Yale University
Department of Computer Science

Creating Texture Exemplars from Unconstrained Images

Yitzchak David Lockerman Su Xue Julie Dorsey
Holly Rushmeier

YALEU/DCS/TR-1111
October 2013

Abstract

Texture is an essential feature in modeling the appearance of objects and is instrumental in making virtual objects appear interesting and/or realistic. Unfortunately, obtaining textures is a labor intensive task requiring parameter tuning for procedural methods or careful photography and post-processing for natural images. Many texture synthesis techniques have been developed to generate textures of arbitrary spatial extent, but these techniques require the user to first produce an exemplar consisting solely of the desired texture. We present a fast method using diffusion manifolds to locate textures in unconstrained photographs, and extract exemplar tiles. The method requires the user to only specify a single point within the image containing the desired texture and the scale of the desired texture. The user may tune the result using simple interactions. The method is non-local, in the sense that the desired texture does not have to appear in a single contiguous region in the source image. Supplemental material and an interactive demonstration is available from the paper's companion website [1].

Figure 1: A photograph, sample, tiles extracted from the photograph and texture synthesis from the texture tile sets.

1 Introduction

The ability to model visual textures is essential in any computer graphics modeling system. Yet preparing textures is still a labor intensive task. In current practice users obtain textures by tuning parameters in procedural models [2] or by using textures from prepared collections, either commercial or free [3]. With procedural textures, in addition to the effort required to set parameters, there is no guarantee that a particular model is capable of representing a desired texture. With prepared texture collections, the user has a limited choice and may either have to pay a fee or be restricted by intellectual property concerns in subsequent use of the content.

One approach to allow users to create their own textures has been the development of Markov Random Field (MRF) techniques which generate textures of arbitrary extent from a small patch texture [4]. This patch, called an exemplar, is then used as the input of a texture synthesis algorithm to create an image of the texture with the required size.

This process fails when a source image contains details other than the single desired texture. While there has been some work done removing regions of undesired content within an image dominated by one texture [5], we know of no automatic method of extracting a texture exemplar from an arbitrary natural photograph without extensive user input. The difficulty of preparing exemplars is a barrier to the use of texture synthesis algorithms.

In this paper we simplify the process of preparing a texture exemplar by providing a method that automatically extracts a desired texture from an image using minimal and intuitive user input. Our technique is based on heat diffusion on a graph, inspired by diffusion manifolds [6].

We achieve orders of magnitude faster texture extraction by focusing only on the texture specified by the user, and by representing a texture tile with a fixed-size feature vector. Our primary output is the collection of texture tiles. The tiles can then be used by any patch-based MRF technique to generate a texture of arbitrary extent.

Our technique is used in a system with a simple interface. In the system the user specifies the scale of the texture of interest within the source image. After a short (on the order of seconds) preprocess, the user can generate texture tiles interactively by clicking on a point in the source image. The set of tiles can be refined interactively, if desired, by selecting additional points, rejecting tiles, and adjusting a single parameter. A texture of arbitrary extent can be generated from these tiles using any of a variety of a Markov random field (MRF) techniques. The final result of this process can be seen in Figure 1.

Specifically, our contributions are:

- a novel click and slider based interface for easy texture selection.
- a texture extraction algorithm that is orders of magnitude faster than previous methods.
- an examination of several different feature spaces that can be used in conjunction with our algorithm.

2 Related Work

Our work serves as a technique that is a preprocess for existing texture synthesis techniques. Our technique uses the idea of diffusion distance manifolds to identify homogeneous texture regions. We analyze previous work in this area, and identify how this work can be modified to achieve orders of magnitude speed-up, thus facilitating an interactive texture extraction system.

2.1 Texture Synthesis

Several techniques for texture synthesis from image exemplars appeared in computer graphics in the 1990's, including Heeger et. al. [7] and De Bonet et. al. [8]. The last decade has seen an explosion in texture synthesis algorithms Wei et al. [4] provide an overview of recent advances. In particular there have been remarkable strides in MRF-based methods.

Some work has been done to improve an existing exemplar. The Inverse Texture Synthesis (ITS) technique [9] starts with a globally varying texture image and accompanying control map and produces a more compact exemplar. Unlike the problem considered here, ITS does not extract an exemplar from an unconstrained natural image.

Eisenacher et al. [10] present a method for extracting textures from images, even to remove the effects of non-planar and affine transformed surfaces. However, their technique required a large amount of user input to locate the textures. Our work is focused on minimizing the necessity for that user interaction.

2.2 Diffusion Distance Manifolds

Manifolds constructed using diffusion distances between nodes is a robust classification technique [6]. A diffusion manifold measures how many different paths "heat" can use to progress from one point to another. This

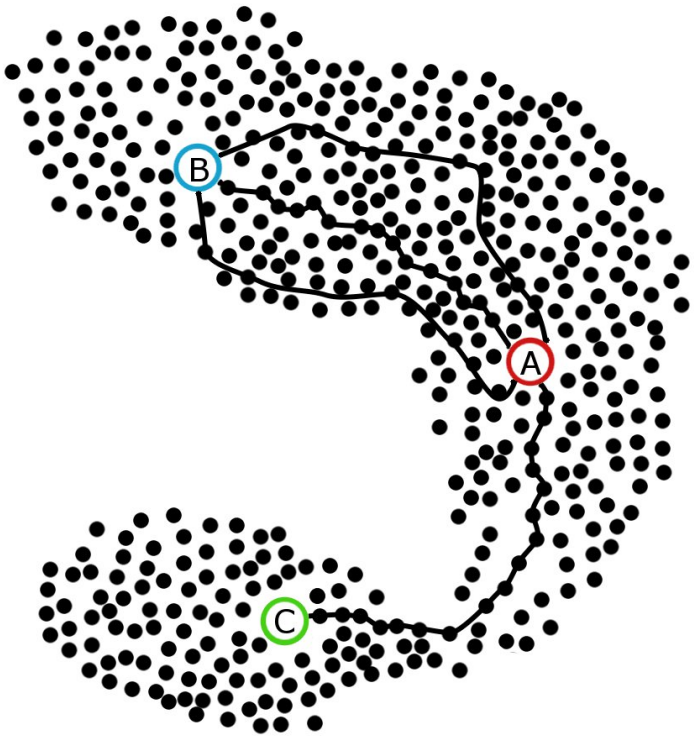


Figure 2: An example of using a heat diffusion to classify points. Each dot represents a feature in a 2D feature space. While the geodesic distances from A to B and A to C are similar, A is closer to point B than C in diffusion distance. This can be seen by the number of paths from A to B; three of which are shown.

