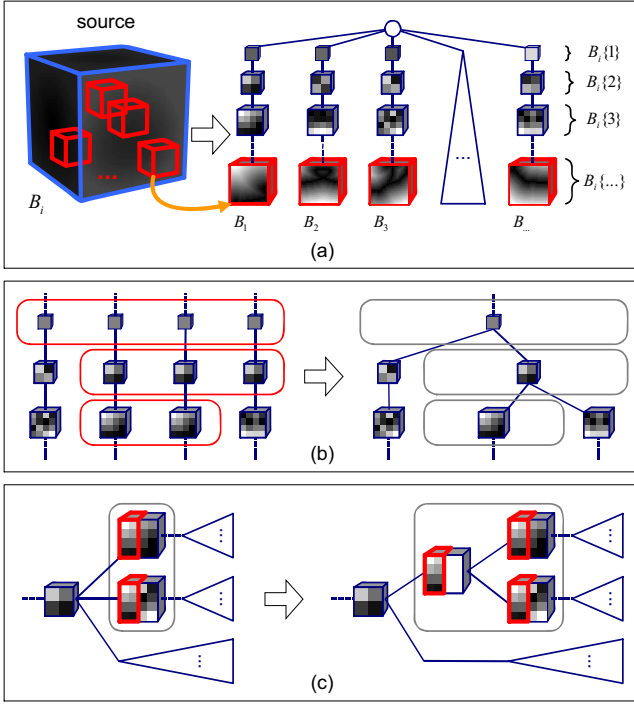


Figure 3: Search tree construction. (a) A tree is constructed from the levels of the block pyramid of each block in the source. (b) Identical nodes are grouped. (c) A parent node is introduced for partially identical nodes.

representation of the source is then organized in a search tree. This allows us to formulate a fast geometry matching algorithm. The construction of the search tree and the geometry matching algorithm are discussed in section 4. In the synthesis phase, we use the geometry matching algorithm to synthesize a new distance field, called the *target*, by copying the best matching source block for a partially synthesized target block from the source to the target. When a block is copied, we output the corresponding geometry. The synthesis phase is discussed in section 5.

4 Geometry Matching

The geometry matching algorithm provides a solution for the following problem: for a given query block, find the most similar block in a given distance field. To this end, we first organize a hierarchical representation of the distance field in a search tree, and then employ a tree search algorithm to locate the best match for the query block.

4.1 Search Tree Construction

For a given size, we consider all possible blocks in the distance field of equal size. Note that all these blocks also overlap each other. For each of these blocks, a complete subsampled distance field pyramid is built, which we call a *block pyramid*, by averaging the samples. To facilitate the construction of the block pyramids, we require that the block size is a power of 2. We also quantize distance field samples to 256 values. For a block B , $B\{l\}$ denotes level l in the block pyramid of B , level 1 using the lowest resolution (1^3). For a block size of N , there are exactly $\log_2 N + 1$ levels in the block pyramid.

These hierarchical block representations are organized in a tree by making each level of each block pyramid a node in the tree, and

connecting the subsequent levels (Figure 3a). An imaginary root node is introduced and connected to level 1 of all block pyramids.

Nodes in the tree that are at the same level and have identical subsampled distance fields are combined (Figure 3b). Note that because we quantize distance field samples to 256 values, there are at most 256 nodes at level 1.

There is a lot of duplicate information in the tree, especially in the leaf nodes (consider two blocks that have a large overlap). Therefore we do not store the levels of the block pyramids in the nodes themselves, but we use the nodes as references into a globally subsampled distance field pyramid.

4.2 Block Similarity

We define a distance measure between two blocks B and B' of equal size N :

$$\text{dist}(B, B') = \frac{1}{N^3} \sum_{i=1}^N \sum_{j=1}^N \sum_{k=1}^N (B[i, j, k] - B'[i, j, k])^2 \quad (1)$$

where $B[i, j, k]$ denotes the sample in block B at location (i, j, k) . The distance between two blocks is a measure for their similarity. If the distance increases, the similarity decreases.

Given the block pyramids of two blocks B and B' , we define the distance between the two blocks at the level l , $\text{dist}(B\{l\}, B'\{l\})$, in the same way, but now N is the resolution of the l th level rather than the block size.

Because the levels in a block pyramid are constructed by simple averaging, we know that

$$\text{dist}(B\{l\}, B'\{l\}) \leq \text{dist}(B\{l+1\}, B'\{l+1\}) \quad (2)$$

for any level l (see appendix A for a formal proof). This means that when comparing the levels of two block pyramids at subsequently increasing resolution, their similarity decreases.

