Synthesizing Natural Textures

Michael Ashikhmin

University of Utah www.cs.utah.edu

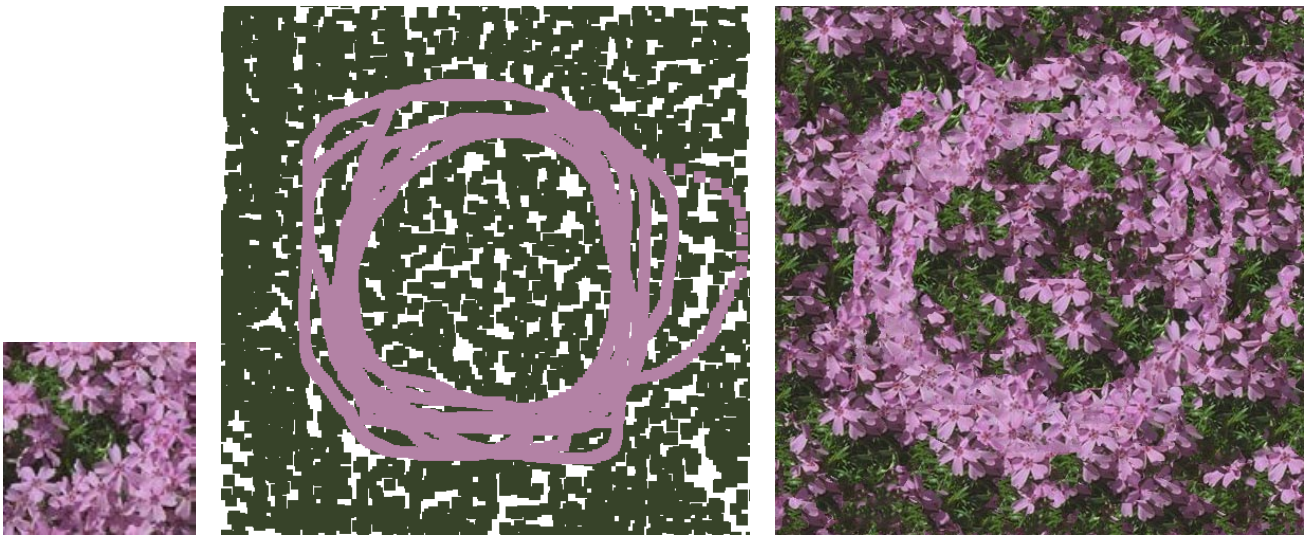

Figure 1: *An example of user controlled synthesis done by a first time user of the system. Left: input sample, Center: user-drawn target image, Right: synthesized result created in time short enough for an interactive loop. This particular result was subjectively chosen by the user from a few trials with the same target image.*

Abstract

We present a simple texture synthesis algorithm that is well-suited for a specific class of naturally occurring textures. This class includes quasi-repeating patterns consisting of small objects of familiar but irregular size, such as flower fields, pebbles, forest undergrowth, bushes and tree branches. The algorithm starts from a sample image and generates a new image of arbitrary size the appearance of which is similar to that of the original image. This new image does not change the basic spatial frequencies the original image; instead it creates an image that is a visually similar, and is of a size set by the user. This method is fast and its implementation is straightforward. We extend the algorithm to allow direct user input for interactive control over the texture synthesis process. This allows the user to indicate large-scale properties of the texture appearance using a standard painting-style interface, and to choose among various candidate textures the algorithm can create by performing different number of iterations.

CR Categories: I.3.7 [Computing Methodologies]: Computer Graphics—Three-Dimensional Graphics and Realism, Color, shading, shadowing, and texture

Keywords: texture synthesys, user interaction

1 Introduction

Texture mapping [2] is a common method that adds realism to computer generated images. While texture mapping itself is straightforward, acquiring the images to use for textures is not always easy. Procedural texture generation [11, 12] provides help for a specific classes of materials such as wood and marble, although the control parameters of such textures can be difficult to choose. Alternatively, a texture *synthesis* method starts from a sample image and attempts to produce a texture with a visual appearance similar to that sample. Potential benefits of texture synthesis include the ability to create large and/or tilable textures from a small sample in a direct manner. Unfortunately, creating a robust and general texture synthesis algorithm has proved difficult. A vast body of work on textures is available so we only touch on the most related publications in this paper. Most of early work on textures was done in the computer vision community (for a survey of this earlier literature see the paper by Haralick [8]). Not surprisingly, that work mainly emphasized the aspects of textures useful for the vision problem, such as their great value for object recognition. Several general texture synthesis algorithms have been developed [5, 7, 9, 13] but until recently both running time [7] and, in many cases, quality [5, 9] of synthesized textures has left much to be desired. In addition we are aware of no published attempt to make the texture synthesis process more “end-user-friendly”. The only control the user usually has is a

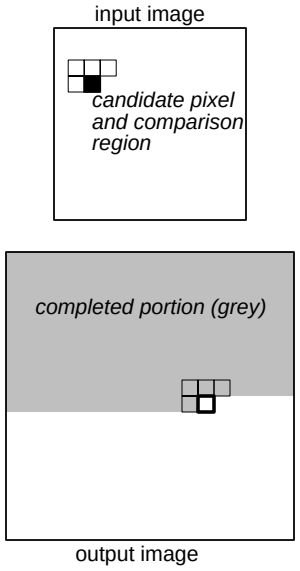

Figure 2: *The Wei and Levoy (WL) texture synthesis process. Pixels are generated in scanline order. The value of a pixel is determined by choosing the best pixel in a candidate list. For WL this list includes all pixels in the input image. Best pixel is the one whose L-shaped neighborhood most closely resembles the neighborhood of the pixel currently being synthesized. For clarity, a smaller than usual (only 3x3) neighborhood is shown on all figures.*

set of largely unintuitive parameters which usually change only the overall quality of the output texture with no clear effect on, for example, the spatial distribution of features in the synthesized image. Dischler and Ghazanfarpour [6] address a more difficult problem of extracting 3D macrostructures from texture images. Once this is done, the shape of the macrostructures can be interactively changed and then mapped onto a model. Since this problem is much more complicated than the usual 2D texture synthesis, it is not surprising that their algorithm is much more involved than the one we present.