leads to a classification that is robust in the presence of noise, as incorrectly placed points change only a few paths.

A manifold is created by forming a weighted graph between nodes representing feature points. An edge is created between each pair of nodes and given a weight of $e^{-||x_i - x_j||^2/\epsilon^2}$ where $||x_i - x_j||$ is the Euclidean distance between the two feature points and ϵ is a constant. Each node can be thought of as a location where heat can rest, and the value of each edge can be thought of as the "conductance" of heat from one node to another. Each node also has an edge, with weight 1, to itself. Finely, this graph is normalized so that heat is conserved using the formula (where $\widetilde{W_{i,j}}$ is the normalized graph weights, and $W_{i,j}$ are the normalized weights)

$$W_{i,j} = \frac{\widetilde{W_{i,j}}}{\sum_{i=1}^{n} \widetilde{W_{i,j}}}$$

Figure 2 illustrates the concept of diffusion distance. The geodesic distance within the manifold from A to C is a bit shorter than the distance from A to B. However, there are many more possible paths from A to B within the manifold, making the diffusion distance from A to B much shorter.

Diffusion distance is measured using heat diffusion on the graph. Some units of heat are placed at a starting node and the heat is allowed to spread. At each step, the heat spreads out across the edges, distributed proportionately to the weight of each edge. More formally, if $W_{i,j}$ is the weight between i and j and $h^t(x_i)$ is the heat at i at time t then the heat propagates by the heat equation

$$h^{(t+1)}(x_i) = \sum_j W_{j,i} h^t(x_j) \tag{1}$$

Define $h_i^t(x_j)$ as the amount of heat on node j at time t if the initial distribution consists of a single unit of heat at node i. The diffusion distance on the manifold is formally defined by $d^t(i,j) = \sum_k (h_i^t(x_k) - h_j^t(x_k))^2$.

If the application does not need a measure that is strictly a metric, the function h_i^t itself tends to provide a good objective function for determining how 'close' a node is to i. We will use this approach here; that is we will use graph diffusion, not the diffusion distance. This has a large performance benefit, as methods that use the actual diffusion distance tend to need an iterative eigenvector solver.

We describe a few of h_i^t's relevant properties. First, the amount of heat is conserved, or $\sum_k (h_i^t(x_k))$ is constant. Accordingly, $h_i^t(x_k)$ is interpreted as the probability of heat being transported from node i to node k in t steps. Large values of $h_i^t(x_k)$ indicate that node k is "closer" to node i. $h_i^t(x_k)$ can be visualised by arbitrary scaling the values and embedding it in a grayscale image, or "heat map."

In recent work Farbman et al. [11] use diffusion maps for classifying images regions to make editing tools sensitive to edges. The focus is on editing the original image, not extracting texture information. In addition their use of diffusion maps differs from the method we present here in two key ways. First, their algorithm involves complex linear algebra while ours iterates a simple difference equation (Equation 1), or equivalently, just a simple matrix-vector multiplication. Second, our feature set uses information from

multiple scales which takes into account information in a neighborhood of a pixel, while their feature set only uses the color of individual pixels. Since we are identifying texture tiles, rather than specific edges to limit editing operations, our application does not have to be as precise in its classification.

2.3 Dominant Texture Detection

Lu et al. [5] used a manifold built with diffusion distances to detect images that were overwhelmingly covered, or dominated, by a single texture. In particular, they used $h_i^t(x_j)$ to decide if two tiles centered at pixels i and j belong to the same texture. They were then able to mask the parts of the images that were not part of that dominant texture, allowing the texture to be used as an exemplar. Lu et al. [12] subsequently conducted a psychophysical test to determine if the dominant textures they detected matched users' expectations for extracted texture. The study suggested that their dominant texture detection method based on diffusion distances performed better than other approaches, including normalized-cut texture segmentation [13]. Their work was the original inspiration for our method. However, we present a substantially modified, more efficient algorithm and implementation for extracting texture from arbitrary images rather than images dominated by a single texture.

First, assuming that the image was dominated by the texture of interest, Lu et al. began with a Fourier analysis to determine the appropriate texture tile size. In our method, we replace this analysis with a simple interface that allows the user to specify the scale and a sample location of the texture of interest in an image that may contain multiple texture regions.

Next, Lu et al. represents a texture tile as a vector of length n^2 for a tile of $n \times n$ pixels. This results in n^2 calculations to compare each pair of tiles to determine the edge weights. In our work we replace this time consuming scale dependent representation with a feature vector that is independent of tile size. There are many possible feature vectors that many be used, and we experimentally compare several different candidates to find an effective one.

Lu et al.'s [5] method continues then to find the values $h_i^t(x_j)$ that approximate the distance between tiles i and j. To reduce the memory required for this process, an Approximate Nearest Neighbor (ANN) algorithm was used to find a sub-graph that approximates the diffusion property of the original manifold, in place of the graph relating all tiles. In our method, we also use ANN to simplify the process. We make a major improvement over Lu et al.'s method in that we do not estimate the distances between all tile

pairs. We exploit the fact that we know the location of a texture tile the user is interested in, and find the estimated distances only to that tile. This reduce the calculation from being quadratic in the number of tiles to linear.