The value of any two samples in a distance field can differ at most the actual distance between those sample locations, and therefore there exists a bound ϵ such that

$$\text{dist}(B\{l+1\}, B'\{l+1\}) \leq \text{dist}(B\{l\}, B'\{l\}) + \epsilon \quad (3)$$

for any level l . We derive this bound in appendix A.

The geometry matching algorithm is based on these two equations. If we know the distance between blocks at a specific level, then equation 2 and 3 respectively provide a lower bound and an upper bound for the distances between those blocks at all higher levels. In fact, our algorithm can be used with any distance measure and subsampling strategy, as long as at least equation 2 is satisfied.

4.3 Geometry Matching Algorithm

Given the tree representation of a distance field, and a query block, the search algorithm finds the block in the tree that best matches the query block, according to equation 1.

The algorithm associates each node with the distance between the level of the block pyramid contained in that node and the corresponding level in the block pyramid of the query block. The goal is now to find the leaf node with the smallest associated distance. Equation 2 tells us that distances associated with child nodes are always larger than the distance associated with their parent. Therefore, the descendants of a specific node need not to be considered as long as the algorithm knows other nodes with an associated distance smaller than the one of that specific node.

The algorithm keeps all visited nodes in a priority queue, which is sorted according to increasing distance. Initially, the queue contains only the root of the tree. In each iteration, the front node of the queue is removed, and all its children are added to the queue. When

```

// a node contains a single level of a block pyramid
Node { Block block, Int level }

// a priority queue entry contains a node and a distance
// the priority queue keeps entries sorted according to increasing distance
PriorityQueueEntry { Node node, Real distance }

// find the block in the search tree that best matches the given block
Block FindBestMatch(SearchTree tree, Block block) {
    PriorityQueue queue
    queue.Insert(PriorityQueueEntry(tree.GetRoot(), ∞))
    loop {
        PriorityQueueEntry front = queue.RemoveFront()
        Node node = front.node
        if (node.IsLeaf()) return node.block
        else foreach (Node child in node.GetChildren()) {
            Real distance = dist(block { child.level }, child.block { child.level })
            queue.Insert(PriorityQueueEntry(child, distance))
        }
    }
}

```

Figure 4: Pseudocode for the geometry matching algorithm.

a leaf node ends up at the front of the queue, the distance between the block contained in that node and the query block is less than the distance associated with each other node in the queue. The distance associated with each other node in the queue is also less than the distance associated with its descendants, and all other leaf nodes are still reachable from a node in the queue. Therefore, the leaf node at the front of the queue contains the block that best matches the query block. Figure 4 shows the pseudocode of the algorithm.

As an optimization, the priority queue can be pruned during a run of the algorithm. When a node is added, equation 3 predicts the worst possible distance associated with its child nodes. Because of equation 2, any node in the queue with a distance greater than that one can be safely removed from the queue. This optimization only makes sense if the size of the queue becomes a problem, because pruned nodes will never reach the front of the queue.

Our search algorithm is a heuristic tree search algorithm (in which equation 2 serves as the heuristic), and is related to the A^* algorithm [Hart et al. 1968], on which it was inspired.

4.4 Branching Factor

When removing a node from the queue, the distance associated with each child node is calculated. If the branching factor in the tree is high, nodes have many children, and this becomes an expensive operation. Moreover, this is a wasted effort for child nodes that never reach the front of the queue. To avoid this situation, we decrease the branching factor of the tree by introducing a parent node for nodes that have partially identical subsampled distance fields (Figure 3c). This reduces the average branching factor per node, and makes the search faster. The most important change in the algorithm as discussed above, is that the distance evaluations must now be carried out incrementally.

4.5 Performance

We have verified experimentally that this algorithm is three to four orders of magnitude faster than a linear search. Table 1 summarizes some performance measurements. Timings were obtained on a 2 GHz desktop PC, using the BALLS1 example from figure 6. Matching a block of size 32 in a 256^3 distance field takes on the average roughly 10s. Note that this corresponds to a nearest neighbor

distance field resolution	64^3	128^3	256^3
block size	8	16	32
blocks in distance field	$\approx 1.75 \times 10^8$	$\approx 1.4 \times 10^6$	$\approx 1.1 \times 10^7$
average matching time	0.025s	0.5s	10s
speedup factor	460	1400	5150

Table 1: Performance of the geometry matching algorithm compared to a linear search.

query in a $32^3 = 32768$ dimensional space with about 11 million candidate points. A simple brute force linear search would take approximately half a day.

