Texture Synthesis using Image Quilting

TSBK03 - Teknik för Avancerade Datorspel

Torsten Gustafsson [1]

December 28, 2016

[1]Torsten Gustafsson, Media Technology student at Linköping University, torgu529@student.liu.se

Chapter 1

Summary

This report aims to present a texture synthesis application made in C++, using the graphics library OpenGL. Texture synthesis is the method of generating large textures based on smaller, patterned textures. The texture synthesis method used here is a method called 'Image Quilting'.

Chapter 2

Introduction

2.1 Theory

Texture synthesis have been an area or research in computer graphics for many years [1]. According to [3], texture synthesis methods can be roughly separated into three distinct classes. The first one is using a set of parameters to describe a variety of textures to be generated. The downside of such a parametric representation is that it requires very good input textures (clearly repeating patterns) to match its parameters with, and is therefore not able to synthesize all patterns in a good way. The ones it will work for though, will generally look very good since those methods generalizes the pattern to such an extent.

The second class is to generate a texture without predefined parameters. Instead it bases the synthesis on pieces of the input texture (e.g. pixel neighborhoods). This differs from a parametrized method in that instead of trying to find a general pattern that can be described for the whole image, it focuses on properties of the actual image, and generates a texture based on those. The common factor of these techniques is that they generate the texture one pixel at a time.

The Third class of techniques described [3] is based on *patches* to generate textures. An area of the input texture is copied an placed on the output image, based on a set of matching criteria. The method used in the application described in this report is based on this class of methods.

2.2 Previous Work

There have been numerous methods presented in the field of texture synthesis. Determining if a method yields 'good' results is often a difficult thing, since the

definition of a 'good' texture is difficult to formalize. Li-Yi Wei and Marc Levoy [2] presented a method using tree-structured vector quantization. Based on the pixel neighborhoods of each pixel to be generated they managed to generate textures of arbitrary sizes, which often matched the input pattern. The trick here was to match the size of the matching neighborhood to the size of the actual pattern in the texture. This required some expert knowledge in both the algorithm and the texture to be synthesized for each use, but the results were often good.

Alexeis A. Efros, William T. Freeman [1] presented another method, based on image patches. By taking square patches from the input texture and placing them on the output texture, after matching each patch with its neighboring patches they managed to get relatively good results with good performance. This method also required the knowledge of the pattern size beforehand to match the patch size with the size of the pattern in order to get good textures. This method is called "Image Quilting" and is also the method used by the application described here.

A third interesting method is based on *graph cuts* and was presented by [3]. This method uses the separating lines of distinct objects in the input texture to cut out its patches, which are then used to synthesize new textures. this method succeeds very well in generating new textures without visible seams. This method is not directly applicable in the context of this report, but shows some of the possibilities of texture synthesis.

Chapter 3

Method

The application was developed in C++, using Ingemar Ragnemalms common library for OpenGL. The library allowed for easy use of Frame Buffer Objects (hereafter referred to as 'FBO') which was used extensively for generating the textures.

The method used is called Image Quilting, which is based on taking small "patches" from the input texture and aligning them on the output texture, based on matching of each patch's neighbors. Figure 3.1 shows how the method fills an output texture using patches.

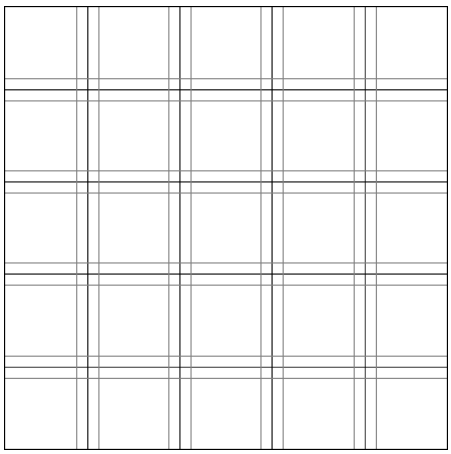


Figure 3.1: The image quilting method generates an output image by placing patches in a chessboard-like pattern. Patches overlap to hide any visible seams between them.

The patches must be several times smaller than the input texture for the method to generate a varying result pattern. Each patch is matched with its neighbors to determine if it is a good fit. The way the matching was implemented in this application was by adding a completely random first patch in the lower left corner, which was then used to test against for the next patch, and so on. Figure

3.2 shows the order in which the patches was placed. This way of taking purely random patches may give rise to some problems, which will be discussed later.