3 The Texture Exemplar Pipeline

Our method extracts textures from images in the following steps:

1. The user specifies the scale of the desired texture using a slider with immediate visual feedback.
2. The image is divided into overlapping tiles and a size-independent feature vector is computed for each tile.
3. A KNN graph is computed for the tiles.
4. The user selects a point in the image containing the desired texture.
5. The system selects tiles to extract based on the heat distances from each tile to the selected tile.
6. If the user desires, the user specifies additional good and bad points or rejects tiles. The system uses this information to refine the extracted tiles in real time.
7. A texture synthesis algorithm is run in the background. As it progresses the user is shown intermediate results, allowing the user to quickly evaluate the resulting texture.

Figure 3 outlines this pipeline. A video of the process is provided as supplemental material [1].

3.1 Defining Texture Scale

A natural image frequently contains multiple textures at many spatial scales. This is an unavoidable problem, as many textures have details that themselves are made up of textures. For example, a brick wall contains a texture that represents the wall pattern and each brick itself has a brick texture. For these situations some sort of user interaction is needed to inform the algorithm which texture scale the user desires.

To solve this problem we present the user with an interface (see Figure 4) with a slider that adjusts the scale parameter. As the parameter is changed,

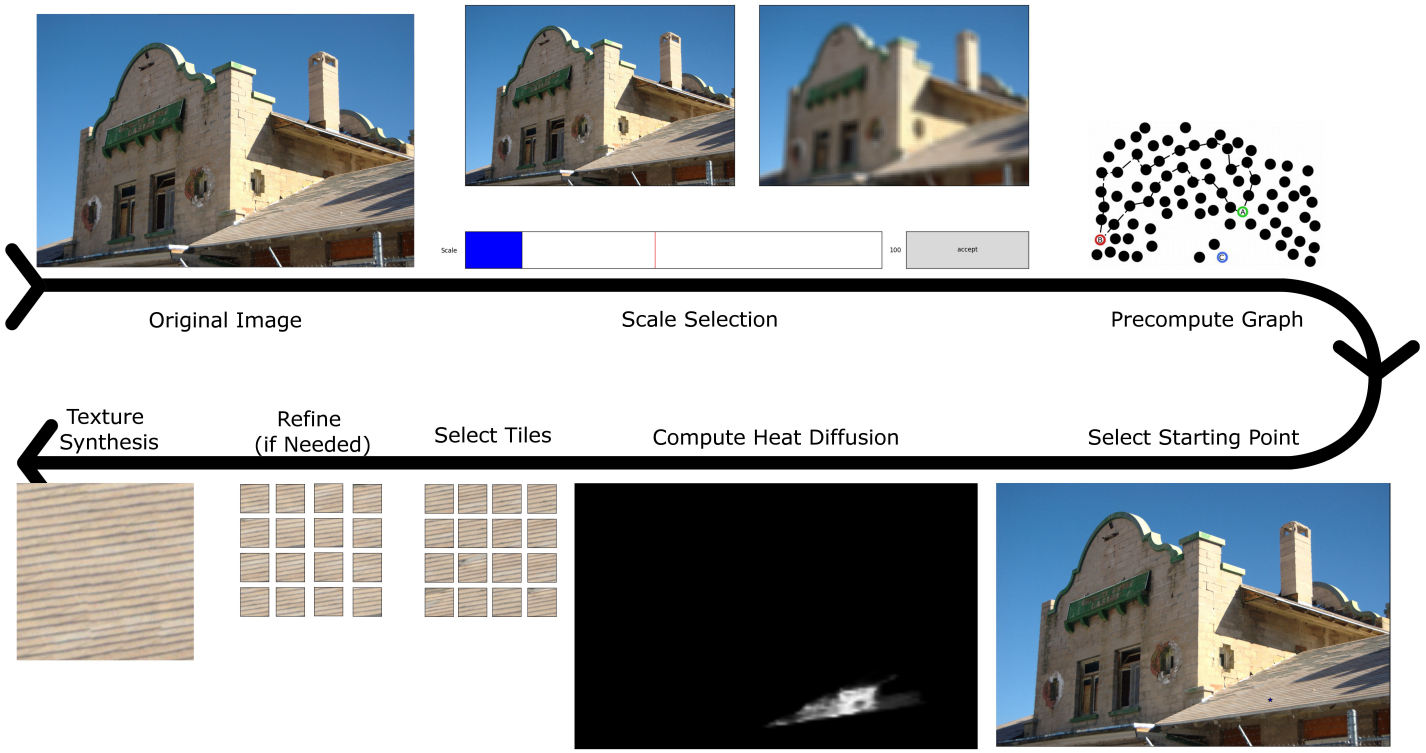


Figure 3: The steps in the pipeline.

the user is shown a blurred version of the image. This is done by dividing the image into tiles, then replacing each with its average color.

The user is asked to find a scale where the details of the texture are no longer visible. Equivalently, they are asked to identify the level where the desired texture tiles first appear as single color pixels.

Figure 5 demonstrates the effect of analyzing the image at different scales. Larger scales include details of the brick wall, while smaller scales are mostly limited to the details of the individual bricks.

3.2 The Diffusion Graph

We begin constructing the graph by dividing the image into overlapping tiles. We then create a feature for each of the tiles (as discussed in the next section) and then create a k-nearest neighbor (KNN), or approximate nearest neighbor, graph based on the features. In our implementation, we fix k at 128. When the feature space has less then 16 components we use the ANN Library [14] to find the approximate nearest neighbors to an accuracy of 1%. When there are 16 or more components in the feature vector we switch to a brute force GPU implementation to calulate the exact KNN.