We would like to point out that the performance of our algorithm is dependent on the actual data it is working on. In the worst case, all leaf nodes have to be considered, and our algorithm is equivalent to a linear search. For example, this would be the case when matching blocks in a grid containing random noise. However, in practice (with *real* distance fields), this situation never occurs.

Considering that there are no general purpose (approximate) nearest neighbor algorithms that can handle search spaces with a dimension larger than about 30 much more efficiently than a linear search, the geometry matching algorithm is very fast. This kind of performance is of course only possible by exploiting as much problem specific information as possible.

5 Geometry Synthesis

In this section, we explain how the geometry matching algorithm can be used for geometry synthesis. For clarity, we will first present a single resolution version of our method, and then extend it to multiresolution.

5.1 Single Resolution

For single resolution synthesis, we first create a distance field that will hold the synthesized result (the target). Usually, the target is larger than the source. The target is seeded by assigning to each sample value the value of a random sample from the source. Then, an initial block in the target is selected (e.g. a corner block), and the geometry matching algorithm is used to locate the best match in the source, which is then copied to the target. We select a next block in the target that overlaps an already synthesized area, and again copy its best match to the target. If a block is copied from the source to the target, only the non-synthesized samples from the target are overwritten. This procedure is repeated until the target is completely synthesized. Figure 5 illustrates the initial steps of this process.

Usually, we select the next block so that $\frac{1}{8}$ th of block contains unsynthesized samples. In the most common synthesis case, this corresponds to a configuration where 7 octants of the block contain already synthesized data, and 1 octant of the block is being synthesized.

Many texture synthesis methods also initialize the target with random values to introduce some randomness in the synthesis process. However, in texture synthesis it is common to use an L-shaped causal neighborhood (already synthesized pixels), or match patches based only on their already synthesized overlapping parts. With our geometry matching algorithm this is difficult, because block overlaps are in general not cubic. Therefore, we match whole blocks, but initialize the target with random samples from the source. These will match approximately equally well with any block in the source, so the matching process is dominated by the overlap.

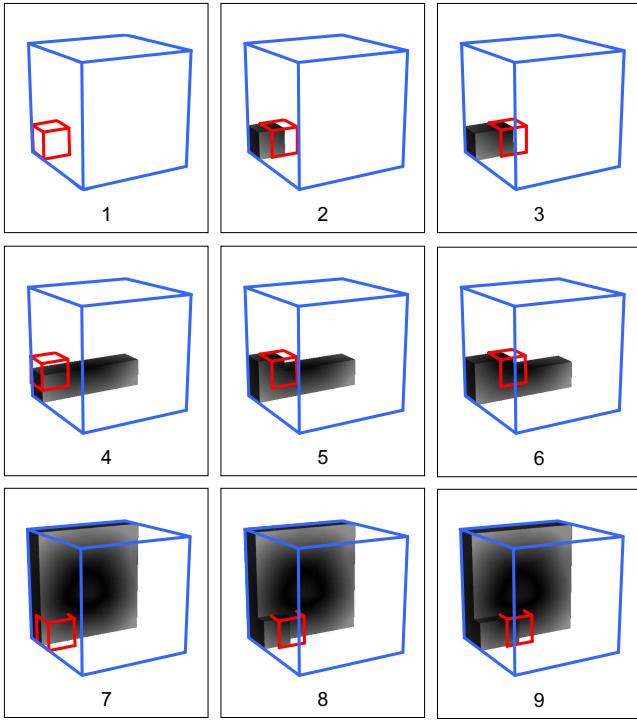


Figure 5: Consecutive steps in the single resolution synthesis process.

5.2 Multiresolution

Single resolution synthesis enforces local similarity by matching blocks. To also enforce similarity on a larger scale, we apply single resolution synthesis iteratively on different scales. The source is repeatedly subsampled (in the same way blocks are subsampled), and the first iteration works on the lowest resolution version of the source. The target is seeded with random samples from the source only for the first iteration. For next iterations, the target is seeded with the result of the previous iteration. The first iteration works with a relatively large block size, and the block size decreases in each iteration.

The idea behind this is that the first iterations will capture large features of the input geometry, while detail is added in the last iterations. Note that the search tree does not need to be reconstructed for each iteration. By removing all leaf nodes from a search tree, the tree for a subsampled distance field is obtained.

The target is composed of (parts of) blocks from the source, and neighboring blocks in the target are not necessarily next to each other in the source. Because of this, the target is not a *valid* distance field in the sense that the premise of equation 3 does not necessarily hold, especially when there are random samples in the query block. This implies that it is not correct to use equation 3 to prune the queue when employing the geometry matching algorithm for geometry synthesis. However, this is not a big problem for two reasons. First, a priority queue entry is very small, it only contains a distance and a reference to a node (in our implementation we did not experience problems with the queue size). Second, in a multiresolution setting, queue pruning is not needed for the first iteration, because the search tree is relatively small. After the first iteration, there are no random samples in the target, so queue pruning can be applied if (in the worst case) some error can be tolerated. After all, it is not imperative that the block that is synthesized is the best possible match, as long as it is a good match.