The inspiration for our work comes from the recent algorithm by Wei and Levoy [16]. Their simple approach (abbreviated as WL below) is intuitive and works surprisingly well for a wide variety of textures. Unfortunately, we have found that there is a particular class of familiar textures on which the WL algorithm performs relatively poorly. These are textures consisting of an arrangement of distinct objects of irregular but familiar shapes and sizes. Such arrangements are often encountered in nature such as talus fields, grass, tree branches and leafs, pebbles, blossoming flowers, bushes, and forest undergrowth. Because our initial interest in texture synthesis was primary due to its value for outdoor rendering, this class is of prime importance to us. We present a modification of the WL algorithm which performs better on this specific type of textures. Our method is significantly faster than the basic or accelerated WL algorithm while the coding complexity is about the same as the basic WL algorithm. This speed up is crucial in allowing us to develop a technique which provides the user with intuitive interactive control over the results of the synthesis process.

Because our method is based on the WL algorithm, we provide a brief overview of this algorithm in Section 2 and our results are compared with it throughout the paper. Sections 3 and 4 present our approach and explain how to give intuitive control to the user. We show the results we obtain and further discuss the algorithm and its limitations in Section 5. We conclude by outlining some possible extensions of this work.

2 Wei and Levoy Texture Synthesis

The WL algorithm uses a synthesis approach which, although based on the Markov Random Field model of texture, avoids explicit probability function construction and consequent sampling from it. This is accomplished by generating the output image pixel by pixel in scanline order, choosing at each step a pixel from the sample image which neighborhood is most similar with respect to a specified measure to the currently available neighborhood in the texture being synthesized. More specifically, in the most basic version of the algorithm, the following steps are performed:

- The output image is initialized to random noise with a histogram equal that of the input image.
- For each pixel in the output image, in scanline order, do:
  - in the output image, an L-shaped neighborhood of current pixel of a specific (fixed) size is considered, see Figure 2.
  - a search is performed in the input sample for a pixel with a neighborhood most similar to the one identified in the previous step.
  - the current pixel value in the output image is copied from the position in the input sample identified as the most similar by this search.

The similarity of two neighborhoods N_1 and N_2 is computed according to the L_2 distance between them which is a sum over all pixels in the neighborhood of squared differences of pixel values at a particular position:

$$D(N_1, N_2) = \sum_{p\ in N} \{(R_1(p) - R_2(p))^2 + (G_1(p) - G_2(p))^2 + (B_1(p) - B_2(p))^2\}. \quad (1)$$

Here R, G and B are pixel values at position p in red, green and blue channels respectively and subscripts refer to the image (either sample or output) to which particular neighborhood belongs.

The size of the neighborhood should be on the scale of the largest regular texture structure to capture its low frequency components. Since the amount of work to evaluate Equation 1 is directly proportional to the number of pixels in the neighborhood, this size critically affects the running time of the algorithm. For a typical neighborhood necessary to synthesize a good quality image (9x9 square of pixels or 40 pixels in the L-shaped neighborhood) the simple algorithm just presented is relatively slow, on the order of minutes for a 200x200 texture. There are a number of ways this basic form can be modified to make the algorithm produce better results faster. The two most important ones discussed in the original paper are multiresolution synthesis which allows to use smaller neighborhoods, and tree-structured vector quantization (TSVQ) to accelerate the search for the best matching pixel. These methods dramatically increase the speed of the algorithm but they also introduce significant extra implementation complexity. In this work we build upon only the simplest version described above. Reader interested in other aspects of the original algorithm are referred to Wei and Levoy's original paper [16] for more details.

2.1 Discussion

The WL algorithm works surprisingly well given its simplicity. For many textures the quality of synthesized images is as good as with any other known technique but with runtime orders of magnitude faster. However, the algorithm produces output using the smooth L_2 norm as a measure of neighborhood similarity. This is certainly

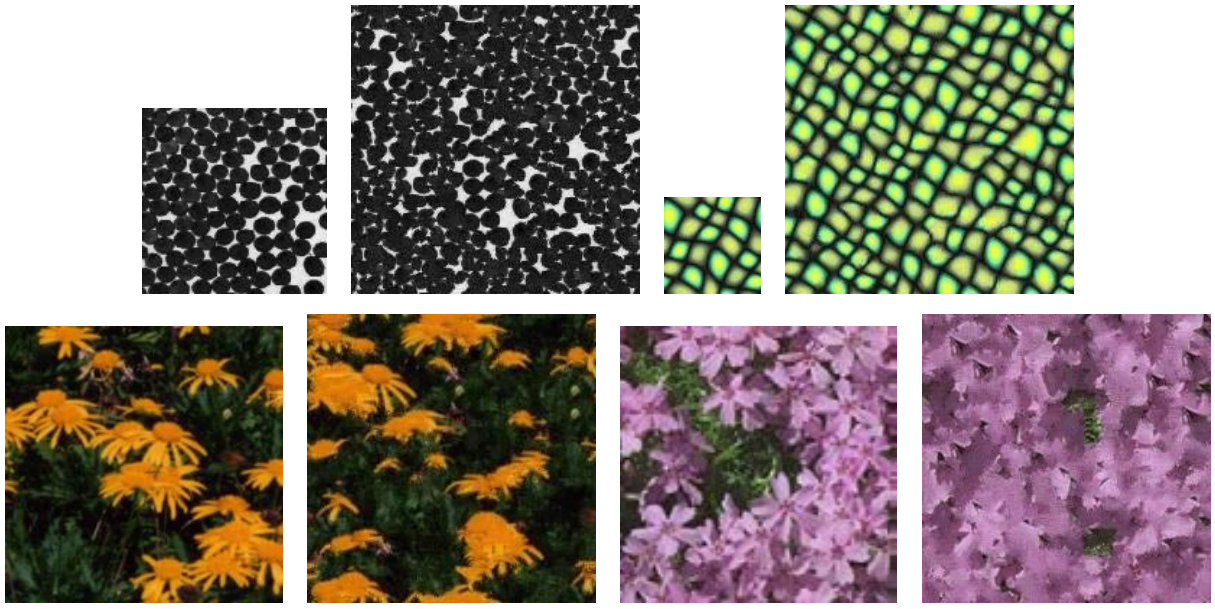

Figure 3: *Wei and Levoy texture synthesis algorithm: good and bad cases. The algorithm has a tendency to blur out finer details for some textures. See text for more details.*