We are then able to create a diffusion matrix, W, as follows. First, we compute ϵ, the root mean square of the distance between each node and the its nearest neighbor, that is $\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n} ||x_i - x_{nearest(i)}||^2}$. We then create a normalized matrix with the formula

$$\widetilde{W_{i,j}} = \begin{cases} e^{-\frac{||x_i - x_j||^2}{2\epsilon^2}} & x_i \text{ and } x_j \text{ are neighbors} \\ 0 & o.w. \end{cases}$$

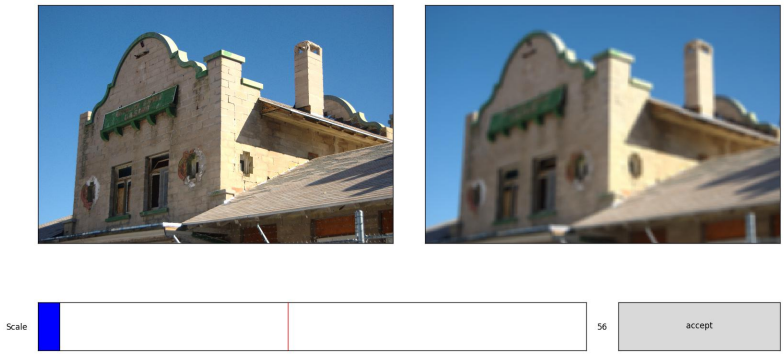
and finally create a normalized diffusion matrix by the standard formula

$$W_{i,j} = \frac{\widetilde{W_{i,j}}}{\sum_{i=1}^{n} \widetilde{W_{i,j}}}$$

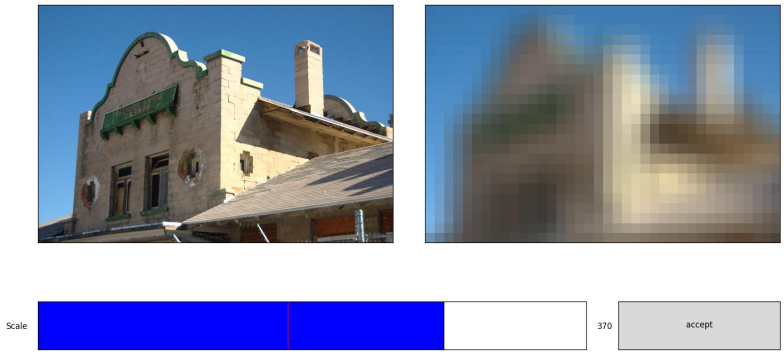
chosen to conserve energy.

3.3 Identifying Texture Tiles

Once the manifold is created, it can be used to select texture tiles on the fly. By focusing on the user's selection, rather than trying to classify all textures in the image into classes as done in previous work, we introduce



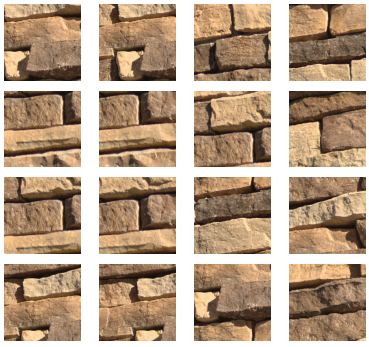
(a)



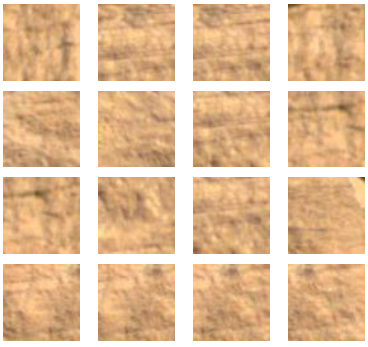
(b)

Figure 4: Examples of the scale selection interface. The user is able to adjust the slider on the bottom of the window which changes the blur of the figure on the right. When a smaller scale is selected more details are visible (a). When a larger scale is selected, fewer details are visible (b). Users are instructed to select a scale in which the details of the texture are not visible but the location of the texture is distinguishable from other locations.

(a)



(b)



(c)

Figure 5: Demonstration of the algorithm's ability to work on multiple scales. Both tile sets were created from the same image, but from different scales. (a) is the original image, (b) is an image where we selected the scale of the brick wall, while in (c) a smaller scale was chosen to get the rough surface of the bricks themselves.

another major efficiency. Our calculations compare each texture tile to the selected tile only, rather than computing distances between all tiles pairwise.

Specifically, when the user clicks on a point, i, within the image $h_i(x_j)$ is calculated for the entire manifold. This is done by setting the value of the node i to 1 and setting the value of the rest of the nodes to 0 then iterating Equation 1:

$$h^{(t+1)}(x_i) = \sum_j W_{i,j} h^t(x_j)$$

It is important to note that since $W_{i,j}$ is sparse, the complexity of the above equation grows linearly with the number of tiles making the diffusion tractable even for large images.

Once the values of $h_i(x_j)$ are determined, tiles corresponding to the selected texture are found by simply taking a pre-set number of tiles with a maximum value of $h_i(x_j)$.

3.4 Refinement

The set of selected tiles are displayed for the user. In cases where a tile appears to include some unwanted feature, the user can provide input to the algorithm by clicking on those tiles. This in turn modifies the initial heat map by decreasing the initial value of $h_i(x_j)$ corresponding to the selected tile by 1. This makes the tile a "sink" for heat, making it and similar tiles less likely to appear on the output. The user can also create sinks by right clicking on the image. However, care must be taken as too many sinks can cause the system to ignore the good tile.