5.3 Geometry Reconstruction

It is unlikely that the geometry corresponding with two neighboring (parts of) blocks in the target will fit exactly. A similar problem is encountered in patch-based texture synthesis, where it is solved by making patches overlap, and then blending their overlap [Liang et al. 2001] or searching a minimum error boundary in the overlap [Efros and Freeman 2001].

For geometry synthesis, this problem is more difficult to solve. When a block is copied from the source to the target, we output the corresponding geometry, and afterward, we smooth the output geometry using mesh relaxation. An alternate approach would be to use an extension of an algorithm based on marching cubes (e.g. the algorithm described in [Kobbelt et al. 2001]) to generate the final geometry directly from the target. However, discontinuities in the synthesized distance field would probably complicate this. Another approach would be to handle the generated geometry as an unorganized point cloud (most connectivity information was lost during synthesis anyway), and generate geometry from that representation (e.g. using the technique described in [Hoppe et al. 1992]).

Most likely, the best results would be obtained with a specialized geometry reconstruction operator, guided by the input geometry and the synthesized distance field. However, we did not implement such an operator.

6 Results & Discussion

Figure 6 shows synthesis results for a variety of inputs. The input geometry for FIBERS was generated procedurally, and then the position, thickness, and scale of the individual fibers were perturbed by hand. The input geometry for METAL was generated from a photograph of the micro-structure of rough metal. The input for CHAINMAIL consists of a number of identical tori, placed and rotated by hand, to mimic the structure of chainmail. The input geometry for GRID was also modeled by hand, and then perturbed. In each of these cases, our method succeeds in capturing the appearance of the input geometry, and synthesizing similar geometry. The geometry that served as input for BALLS1 and BALLS2 was created by a physical simulation of dropping a large number of balls in a box. BALLS2 was generated with a smaller window size than BALLS1, and is therefore less similar to the input geometry. For clarity, Figures 6d and 6f only show a slice of the synthesized geometry.

Figure 7 illustrates that geometry synthesis can also be used to synthesize textures. We have created a height field from the luminance values of the highlighted part of the texture in Figure 7a. This texture has been used extensively by a number of researchers in the field of texture synthesis, e.g. [De Bonet 1997] and [Efros and Leung 1999]. Figure 7b shows the height field and the synthesized geometry. Figure 7c shows the textures created from the geometry by encoding the height of the geometry in the textures.

Figures 1 and 8 show that geometry synthesis can also be used to generate terrain. TERRAIN1 and TERRAIN2 were both generated from the same input geometry. Note that our synthesis algorithm does not take into account color information. Each vertex of the terrain geometry was assigned a color according to its height.

Figure 9 shows the GRASS example. The input geometry consists of a collection of individual grass blades. It is rather high frequent, and poses an large challenge for our method. Although some small pieces of grass are visible, many grass blades have “grown” larger than any blade in the input geometry. The GRASS example was also colored afterward.

All examples were generated with multiresolution synthesis using 3 iterations, a source resolution of 64^3 , a target resolution of 128^3 , and a block size of 16 for the last iteration (except for BALLS2). The size of a unit of synthesis at the most detailed level

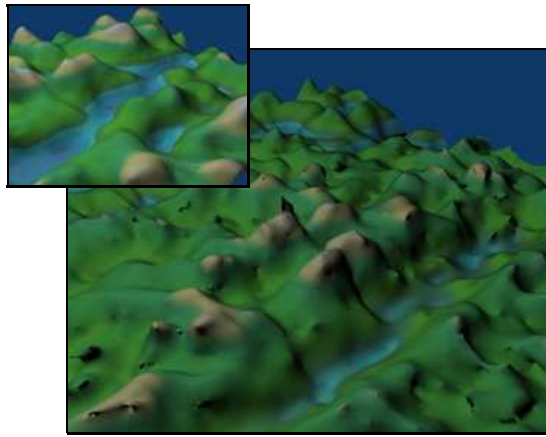


Figure 8: Using geometry synthesis to generate terrain (TERRAIN2). This terrain was generated using the same input geometry as is in Figure 1.