not a good measure of perceptual similarity between two patches. For example, it is well-known that the human visual system is very sensitive to edges, corners and other higher level features in the image. This behavior is not captured by the simple L_2 norm and the algorithm tends to produce textures with smoothed out edges in some cases. Since it can never assign a value not present in the original sample to a pixel, this problem does not exist in an image with a histogram not rich enough to create a smooth transition between two areas. A black-and-white image is the extreme case of this, see Figure 3, top left. Another case where applying the algorithm produces excellent results occurs when the underlying textures are smooth and the absence of well-defined edges is what is actually wanted. This is the case for the texture used as an example throughout the original paper (see Figure 3, top right). Finally, the current neighborhood is often unique enough to compensate for the blurring tendency of the L_2 measure, allowing only a relatively edge-preserving construction of the output image, as for the texture in Figure 3, bottom left. All images generated by the WL algorithm which are shown in this paper were taken directly from the project's web page (http://www.graphics.stanford.edu/projects/texture/).

The WL algorithm runs into difficulties in cases where the texture to be synthesized should consist of an arrangement of small objects. These are very common in outdoor rendering with some examples mentioned in the introduction and one such difficult case shown on Figure 3, bottom right, and more can be seen in Figure 9. Unfortunately, an observer is usually familiar with general forms and sizes of the objects being looked at (leaves, flowers, pebbles, etc.) and the tendency of the algorithm to grow smooth shapes of uncontrollable form degrades the quality of the output significantly. The problem gets only worse with TSVQ acceleration which, as Wei and Levoy point out, tends to blur the output even more and special measures to avoid this behavior become necessary (Section 4.2 of [16]). It is not clear whether this behaviour is partially due to the use of inherently smoothing Gaussian pyramids by WL algorithm and whether a different type of pyramid would help. The fact that single resolution results of Wei and Levoi look very similar to multiresolution ones suggest that this solution might not be fully adequate. Another possible way to improve the algorithm is to replace simple L_2 norm with a more sophisticated perceptual metric. Although some such metrics do exist [3, 15], they are currently not completely reliable due to the imperfect understanding of human visual system. Also, these metrics are much more computationally expensive than the L_2 norm.

An even simpler approach which works in exactly these cases which are difficult for the WL algorithm is chaos mosaic approach to texture synthesis [18]. The idea is to create a large number of relatively small rectangular image pieces and paste them into random position in the output image. Unfortunately, this inevitably produces seams between individual texture patches and the technique either uses a fallback algorithm to fill in transition regions or simply blurs the seams. Effectively, chaos mosaics implicitly rely upon the visual masking effect of human vision; if there is significant high frequency content in the image, then seams produced by this method will tend to get lost among image details. Still, the fact that no attention is paid to make pieces line up better and that seams run in only vertically and horizontally makes the artifacts more noticeable. Many of examples presented in [18] involve brick walls which naturally provide the best masking situation for the algorithm. Using irregularly shaped (or even user-created, as in [14]) patches might help but this would complicate the algorithm and probably hurt the impressive speed of chaos mosaics while the basic misalignment problem will remain. More importantly, there is no clear way to control the results of the synthesis in chaos mosaics which is effectively a "write-only" algorithm in the sense that it does not examine either the input texture or the image it produces in any way.

3 New texture synthesis algorithm

We modify the WL algorithm to encourage verbatim copying of pieces of the input sample, attempting to solve the problem outlined above. We rely on visual masking to hide the seams between the patches. Pieces created by our algorithm have irregular shapes and are therefore more suitable for synthesizing natural textures.

The key observation we use is that at a given step during the WL synthesis process, we have already found pixels in the input sample with neighborhoods similar to shifted current neighborhood

in the output image. This information is not used in any way in the original algorithm; the search process starts from scratch at each new pixel, ensuring that the best available choice is made. However, it is reasonable to assume that pixels from the input sample that are appropriately "forward-shifted" with respect to pixels already used in synthesis are well-suited to fill in the current pixel (see Figure 4). For example, if a pixel two rows above and one column to the right of current position was taken from position (x=37, y=15) in the input sample, the new candidate is pixel (x=36,y=17) in the input sample. We use only pixels in input sample with neighborhoods completely inside the image as valid candidates. If the candidate is outside this valid region, we will replace it with a random valid position.

This approach tends to grow patches starting from some position in the input sample and continuing down in y direction to the bottom of the image (see Figure 5). A short horizontal edge which may appear at the boundary of such a patch if the synthesis process runs into the bottom of the input image. Such edges are usually well masked and do not present substantial visual artifacts. However, this might lead to a problem if there is a recognizable feature near the bottom of the input texture because it will appear in the output image more often than it should. Strictly speaking, such an input sample violates one of the underlying assumption of the method: stationarity of the texture. However, if necessary, we can artificially decrease the percentage of output pixels being taken from the bottom part of the input sample by replacing a candidate from near the bottom with a random valid candidate with probability related to the normalized distance $0 < d < 1$ to the lowest row of the valid region in the input sample. The condition $rand^4 > d$ for such replacement (where $rand$ is a random variable uniformly distributed on $[0, 1)$) works well in our test cases but frequent calls to the random number generator degrade the performance of the algorithm. In addition, the regions grown by the algorithm are naturally smaller in this case and since copying relatively large input image pieces is what we rely upon, the quality is also somewhat lower. We have not done extensive investigation of this issue since for most images (all images in this paper, for example) this technique is not necessary and no artificial constrains were put on the synthesis process.