Alternatively, the user can add more source points to the diffusion process by clicking on additional points in the image. This in turn increments the initial value of $h_i(x_j)$ corresponding to the selected tile by 1.

Finally, the set can be adjusted by adjusting the value of steps t used in iterating the heat equation.

The user does not need to understand the mathematical meaning of these interactions. The updated results appear quickly, and the user simply adjusts the parameters to taste. It should be noted that experimentally the original result is often satisfactory and adjustments are not needed.

3.5 Exemplar Generation

It is possible to modify most MRF algorithms to use this tile set as an exemplar for generating a texture of arbitrary extent, instead of a single

image. MRF texture synthesis algorithms are now extremely well studied, and many algorithms exist for many different possible situations [9].

In our implementation, we chose to use a basic image quilting technique [15]. Each time the tile set is changed the quilting algorithm is spawned as a background process. As the synthesis algorithm progresses, the user is shown intermediate results. This allows for texture extraction in real time, without needing to wait for the synthesis algorithm to complete.

4 Feature Space Selection

In this section we discuss a number of possible feature vectors that can be used to characterize texture tiles, and describe an experiment we conducted to evaluate them.

4.1 Types of Feature Vectors

There are an infinite number of possibilities for feature vectors. We experimented with several fundamentally different feature spaces:

Trivial Vector Features The simplest was a modified 'trivial' feature space that was generated by scaling each tile image to a fixed 15×15 pixel image using Scikit Image's resize function [16]. A feature vector is then formed using these pixel values. This leads to a $15 * 15 * 3 = 675$ dimension space.

Moment Features The next feature space was a normalized set of moments of the pixel color distribution within the tile. That is, if μ_i^c is the ith moment of the cth of the three color components, then we chose some cutoff n and used a feature vector with $3n$ components

$$\frac{1}{i!}\mu_i^c \quad 1 \leq i \leq n$$

In particular we looked at moments with a cutoffs of n equal to 2 and to 5.

Autocorrelation Features We looked at autocorrelation based feature space, by calculating the autocorrelation function by the formula

$$\Re\left(FFT^{-1}\left\{FFT\left\{I\right\}\overline{FFT\left\{I\right\}}\right\}\right)$$

where I is the image, FFT is the two dimensional discreet Fourier transform and $\Re$ is the real part. We then create a fixed size 675 element feature vector by re-sizing the result in the same manner as the trivial feature space.

Histogram of Gradient (HOG) Features Finally, we used a histogram of gradient (HOG) feature vector as an off-the-shelf test. In particular, we used Scikit Image's HOG function [16] with 4 orientations and 3×3 cells per block. We chose the pixels per cell by dividing the width and height of the image by 4.

Our goal was not to find the "best" feature space, nor was it to do a complete census of all possible feature spaces. Rather, we wanted to examine the effect that choosing a feature space had on the outcome and to show that a given space was adequate for our purpose.

4.2 Feature Vector Evaluation

To develop an unbiased collection of test data for evaluation, we had a naive user of our system select a set of natural images, and specify the size and scale of the desired texture. Then we had a group of 8 graphics students, knowledgeable in the concept of textures and their use in computer graphics, evaluate the quality of texture tiles extracted using different candidate feature vectors.

We gave a naive user minimal training on how to use a simplified version of our system. By "naive", we mean a person who had not been working in our laboratory, was not involved in the development of the software, and was not aware of the algorithms used. The simplified version did not allow for refinement but generated tile sets using all of the above feature spaces. We then instructed the naive user to find images licensed with the Creative Common's attribution license and use the system to extract a single texture from them by specifying a scale and an image location. In total the user selected 19 images and extracted textures from them

For each of those 19 images, we twice extracted 16 tiles for each of the 5 feature spaces. This led to a total of 190 tile sets of 16 tiles. (NOTE: The test images and tile sets are provided in supplemental material [1].) We then used a web interface to display those tile sets to the knowledgeable graphics students, as well as the tile in the location which the naive user originally selected. The graphics students were asked to select tiles in the set that were different from the selected tile. The students were given three options: "Tiles Selected Above.","All Tiles are good","No tile is good". The system would ensure that the user's input was consistent. If the user selected "Tiles Selected Above." the system would insure at least one tile was chosen. If "All Tiles are good" was selected the system would insure that no tiles were selected. Finally, if the user selected "No tile is good" the system would ensure that either all tiles were chosen, or no tiles were chosen.