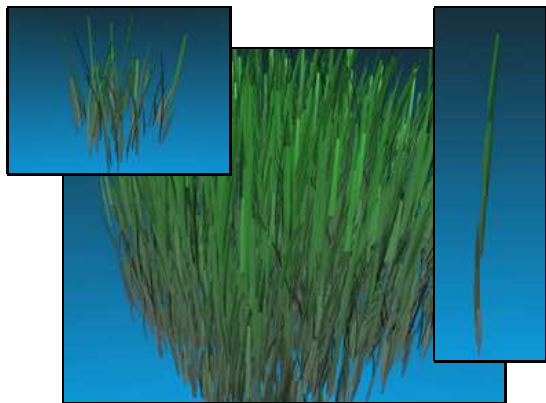


Figure 9: Synthesizing grass (GRASS). The input geometry, the synthesized result, and a closeup of a single grass blade.

corresponds roughly with one octant of the ball at the crossing of tree cylinders in the GRID example. The scale of all examples is approximately the same.

Synthesis times are in the range from about 1 hour (e.g. for GRID) to 12 hours (e.g. for BALLS1), measured on a 2 GHz desktop PC. This includes the time needed for constructing the search tree. Although our geometry matching algorithm is fast, a lot of matches are needed to synthesize the complete target.

Our input geometry is represented using regular triangle meshes, and we use a naive automatic smoothing operator to repair discontinuities in the synthesized geometry. This is why some artifacts are visible in the output geometry. We believe that our results can be further improved with a specialized geometry reconstruction operator.

7 Conclusion & Future Work

We have presented a technique for geometry synthesis based on a fast geometry matching algorithm, and have showed that, although computationally expensive, geometry synthesis is possible. We hope that our geometry matching algorithm will also be useful in other contexts, and that its ideas will be used to speed up other applications that require searching in a high dimensional space.

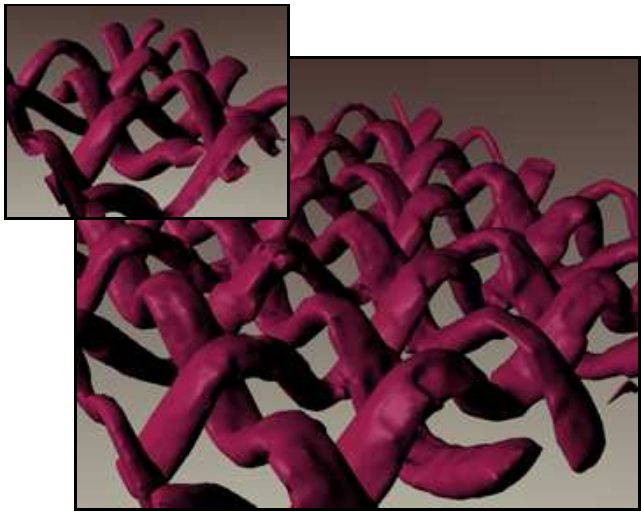
Opportunities for future work include:

- Specialized geometry reconstruction operators to cope with the typical artifacts in synthesized meshes should be investigated.
- Our algorithm could be extended to account for other signals present on the geometry, such as color information.
- It would be interesting to extend existing applications of texture synthesis to similar applications operating on geometry. For example, constrained texture synthesis could be extended to constrained geometry synthesis. This could then be used, for example, to fill holes in scanned data.
- We are interested in extending the ideas presented in this paper to geometry synthesis over arbitrary topology surfaces. For example, one could model the shape of cloth, and then synthesize cloth fiber geometry over the surface. This would not only be a valuable modeling tool, but could also be useful for level of detail applications, and to synthesize BTF's.

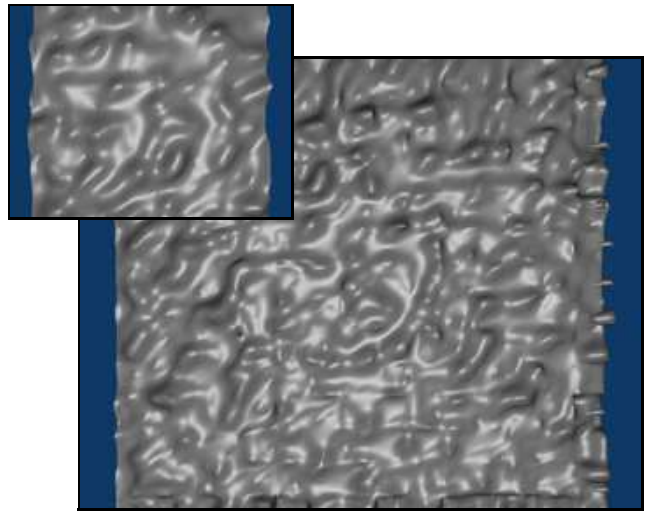
Acknowledgments Many thanks to all the people in our research group for their countless clever suggestions. The first and second author are funded as a Research Assistant by the Fund of Scientific Research - Flanders, Belgium (Aspirant F.W.O.-Vlaanderen).