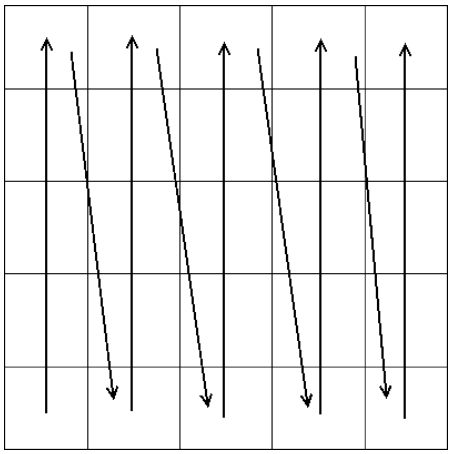


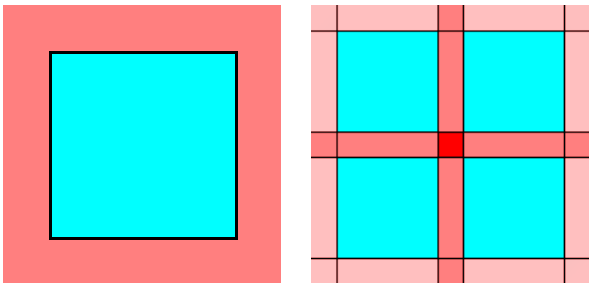
Figure 3.2: The order patches was added in this specific application.

By placing patches in this order, each new patch had to be tested against its lower neighbor, as well as its left neighbor (patches at the far left or at the bottom only had one of those neighbors, and was thus only tested against that one).

For every new patch to place, several temporary patches was generated randomly. Each of those temporary patches was matched with its neighbors, one at a time, using a matching function called "Minimum Error Boundary Cut". This matching function is based on checking the distance, for each pixel, in color value, in the overlapping area of the two patches. This matching function was implemented in a shader that took the two textures (patches) as input, and extracted the pixel values of the two areas that was overlapping.

Patches was placed using several shaders. Since patches needed to be matched against each other, the texture coordinates for the relevant matching areas had to be known. To solve that problem the patches was placed in an FBO with known width and height, and then sent in, two at a time to the matching shader. When a new patch was considered a better match than the previous best result, it was placed in an FBO with the correct output texture coordinates, and then added to the result image. The adding of textures was achieved using a simple combine shader. The result texture was worked through in that way, one patch at a time, until every patch position had been filled.

To make a smooth overlap between the patches, some overlap regions needed to be defined, as shown in figure 3.4a. Figure 3.4b shows these regions when they overlap with other patches. Here it can be seen that the edge cases need to be handled separately, since that area will otherwise be considered in two of the side cases. A linear interpolation from the start to the end of the overlap region was performed on each side of the patch, giving the appearance of figure 3.4a. Figure 3.4b shows the result of the overlapping of multiple patches, using

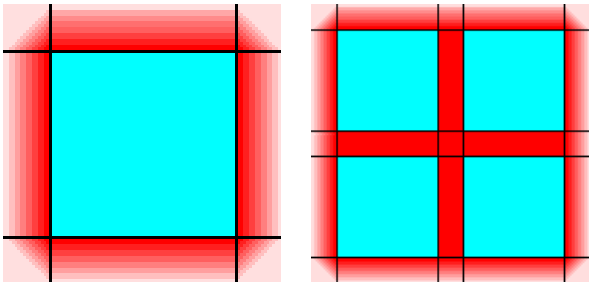


(a) Overlapping region is shown in red.

(b) Overlapping regions are additive.

Figure 3.3: Here the patches overlapping regions are shown in red. When multiple patches overlap, the overlapping areas must be blended in a satisfying way.

this interpolation.



(a) A linear interpolation is added to the overlapping regions.

(b) Color value adds up evenly on overlapping areas.

Figure 3.4: A linear interpolation of the color value is added in the overlapping regions for each patch, as shown in (a).

Chapter 4

Results

This method have several important parameters to tweak in order to generate good textures based on an input texture. The input texture itself must also fill certain criteria for the method to be able to generate good textures from it. The first and foremost of them is that it must be some kind of repeating pattern. If it does not contain a clear pattern, the generation will not be able to find good matches for neighboring patches, and thus making visible seams in the output texture, which is obviously not what we want.

The method yields best results when the size of each patch is similar in size to the actual pattern in the texture. This is difficult for the program to recognize on its own, so the user will have to set a fitting value beforehand.

Since the implementation generated a fixed number of patches to choose from, for each new patch to place, the number of these generated patches played a big part in how good the output texture looked. With more patches, the final result was more likely to look good, but it also increased computation time. Figure 4.1 shows the computation times for different values of this number.

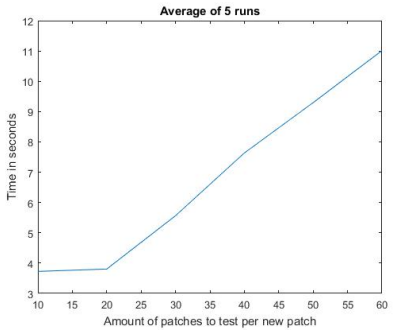


Figure 4.1: The performance difference when increasing the number of patches to test.

Since the RGB color space is not intended for calculating distances in color val-

ues, an experiment using CIELAB was performed instead, which yielded marginally better results, as shown in figure 4.2.

(a) RGB color space matching.

(b) CIELAB color space matching

Figure 4.2: Difference of RGB- and CIELAB color space matching. Both images was generated using the same input image. CIELAB is better at hiding the seams between patches, which are clearly visible in the RGB image.

Chapter 5

Evaluation and Conclusion

The application works quite well in accordance to the method used. It has some problems removing the visible seams, especially when the pattern contains many distinct colors. This can in some cases be fixed by increasing the number of test patches, but not always.