An array is used to store input sample locations from which already assigned pixel were taken. The array is initialized to random valid positions before synthesis begins. Using this array, we create a list of candidates for each pixel to be synthesized and choose the best match among these candidates by comparing their neighborhoods in the input sample against the current one in the output.

The length of this candidate list could be a user specified parameter but we have found that a fixed number of candidates generated only by pixels from the L-shaped causal neighborhood works well in practice. This is justified by initial intuition that the size of the neighborhood corresponds to the size of the largest texture feature and outside this region the texture to large extend repeats itself (for more discussion of the neighborhood size see Section 5). In practice, however, this short list is often even shorter since our algorithm encourages copying of relatively large regions and the candidates generated by several different pixels in the L-shaped neighborhood can really be the same pixel in the input sample (Figure 4). Since comparing two neighborhoods is an expensive operation performed in the inner loop, it is worth the effort to remove the duplicates from the candidate list.

Our method can be extended in a straightforward way to create a multiresolution synthesis algorithm similar to the one described by Wei and Levoy. Using the multiresolution extension is important, for example, if the method is to be used for constrained texture synthesis as described in Section 5.1 of [16]. Otherwise it is of limited value in our method. We have implemented a multiresolution version of our algorithm and confirmed Wei and Levoy's

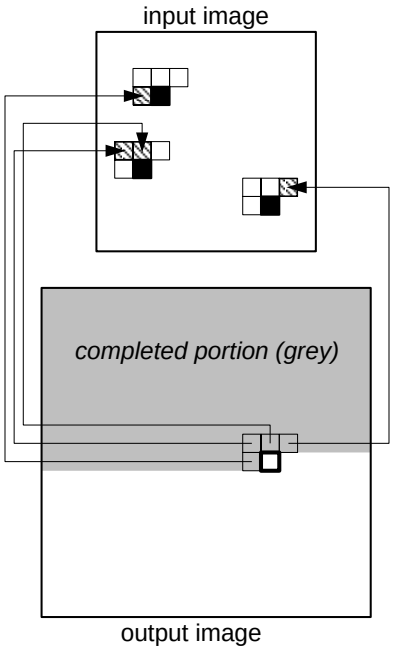

Figure 4: *Candidate pixels for our algorithm. Each pixel in the current L-shaped neighborhood generates a "shifted" candidate pixel (black) according to its original position (hatched) in the input texture. The best pixel is chosen among these candidates only. Several different pixels in the current neighborhood can generate the same candidate.*

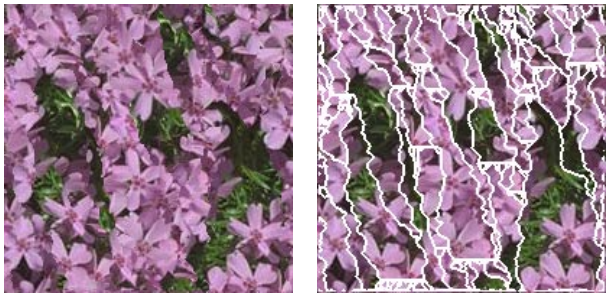

Figure 5: *Region-growing nature of the algorithm. Boundaries of texture pieces are marked white on the right. The input sample and WL result for this texture are shown on figure 3*

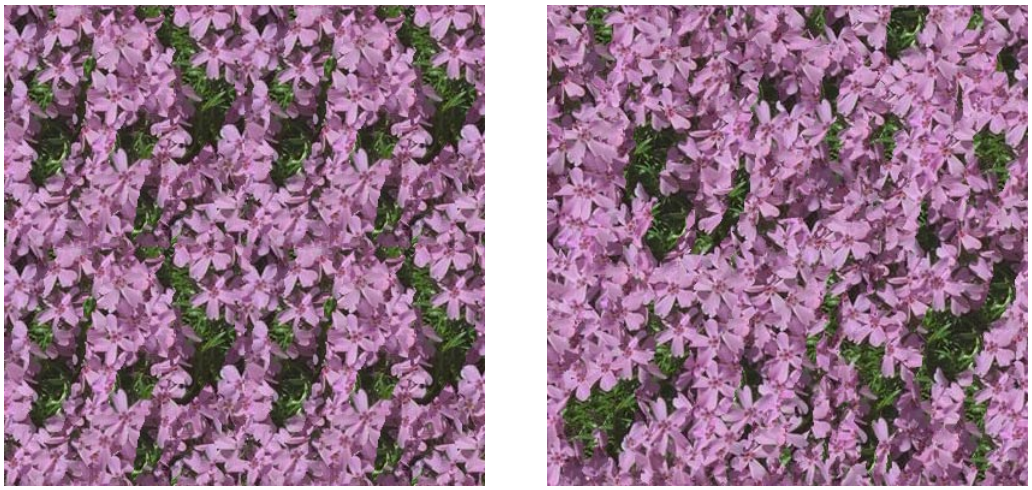

Figure 6: *Texture tiling vs. synthesizing larger image. Left: 200x200 synthesized texture from Figure 5 tiled vertically and horizontally. Right: a 400x400 synthesized texture.*

statement that the main benefit of using multiresolution synthesis is the "neighborhood expansion" effect: it allows to obtain the quality characteristic of a larger neighborhood while using a smaller one. Since using larger neighborhood is not necessarily better in our method (and rarely provides any benefits at all for the class of natural textures our algorithm is designed for, see discussion in Section 5), we are in favor of the faster and easier to implement single resolution version.