References

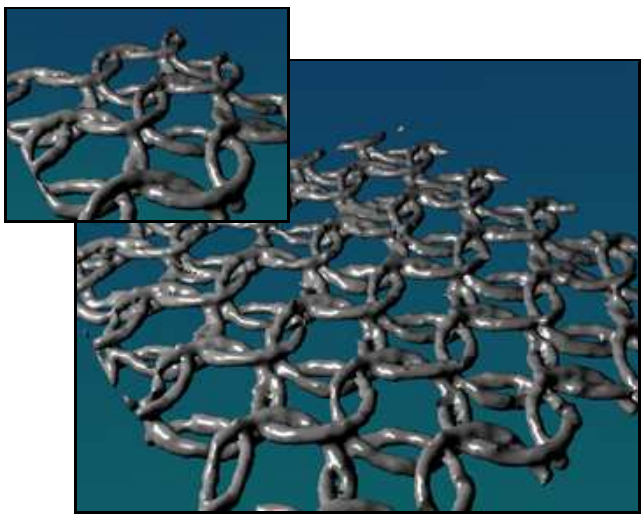
- ARYA, S., MOUNT, D. M., NETANYAHU, N. S., SILVERMAN, R., AND WU, A. Y. 1998. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM* 45, 6, 891–923.
- ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, 217–226.
- DE BONET, J. S. 1997. Multiresolution sampling procedure for analysis and synthesis of texture images. *Computer Graphics* 31, Annual Conference Series, 361–368.
- EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *SIGGRAPH 2001, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH, E. Fiume, Ed., 341–346.
- EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *ICCV (2)*, 1033–1038.
- FLEISCHER, K. W., LAIDLAW, D. H., CURRIN, B. L., AND BARR, A. H. 1995. Cellular texture generation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM Press, 239–248.
- FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Siggraph 2000, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, K. Akeley, Ed., 249–254.
- GREENE, N. 1989. Voxel space automata: modeling with stochastic growth processes in voxel space. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, ACM Press, 175–184.
- GUO, B., SHUM, H., AND XU, Y.-Q. 2000. Chaos mosaic: Fast and memory efficient texture synthesis. Technical report MSR-TR-2000-32, Microsoft Research.
- HARRISON, P. 2001. A non-hierarchical procedure for re-synthesis of complex textures. In *WSCG 2001 Conference Proceedings*, V. Skala, Ed.
- HART, P. E., NILSSON, N. J., AND RAPHAEL, B. 1968. A formal basis for heuristic determination of minimum path cost. *IEEE Transactions Systems Science and Cybernetics* 4, 2, 100–107.
- HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *SIGGRAPH 2001, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH, E. Fiume, Ed., 327–340.



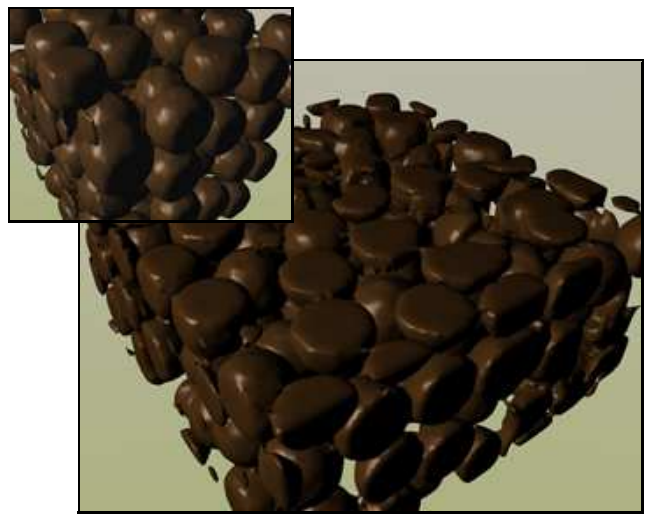
(a)