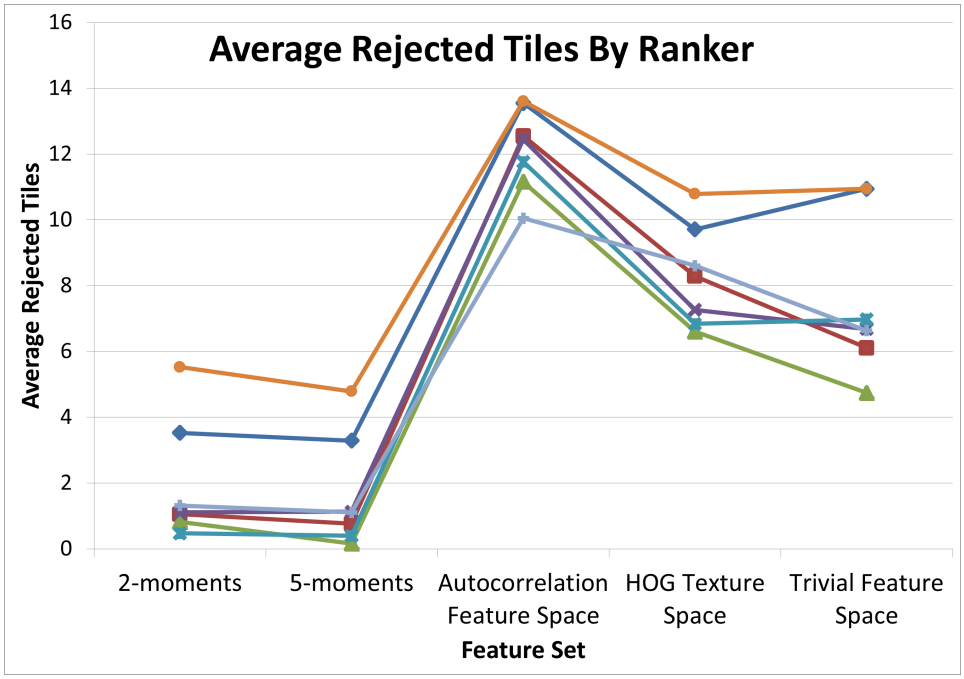


Figure 6: The average number of textures rejected by our graphics student evaluators. Each line represents the responses from a different person. While different tastes varied, the 2-moments and 5-moments feature space were preferred by all users to the other feature spaces. This graph also shows the level of success of our method without any user refinement.

The graphics students who evaluated the tiles were not able to see the images before the evaluation. They were shown the tile sets in a random order, without knowing which feature space generated each tile set.

Figure 6 shows the result of the evaluation process. The moment based feature spaces were clearly superior to the others for this case. The 5-moment space (which requires more computation) was not clearly superior to the 2-moment space. We therefore choose to use the 2-moment feature space and use it in the rest of the paper, unless otherwise noted.

5 Results

Figure 7 shows results of our pipeline produced by the naive user of the system in the experiment described in Section 4. All these results were gen-

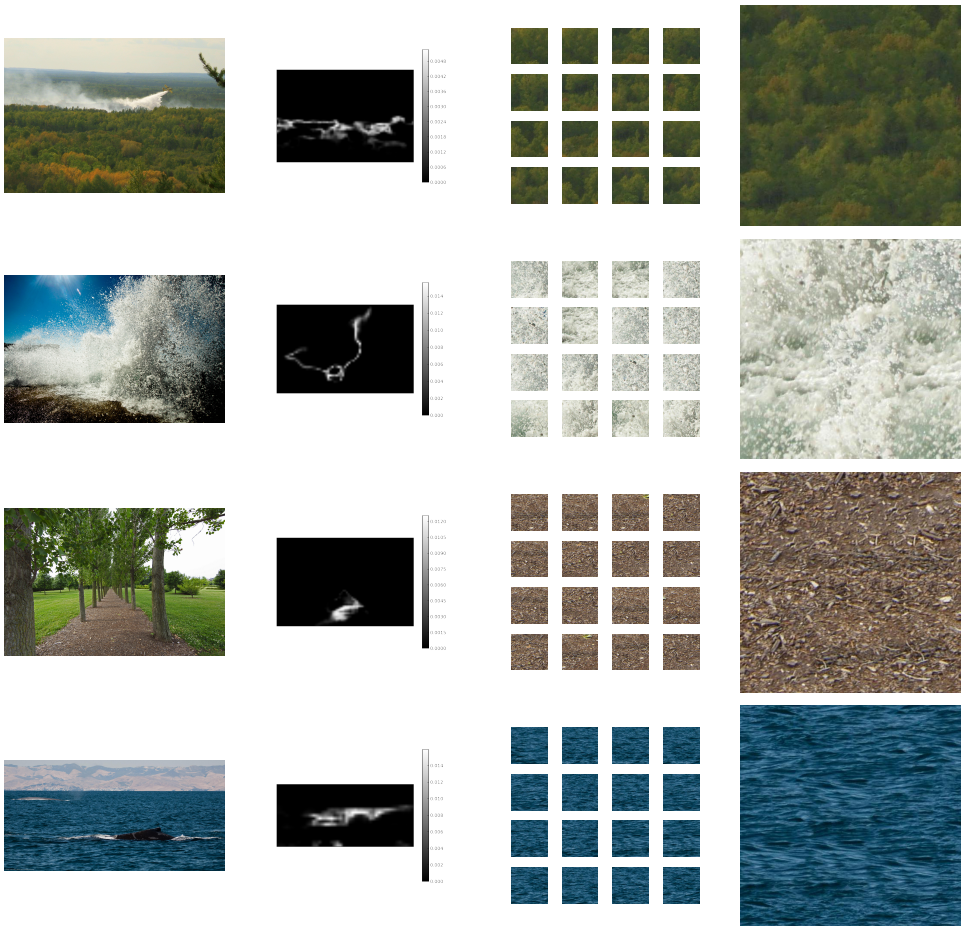


Figure 7: Examples of tiles selected using our method. From left to right: original image, heat map, tiles, generated texture using a basic image quilting technique.

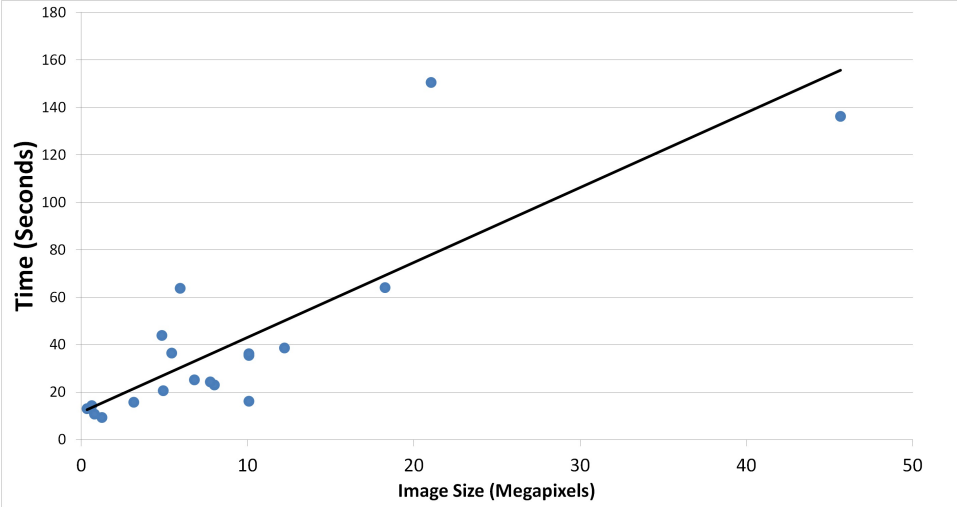


Figure 8: The time complexity of our algorithm. The line shows liner growth with 3.16 seconds per megapixel and a fix cost of 11 seconds. However, this time can be effected by many other parameters.

erated without refinement; only the scale and starting points were selected.