It is often desirable to have textures tile both vertically and horizontally. To ensure that the synthesized texture tiles horizontally we simply treat neighborhoods in the output image toroidally in the x dimension. However, since only casual neighborhoods are used, to improve vertical tileability we need to slightly modify the algorithm for the last $Neighb\_size_y/2$ rows of the output image. Here we consider all already assigned pixels in the toroidal neighborhood when computing L_2 norm and creating the candidate list instead of only pixels in the usual L-shaped region. After the synthesis is finished, we go over first $Neighb\_size_y/2$ rows of the output image again and apply the same synthesis procedure but now with the full square neighborhoods considered. In our experience, for the class of textures we are interested in, these measures are sufficient to mask any remaining seams between two copies of synthesized textures. If multiresolution synthesis is used, these extra steps might not be necessary since the neighborhood in this case will contain symmetric regions at lower resolution levels. Figure 6 shows that seams between two copies of the same texture are well masked but periodic large-scale structure due to tiling is easily identifiable. Our algorithm is fast enough to synthesize a texture of necessary size directly in most cases. Comparing the two sides of 6 shows that this solution is clearly superior.

We summarize our algorithm as follows:

- The array of original pixel positions is initialized to random valid positions in the input image
- For each pixel in the output image, in scanline order, do:
  - in the output image, an L-shaped neighborhood of current pixel of a specific (fixed) size is considered, see figure 4.
  - for each pixel from this neighborhood, use its original position in the input image (taken from the array) to generate a candidate pixel which location is appropriately shifted, see Figure 4.
  - remove duplicate candidates
  - search the candidate list for a pixel with a neighborhood most similar to the current L-shaped neighborhood in the output image
  - the current pixel value in the output image is copied from the position in the input sample identified as the most similar by this search and this position is recorded in the array of original positions.
- if necessary, modify the algorithm for the last few rows and go over the top few rows again to improve tileability of the result texture.

4 User Control

The basic algorithm described in the previous section has substantial value for synthesizing natural textures. However, its utility can be significantly improved if an intuitive way to control the result image is provided. We believe that the simplest and most general way for a user to control the output of a texture synthesis algorithm is to provide a target image which would show how the result should look. Of course, this target should only outline some general properties of the result leaving it up to the synthesis algorithm "to fill in the details". The source of this target image can be completely arbitrary, e.g., a hand drawn picture, a photograph, or a computer generated rendering.

Consider the flower texture example on the top of Figure 10. One might want to tell the algorithm that the result texture should have more flowers in the top part of the image and more open grass in the bottom half. A natural way for a human to give this information to the algorithm is to create a simple image on which the top half has more of the color of the flowers (pink in this case) and the bottom has more green, as shown in the second column of Figure 10. We would also like to have some control over how much the instructions are followed: one might want to have the result conform rather strictly to the target (to simulate a pattern made of flowers in a garden) or get only a general result of the type "there is more grass in the bottom part of the texture than in the top one". Fortunately, this type of control is easily achieved with the extension of our method we will now describe.

If we have a target image as a part of an input to the algorithm, the first change required is that during the synthesis we consider the

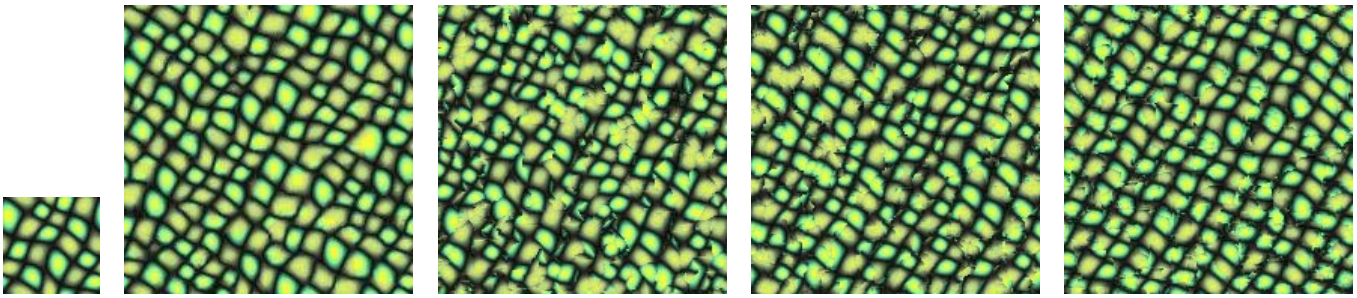

Figure 8: *Effect of the neighborhood size for a smooth texture. Input texture size is 64x64 pixels, all results are 192x192. Left to right: input sample, WL result, and our results with 5x5, 9x9 and 21x21 neighborhoods.*

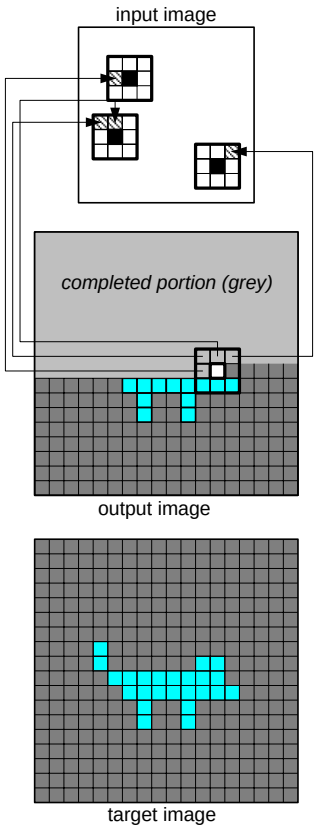

Figure 7: *User controlled version of the algorithm (first pass). Candidates are chosen as before but a square neighborhood in the input is now used to choose among them. This square of pixels is compared with a special neighborhood, top L-shaped half of the which is taken from the output image while the bottom half is from the target image (with only pixel which received user input being considered). The target image can be thought of as is being overwritten as the synthesis progresses. For additional passes the procedure is the same but candidates are created based on the full square neighborhood.*

complete square neighborhood of a pixel instead of the L-shaped casual one (Figure 7). The candidates in the input image are found as before, based only on the already assigned pixels in the causal L-shaped top part of the neighborhood. Computing the value used to choose the best candidate proceeds now in two steps. Fist, as in the basic algorithm, pixel-by-pixel L_2 difference of the top L-shaped parts of the neighborhoods in the input and output images is computed. We then add the L_2 difference between the pixels in the bottom part of the candidate neighborhood in the input texture and pixels in the target image which location corresponds to bottom-L half of the output neighborhood under consideration. Only pixels which received user input are considered valid at this second stage which makes the algorithm identical to the basic one for parts of the output left blank by the user.