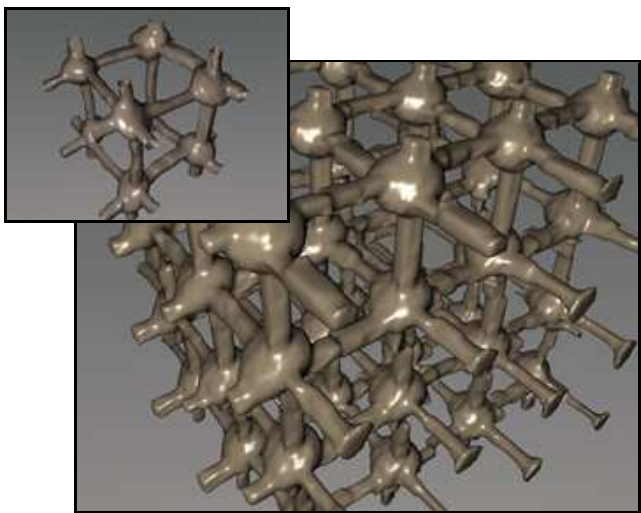
(b)



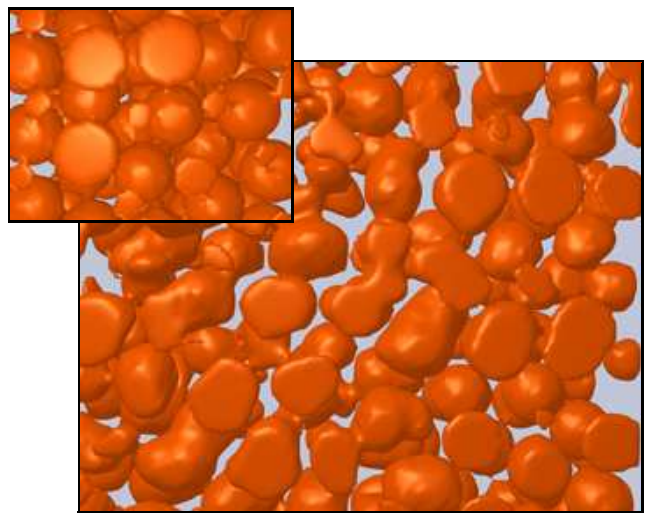
(c)



(d)



(e)



(f)

Figure 6: Results of our geometry synthesis method. The input geometry is shown at the top left corner of the output geometry. Both input and output geometry are shown at approximately the same scale. (a) FIBERS. (b) METAL. (c) CHAINMAIL. (d) BALLS1. (e) GRID. (f) BALLS2.

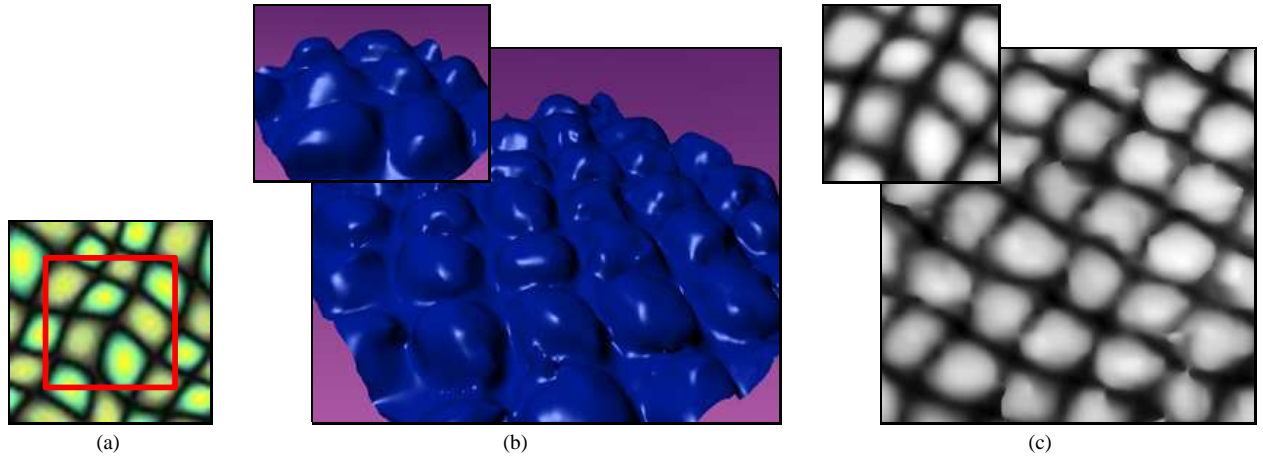


Figure 7: Using geometry synthesis to synthesize textures (TEXTURE). (a) The highlighted part of this texture was used to create a height field. (b) The height field was used to synthesize a new height field. (c) Textures created from both height fields.

HOPPE, H., DE ROSE, T., DUCHAMP, T., McDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. *Computer Graphics* 26, 2, 71–78.

KOBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature-sensitive surface extraction from volume data. In *SIGGRAPH 2001, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH, E. Fiume, Ed., 57–66.

KWATRA, V., SCHDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003* (July).