The images selected by the user ranged in size from 0.3 to 45 megapixels. Running on a workstation with a Intel Xeon W3565 CPU and 12 GB of Memory, our algorithm took between 9 and 157 seconds, not including user interactions. Figure 8 shows that the time complexity of the algorithm is approximately linear with the size of the image.

Our method is dramatically faster than that of Lu et al. [5], allowing it to be used interactively. Lu et al. reported that it took 18 minutes to create a texture for a 11,750 pixel image (i.e. 0.011 megapixels). It took our system, running on a laptop, 22 seconds to create a texture from the same image, **including all user interaction**. We show the image results in Figure 9. The previous method produces a single binary mask, while our method produces a multivalued mask (the heat map).

It should be noted that this speedup is still present when using vintage hardware where our algorithm can process images of comparable size in under 10 seconds and megapixel images in 38 seconds.

The performance of our implementation could become unsatisfactory for interactive use if extremely large manifolds were needed. This tends not to be a problem in practice, as such large manifolds would occur for cases where texture tiles each cover a small number of pixels in megapixel images. If the

(a) (b) (c)

Figure 9: (a) Original image (b) mask from Lu et al. [5] (c) Heat map used to identify tiles in our method

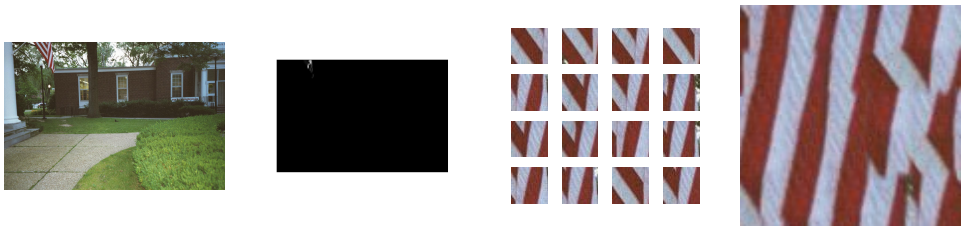


Figure 10: An example of a failure case.

size of the manifold does become an issue we refer the reader to Farbman et al. [11] which discusses sampling when creating a manifold.

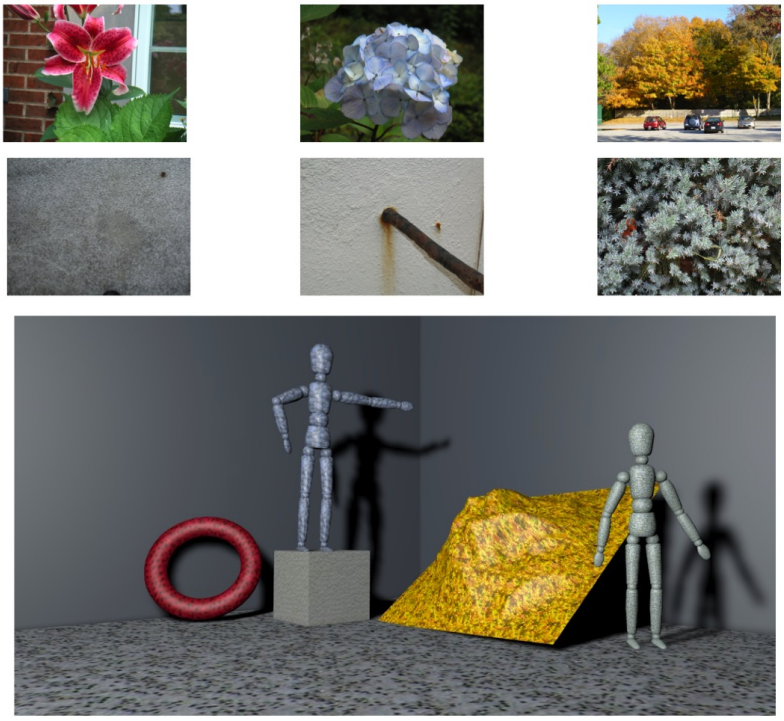
Unfortunately, there are a few specific cases that may cause our algorithm to fail:

1. If a texture is relatively sparse in an image.
2. If two textures within an image are extremely similar.
3. If a texture is directional and the image has multiple rotated representations of that texture.
4. If an image contains multiple scales of a texture.

Often those failures can be easy mitigated by user refinement.

In practice, problems 3 and 4 tend to be the most problematic. In particular they often make extracting structured textures difficult when the texture is on a plane oblique to the plane of the photograph. Eisenacher et al. [10] discuss this problem and a solution.

Figure 10 shows an example of a failure case. Very little of the texture is visible in the image, and different orientations are present. Even such failure cases may produce usable textures.



(a)

Figure 11: Example of application: The first two rows of photographs are used as sources for texture extraction. The extracted textures are mapped on 3D models to render a synthetic scene in the bottom of the figure.

In Figure 11 we model a novel scene using the textures our pipeline extracted from photographs.