For some cases a single pass of the described algorithm is enough to create a texture which sufficiently conforms to the target. If a greater similarity is desired, one or more further iterations of the algorithm can be performed with each next iteration using the result of the previous one as the starting point of the synthesis. That is, instead of initializing the array of candidates to random valid locations at the start of an iteration, we use the information computed at the previous pass. During these additional passes the candidate list is created based on full square neighborhood. One could try to terminate this process automatically by providing, for example, a value for the L_2 difference between the target and the result and repeat the algorithm until the real difference is below this value. Although in our experience the process is subjectively monotonic, i.e. with each iteration the similarity between the synthesis result and the target increases, the exact behavior varies dramatically depending on the particular texture, and the global L_2 difference is not intuitive and is not expected to be a good measure of visual quality. We believe that the best way is to simply show the user the result after some number of iterations (usually after each one) and let the user judge when to stop the process.

We have created a simple painting-style user interface to allow us to sketch the target image using the colors picked from the input texture and pass this information to the synthesis routine. Our algorithm is fast enough to allow multiple iterations to be performed while keeping the total time of a single synthesis session quite reasonable. Depending on the input texture, target image, and the degree of conformity desired we used from one to about twenty passes. This interface is useful even for the basic version of the algorithm: since the results of our approach, as these of any other known texture synthesis method, are far from perfect on many textures, we can run the synthesis routine several times with different random seeds, and choose the best output. Of course, the target image does not have to be hand drawn or strictly conform to the colors of the input texture, as demonstrated by a "shadow of a teapot flying over the forest" in Figure 10, second row.

Figure 9: *Texture synthesis results. Right column: input texture sample, middle column: WL results, right column: our results. Input textures are 192x192 (left, VisTex textures [1]) or 128x128 (right) pixels. All results are 200x200 pixels.*

5 Results and Discussion

Some synthesis results are shown on Figure 9 along with WL-generated images. One can see that the algorithm achieves good results for the class of natural textures it is designed for, as shown on the top half of Figure 9. In most such cases the results are at least as good as those from the WL algorithm, and for some textures the difference is quite dramatic. There are situations when both algorithms do not do a very good job (Figure 9, fourth row), but the types of artifacts are quite different: excessive blurring for the WL algorithm and "too rough" images for our algorithm. Depending on the application, one or another might be preferred, but our interactive extension allows to give some "help" to the algorithm, as discussed below. In addition, information necessary to find pixels near boundaries between individual texture patches is readily available in our algorithm (Figure 5) and, if desired, some selective blurring can be applied to these pixels to help hide the seams. This has not been done for any of the images presented in this paper.

We do not intend our algorithm to be a general-purpose one and it should be kept in mind that there are many natural textures which do not belong to the class we discuss here. Important examples are clouds and waves which are too smooth for our algorithm. In some such cases the quality can be somewhat improved by increasing the neighborhood size, see Figure 8, in others it is simpler to just use the original WL algorithm (Figure 9, bottom row).

Results for user-controlled synthesis are shown in Figure 10 (see color plate). Two different examples created from the same target are shown to illustrate the effect of the number of iterations on the output image. Note that additional passes tend to "flatten" regions in the final image corresponding to single color areas in the target. Clearly, for some applications one would prefer only a few passes while to create more pronounced pattern more iterations might be required. Third row of Figure 10 shows an interesting example of how one could "help" the algorithm to discover large-scale structure in the underlying texture. Compare this result with the basic version shown in Figure 9. Note also that the period of the grid is controlled by the user input rather than by the original texture. With more attention to the details of the input texture one can create more sophisticated effects. The bottom row of Figure 9 shows an example of this technique. A videotape demonstrating how some of these images can be produced is also included.

Our approach is simple to use because the neighborhood size is the only user-specified parameter in our algorithm. Its effect on the quality of the output is shown on Figure 8. This size plays a somewhat different role in our method from that in the original WL algorithm. In the WL algorithm it is necessary to keep this size large enough to capture the low frequency component of the texture. Since we compare neighborhoods only among a few well-chosen candidates, the size need only be large enough to reliably distinguish the best choice among them. Intuitively, since we have many fewer options to choose from, we do not need fine discrimination and the neighborhood sufficient for this task can be significantly smaller than that needed to produce good results with the WL algorithm. In fact, large neighborhoods can be harmful sometimes since it may encourage too much region growing by always giving preference to a particular expansion pattern. This would prevent regions from stopping growth resulting in large repeating areas with little variation (Figure 8, far right). In general, the smoother the texture is, the larger the neighborhood is needed for perceptually better results. This is to be expected since the number of candidates for a given pixel is also controlled by the neighborhood size and more candidates are generally needed to ensure unnoticeable transitions between image patches in a texture lacking high frequency components which we rely upon to mask the edges of the pieces. For such smooth textures our algorithm often fails to produce results comparable to the WL algorithm but, surprisingly, even in such unfavorable cases the quality can be sufficient for some applications (two textures on the right of Figure 8). Of course, one should keep in mind the performance costs of increasing the neighborhood.