LIANG, L., LIU, C., XU, Y.-Q., GUO, B., AND SHUM, H.-Y. 2001. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.* 20, 3, 127–150.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. In *Proceedings of SIGGRAPH'01*, 301–308.

POPAT, K., AND PICARD, R. 1993. Novel cluster-based probability model for texture synthesis, classification, and compression.

POPAT, K., AND PICARD, R. W. 1997. Cluster based probability model and its application to image and texture processing. *IEEE Trans. Image Processing* 6, 2, 268–284.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1996. *The Algorithmic Beauty of Plants*. The Virtual Laboratory. Springer-Verlag, New York.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 665–672.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *Siggraph 2000, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, K. Akeley, Ed., 479–488.

ZHU, S. C., WU, Y., AND MUMFORD, D. 1998. Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling. *Int. J. Comput. Vision* 27, 2, 107–126.

A Proofs

A.1 Proof of Equation 2

Suppose B_x and B_y are two 1D distance fields of size 2, and that the sample values of B_x and B_y are given by $B_x = (x_1, x_2)$ and $B_y = (y_1, y_2)$. Because there are two levels in their block pyramids, $B_x = B_x\{2\}$, and $B_y = B_y\{2\}$. Due to the subsampling,

$B_x\{1\} = \left(\frac{x_1+x_2}{2}\right)$, and $B_y\{1\} = \left(\frac{y_1+y_2}{2}\right)$. The distances according to equation 1 are given by:

$$\text{dist}(B_x\{2\}, B_y\{2\}) = \frac{1}{2} \left((x_1 - y_1)^2 + (x_2 - y_2)^2 \right) \quad (4)$$

$$\text{dist}(B_x\{1\}, B_y\{1\}) = \left(\frac{x_1+x_2}{2} - \frac{y_1+y_2}{2} \right)^2 \quad (5)$$

Substituting (4) into (5):

$$2 \text{dist}(B_x\{1\}, B_y\{1\}) = \text{dist}(B_x\{2\}, B_y\{2\}) + (x_1 - y_1)(x_2 - y_2) \quad (6)$$

Using the binomial theorem:

$$(x_1 - y_1)(x_2 - y_2) \leq \text{dist}(B_x\{2\}, B_y\{2\}) \quad (7)$$

Substituting (7) into (6):

$$\text{dist}(B_x\{1\}, B_y\{1\}) \leq \text{dist}(B_x\{2\}, B_y\{2\}) \quad (8)$$

By applying this reasoning inductively, equation 2 can be proved.

A.2 Proof of Equation 3

We first show that the value of any two samples in a distance field can differ at most the actual distance between those sample locations. Suppose that s_1 and s_2 are two samples in a distance field that are a distance d apart, and that p is the point on the surface that is closest to s_1 . Let d_1 and d_2 be the distances from p to s_1 and s_2 . If p is also the closest point to s_2 , then $d_2 \leq d + d_1$ and thus $|d_1 - d_2| \leq d$. If this is not the case, let d'_2 be the distance between s_2 and its closest point on the surface. Then $d'_2 \leq d_2$, and $d'_2 \leq d + d_1$ still holds, thus $|d_1 - d'_2| \leq d$.

We now prove equation 3, using the same assumptions of section A.1, by bounding the term $(x_1 - y_1)(x_2 - y_2)$ from (6). Let $\epsilon_x = \frac{x_1+x_2}{2} - x_1$, and $\epsilon_y = \frac{y_1+y_2}{2} - y_1$, then

$$x_1 - y_1 = \left(\frac{x_1+x_2}{2} - \frac{y_1+y_2}{2} \right) - (\epsilon_x - \epsilon_y) \quad (9)$$

$$x_2 - y_2 = \left(\frac{x_1+x_2}{2} - \frac{y_1+y_2}{2} \right) + (\epsilon_x - \epsilon_y) \quad (10)$$

From (9) and (10) follows:

$$(x_1 - y_1)(x_2 - y_2) = \text{dist}(B_x\{1\}, B_y\{1\}) - (\epsilon_x - \epsilon_y)^2 \quad (11)$$

We define ϵ as the squared maximum distance between neighboring samples. Then $(\epsilon_x - \epsilon_y)^2 \leq \epsilon$, and from (11) and (6) follows:

$$\text{dist}(B_x\{2\}, B_y\{2\}) \leq \text{dist}(B_x\{1\}, B_y\{1\}) + \epsilon \quad (12)$$

Again, by applying this reasoning inductively, equation 3 can be proved, and a bound ϵ can be obtained for the general case. However, by applying this reasoning inductively, the constraint on the sample values becomes weaker. With considerably more algebra, an even tighter bound ϵ can be obtained.