Since patches are taken purely randomly, there might be problems, say for example if the first patch (which is taken completely random) is taken from the edge of the input texture, and the pattern is not clearly repeating anywhere, based from that area, the method will not be able to find a good matching new patch, no matter how many areas are tested. This could maybe be fixed by having the first patch taken from somewhere in the middle of the texture, so new patches can be extended in all directions, but the problem would reappear whenever a patch is taken in the edge, which is most likely inevitable.

The method can probably be refined somewhat, but to generate really good textures, another method will most likely have to be used, as most of the problems comes from the method itself. A similar method, that uses individual pixels instead of patches, with a bigger neighbor area is described in [2]. In this method, the extracted pixel is compared to a number of its neighbors from the input texture, to match with its placement in the output texture. This method will probably be able to generate better results than the one used here, as it is more restrictive on individual pixels, and will thus probably not have the clear seams that may appear in the method used here, but it will most likely also be considerably slower, as each output pixel have to be tested against every input pixel.

The Graph Cut method described in the "Previous Work" section is using the concept of patches, like the method used here. What makes that method better is that it does not always take square patches of the same size, but instead finds the relevant edges in the input texture to use as boundaries for its patches. This will most likely generate very good looking textures, as the seams will be found explicitly by the algorithm, instead of picking a bunch of random patches to test

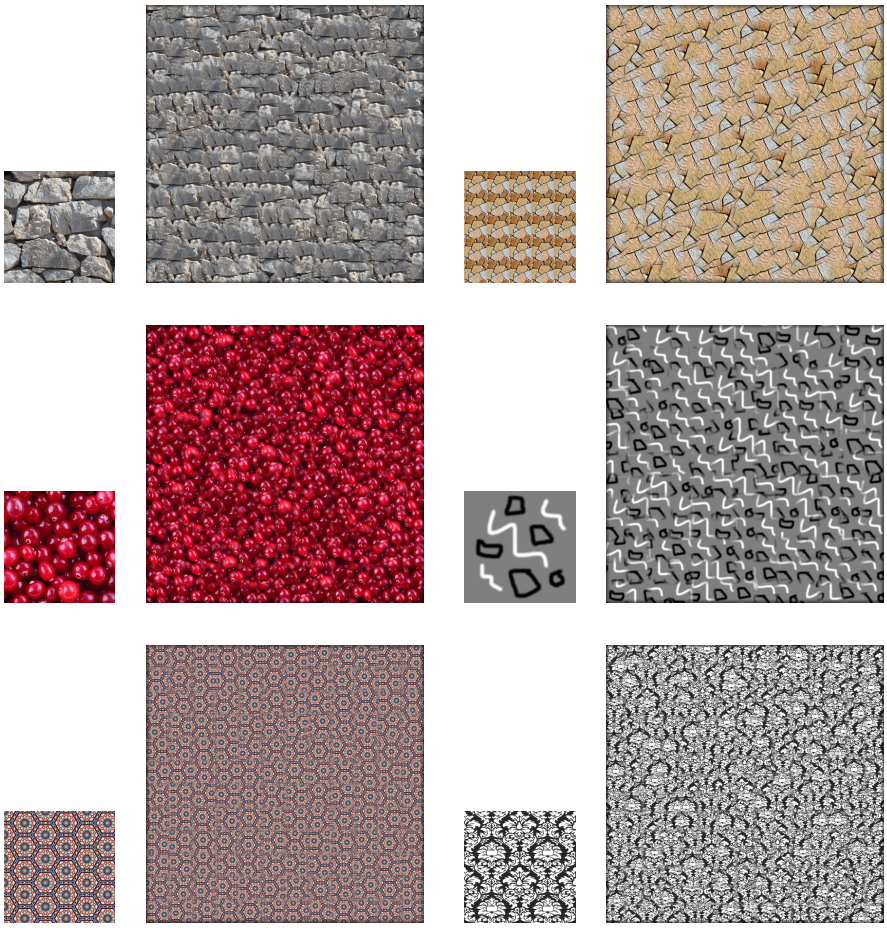
against, hoping to find a good match. Because the actual pattern may not always be strictly horizontal or vertical, this approach is more robust.

So in conclusion, the method used works somewhat good, but to solve the problems it inherently has, another, better method should be considered instead.

Chapter 6

Appendix

Some result images a presented here. Left is input texture and right is the resulting output texture.



Bibliography

[1] Alexeis A. Efros, William T. Freeman (2001) *Image Quilting for Texture Synthesis and Transfer*, SIGGRAPH '01 Proceedings of the 28th annual conference on Computer graphics and interactive techniques

[2] Li-Yi Wei, Marc Levoy (2000) *Fast texture synthesis using tree-structured vector quantization*, SIGGRAPH '00 Proceedings of the 27th annual conference on Computer graphics and interactive techniques

[3] Vivek Kwatra, Arno Schödl, Ifran Essa, Greg Turk, Aaron Bobick (2003) *Graphcut Textures: Image and Video Synthesis Using Graph Cuts*, SIGGRAPH '03 ACM SIGGRAPH 2003 Papers