6 Future Work and Conclusion

This paper demonstrates that diffusion distance manifolds can be used at interactive rates to distinguish textures in arbitrary images or to generate texture exemplars for texture synthesis algorithms. The method is non-local and able to accumulate information from a large number of image regions. In fact, our implementation can combine patches from multiple images and is only limited by computer memory.

There are many possible extensions to this algorithm. It would be interesting to investigate even more possible feature sets for textures. In partic-

ular, adding directional sensitive features and/or statistical moments other than the image variance may help alleviate problem cases. Furthermore, preliminary results indicate that artificially decreasing the edge weights between a tile and the tiles neighboring it within the image may improve the results. Finally, it would be interesting to combine manifolds from multiple pyramid levels. This could lead to automatic detection of problematic cases and the creation of multiscale textures.

The ability to extract texture exemplars from unconstrained photographs provides an important new capability for computer graphics content creation. Users can create textures from their own personal photograph collections. This allows more creative control in the production of content, and allows the user to have complete control of the IP rights for their content.

Photo Credits

The photos from Figures 3, 4, and 5 are by Elliot Lockerman and used with permission.

The photo credits from Figure 7 are listed below, from top to bottom.

1. Image by Flicker user Micky**, under the Creative Common Attribution 2.0 Generic License
2. Image by Flicker user Nina Matthews, under the Creative Common Attribution 2.0 Generic License
3. Image by Flicker user Todd Ryburn, under the Creative Common Attribution 2.0 Generic License
4. Image by Flicker user mikebaird, under the Creative Common Attribution 2.0 Generic License

The photo in Figure 9 is from Lu et al. [5].

Acknowledgements

We would like to thank the rest of the Yale Graphics Group and Professor Steven Zucker for their help, advice, support, and proofreading for this paper.

We would like to acknowledge Cyril Zhang for his assistance with the prototypes that predated the system described in this work and for the his invaluable suggestions and discussions.

This work was funded by NSF Grant IIS-1064412

References

[1] Technical report companion website.
URL http://graphics.cs.yale.edu/site/tr1483

[2] A. Lasram, S. Lefebvre, C. Damez, Scented Sliders for Procedural Textures, in: EUROGRAPHICS short papers, Cagliari, Italy, 2012.
URL http://hal.inria.fr/hal-00748188

[3] K. Software, Texture Warehouse.
URL http://www.texturewarehouse.com

[4] L. Wei, S. Lefebvre, V. Kwatra, G. Turk, State of the art in example-based texture synthesis, Eurographics 2009, State of the Art Report, EG-STAR.

[5] J. Lu, J. Dorsey, H. Rushmeier, Dominant texture and diffusion distance manifolds, in: Computer Graphics Forum, Vol. 28, Wiley Online Library, 2009, pp. 667–676. doi:10.1111/j.1467-8659.2009.01407.x.
URL http://graphics.cs.yale.edu/site/publications/dominant-texture-and-diffusion-distance-manifolds

[6] B. Nadler, S. Lafon, R. Coifman, I. Kevrekidis, Diffusion maps, spectral clustering and eigenfunctions of Fokker-Planck operators, Arxiv preprint math/0506090.
URL http://arxiv.org/abs/math/0506090

[7] D. J. Heeger, J. R. Bergen, Pyramid-based texture analysis/synthesis, in: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, SIGGRAPH '95, ACM, New York, NY, USA, 1995, pp. 229–238. doi:10.1145/218380.218446.
URL http://doi.acm.org/10.1145/218380.218446

[8] J. S. De Bonet, Multiresolution sampling procedure for analysis and synthesis of texture images, in: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997, pp. 361–368. doi:10.1145/258734.258882.
URL http://dl.acm.org/citation.cfm?doid=258734.258882

[9] L. Wei, J. Han, K. Zhou, H. Bao, B. Guo, H. Shum, Inverse texture synthesis, ACM Transactions on Graphics 27 (3). doi:10.1145/1360612.1360651.

[10] C. Eisenacher, S. Lefebvre, M. Stamminger, Texture synthesis from photographs, in: Computer Graphics Forum, Vol. 27, Wiley Online Library, 2008, pp. 419–428. doi:10.1111/j.1467-8659.2008.01139.x.
URL http://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2008.01139.x/abstract

[11] Z. Farbman, R. Fattal, D. Lischinski, Diffusion maps for edge-aware image editing, ACM Transactions on Graphics (TOG) 29 (6) (2010) 145. doi:10.1145/1866158.1866171.
URL http://doi.acm.org/10.1145/1866158.1866171

[12] J. Lu, A. Garr-Schultz, J. Dorsey, H. Rushmeier, A psychophysical study of dominant texture detection, in: Proceedings of the 6th Symposium on Applied Perception in Graphics and Visualization, ACM, 2009, p. 133. doi:10.1145/1620993.1621027.
URL http://doi.acm.org/10.1145/1620993.1621027

[13] T. Cour, F. Benezit, J. Shi, Spectral Segmentation with Multiscale Graph Decomposition, in: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Volume 2-Volume 02, IEEE Computer Society, 2005, pp. 1124–1131. doi:10.1109/CVPR.2005.332.

[14] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, A. Y. Wu, An optimal algorithm for approximate nearest neighbor searching fixed dimensions, Journal of the ACM (JACM) 45 (6) (1998) 891–923. doi:10.1145/293347.293348.
URL http://doi.acm.org/10.1145/293347.293348

[15] A. A. Efros, W. T. Freeman, Image quilting for texture synthesis and transfer, in: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, SIGGRAPH '01, ACM, New York, NY, USA, 2001, pp. 341–346. doi:10.1145/383259.383296.
URL http://doi.acm.org/10.1145/383259.383296

[16] S. van der Walt, Skimage Documentation.
URL http://scikit-image.org/docs/0.7.0/