We have found that uniformly good results for our class of textures are obtained with a 5x5 neighborhood (5x2 L-shaped region). All images shown, except figure 8, were created with this size and it is recommended as a good default value. Note that most of textures in the original Wei and Levoy paper required a significantly larger neighborhoods, at least 9x9 pixels. This ability to use smaller neighborhoods is one of the two main sources of efficiency in our algorithm. The other, and more important, reason is the severe restriction on the algorithm's search space. We create only at most a few dozen (and more often only one or two) candidates instead of considering the complete space of many thousands pixels. This makes acceleration techniques such as TSVQ used in the WL algorithm unnecessary for our method. In addition to simplifying implementation (our code is about 300 lines of C code, excluding I/O and user interface routines), it removes training overhead of TSVQ and makes the algorithm independent of the input sample size.

These two factors together increase the speed of the algorithm substantially. We use the same architecture as Wei and Levoy (195 MHz R10000 processor) to measure the performance of the algorithm. TSVQ accelerated WL runs on the order of 20 to 30 seconds to synthesize a 200x200 texture. Our timing for the first pass of the user controlled version of the algorithm is only about 0.3 seconds per iteration for a texture of this size and about 1.5 seconds for a 500x500 image. Additional passes take about twice as long (2.8 seconds) since the candidate list is now based on full neighborhood. The run time is slightly better (1.3 seconds) for the basic (no user control) version of the algorithm since it deals with only two images instead of three (input, output and target) and works with L shaped neighborhood which is a subset of the square one needed in the user controlled version. Although these timings are not in the real-time domain, they are quite acceptable to provide interactive feedback during user-controlled synthesis process described in Section 4.

Unfortunately, regardless of its efficiency, our algorithm is not well-suited for temporal texture synthesis in most cases. The problem is that most motions one would want to synthesize are rather smooth and do not contain any significant amount of high frequency component in the temporal domain which we would need to mask transitions between separate time-space patches generated by our method. Our experiments show that visually unsatisfactory results are produced by a direct extension of our algorithm to this case. Modifying the algorithm for this situation is an interesting potential area of future research. Other possibilities are discussed in the next section.

6 Conclusions and Future Work

Most texture synthesis algorithms attempt to deal with as large class of textures as possible. This generality is certainly desirable but we are not hopeful about the discovery of such an ideal algorithm. There is a substantial interest in developing of a number of methods to generate textures of a particular well-defined type with high quality and efficiency. This paper is one of relatively few attempts to design an algorithm which is not expected to work well for an arbitrary texture but is well-suited for a particular important class of textures. Other work similar to ours in this sense includes reaction-diffusion textures [17] and a specialized method for generating stone wall patterns [10]. The particular class of textures this paper deals with includes many naturally occuring quasi-repeating patterns consisting of familiar elements, such as pebbles, leaves, bushes, flowers, tree branches, etc. Our technique is simple, easy to implement, efficient and produces good visual results. These qualities allow it to be extended to handle direct user interaction with the

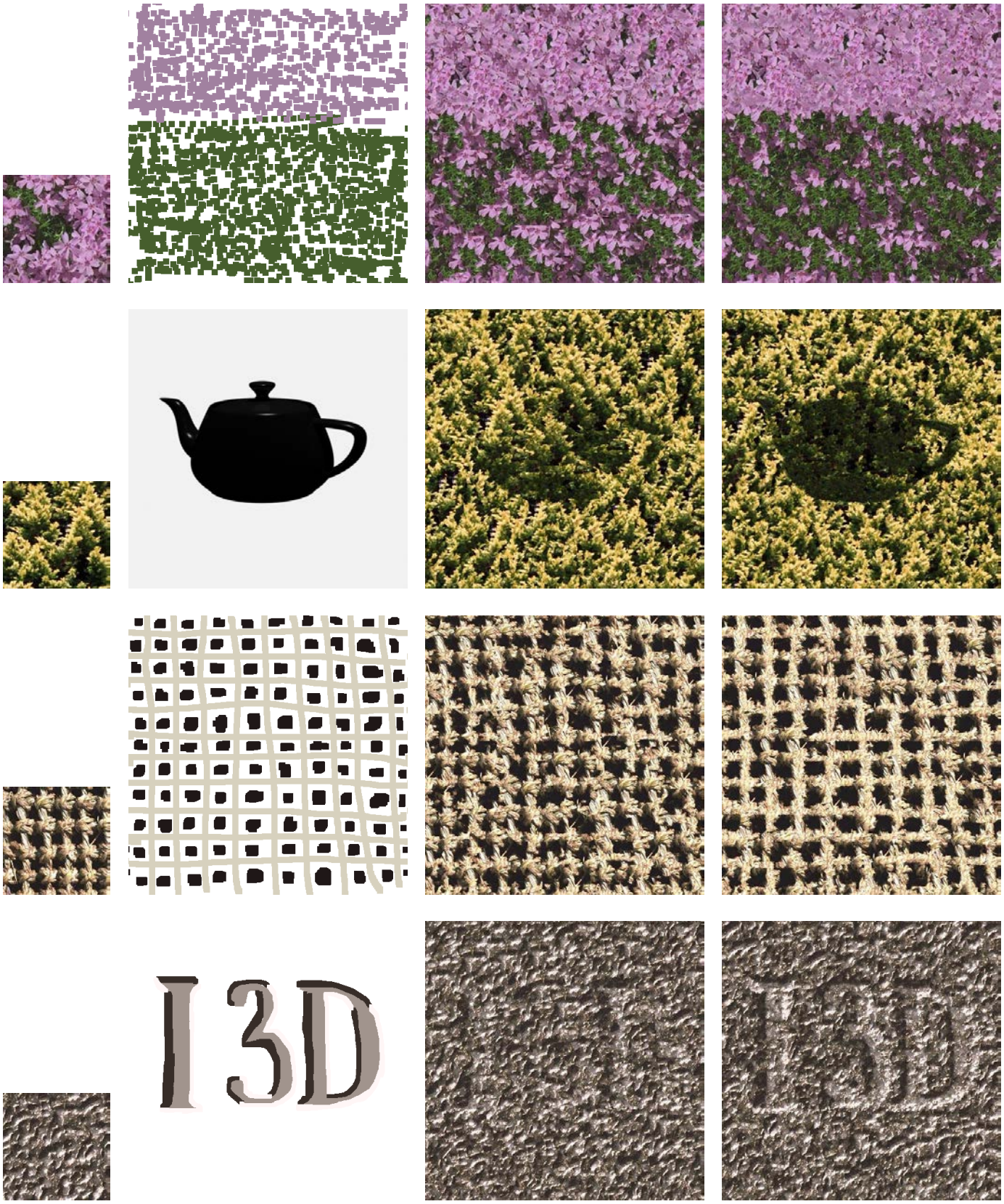

Figure 10: *User controlled texture synthesis results. All input textures are from VisTex dataset [1], 192x192 pixels. All other images are 500x500 pixels. White regions in target images did not receive user input. Left to right: input sample, target image, single pass result, result after more iterations (top to bottom: 5,5,8,20 total passes)*

synthesis process and create textures based on a user-created input image. The technique used to take into account user input is not unique to our algorithm and can probably be used with the original WL algorithm once processor speeds make the iteration of the WL algorithm faster.

It might be possible to create similarly simple and efficient algorithms for other texture classes in the future. In particular, an algorithm for "smooth" time-dependent natural textures, such as waves, clouds, smoke, etc. would be very useful. Another interesting class is textures with underlying regular structure or containing regular features. Neither the WL algorithm nor our algorithm make any attempt to extract such structure from the input texture and as a result, perform relatively poorly on them unless the neighborhood size is increased substantially.

There are many other problems to be solved to make texture synthesis more practical. Like most current methods, our algorithm uses input samples directly and does not attempt to single out effects due to illumination and/or surface geometric structure. If this problem is solved reliably, one could design an algorithm capable of synthesizing view- and illumination dependent textures [4]. Another important area of future work is direct synthesis of texture over 3D geometry.

Finally, we believe that it would be beneficial to invest work into theoretical analysis of simple texture synthesis algorithms to gain more insight into their surprisingly good performance. Aside from intrinsic scientific interest, this might give us an understanding of image characteristics which make photographic images so much better than most computer-generated images. This, in turn, would help to design new techniques (either procedural or image-driven) to increase visual richness of computer graphics imagery.

Acknowledgements

The author would like to thank Peter Shirley for help preparing the paper, Bruce Gooch for help preparing the video, and Li-Yi Wei for making his texture synthesis result imagery available online. This work was supported by NSF awards 97-31859, 97-20192 and 89-20219.

References

[1] Vision texture library. *MIT Media Lab*. http://www-white.media.mit.edu/vismod/imagery/VisionTexture/.

[2] BLINN, J., AND NEWELL, M. Texture and reflection in computer generated images. *Communications of the ACM 19*, 10 (October 1976), 542–547.

[3] BOLIN, M. R., AND MEYER, G. W. A perceptually based adaptive sampling algorithm. *Proceedings of SIGGRAPH 98* (July 1998), 299–309.

[4] DANA, K., VAN GINNEKEN, B., NAYAR, S., AND KOENDERINK, J. Reflectance and texture of real world surfaces. *ACM Transactions on Graphics 18*, 1 (January 1999), 1–34.

[5] DEBONET, J. S. Multiresolution sampling procedure for analysis and synthesis of texture images. *Proceedings of SIGGRAPH 97* (August 1997), 361–368.

[6] DISCHLER, J.-M., AND GHAZANFARPOUR, D. Interactive image-based modeling of macrostructured textures. *IEEE Computer Graphics and Applications 19*, 1 (January-February 1999), 66–74.

[7] EFROS, A., AND LEUNG, T. Texture synthesis by non-parametric sampling. *International Conference on Computer Vision 2* (September 1999), 1033–1038.

[8] HARALICK, R. Statistical image texture analysis. In *Handbook of Pattern Recognition and Image Processing*, vol. 86. Academic Press, June 1986, pp. 247–279.

[9] HEEGER, D. J., AND BERGEN, J. R. Pyramid-based texture analysis/synthesis. In *Proceedings of SIGGRAPH 95 (Anaheim, California, August 6–11, 1995)* (August 1995), R. Cook, Ed., Computer Graphics Proceedings, Annual Conference Series, pp. 229–238.

[10] MIYATA, K. A method of generating stone wall patterns. *Computer Graphics 24*, 3 (August 1990), 387–394. ACM Siggraph '90 Conference Proceedings.

[11] PERLIN, K. An image synthesizer. *Computer Graphics 19*, 3 (July 1985), 287–296. ACM SIGGRAPH 85 Conference Proceedings.

[12] PERLIN, K., AND HOFFERT, E. M. Hypertexture. *Computer Graphics 23*, 3 (July 1989), 253–262. ACM SIGGRAPH 89 Conference Proceedings.

[13] PORTILLA, J., AND SIMONCELLI, E. P. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision 39*, 3 (October 2000). to appear.

[14] PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. Lapped textures. *Proceedings of SIGGRAPH 2000* (July 2000), 465–470.

[15] RAMASUBRAMANIAN, M., PATTANAIK, S. N., AND GREENBERG, D. P. Perceptually based physical error metric for realistic image synthesis. *Proceedings of SIGGRAPH 99* (August 1999), 73–82.

[16] WEI, L.-Y., AND LEVOY, M. Fast texture synthesis using tree-structured vector quantization. *Proceedings of SIGGRAPH 2000* (July 2000), 479–488.

[17] WITKIN, A., AND KASS, M. Reaction-diffusion textures. In *Computer Graphics (SIGGRAPH '91 Proceedings)* (July 1991), T. W. Sederberg, Ed., vol. 25, pp. 299–308.

[18] XU, Y., GUO, B., AND SHUM, H.-Y. Chaos mosaic: Fast and memory efficient texture synthesis. Tech. Rep. MSR-TR-2000-32, Microsoft Research, 2000.