Image Analogies with Patch Based Texture Synthesis

Patrick Gillespie

Abstract

In this paper we introduce a simple new approach to image analogies using patch based texture synthesis. The patch based method takes the place of the pixel based method that formed the foundation of the original image analogy algorithm. The new method is able to generate reasonable output for certain classes of images and is able do it in a speedy manner.

1 Introduction

An analogy helps us figure out the relationship between a set of objects, given that we understand the relationship between a different set of objects that have a similar type of relationship. If we know B relates B' the same way A relates A', and we understand the relationship between A and A', we are able to deduce the relationship between B and B'. This simple reasoning process is extremely powerful and is handy tool in understanding. For notation proposes, analogies can be expressed as follows:

$$A : A' :: B : B'$$

An image analogy is an analogy between two pairs of images. Image analogies can be advantageous in that they allow us to easily learn and apply image transformations. They have the potential of saving one the time of manually programming many different image filters or programming new image filters once a different image transformation is developed.

Tools and methods for creating image analogies have been created, studied, and have produced quite amazing work [Hertzmann et al. 2001]. However, the existing method for creating image analogies can take minutes to hours to compute in certain circumstances. Thus we explore an alternative way of creating them, using some of the latest work in texture synthesis.

Texture Synthesis is the process of taking a small sample of a texture and generating more of it. For example, in Figure 1, given the input image A, a good texture synthesis algorithm should be able to generate more of the texture to create an image like B.

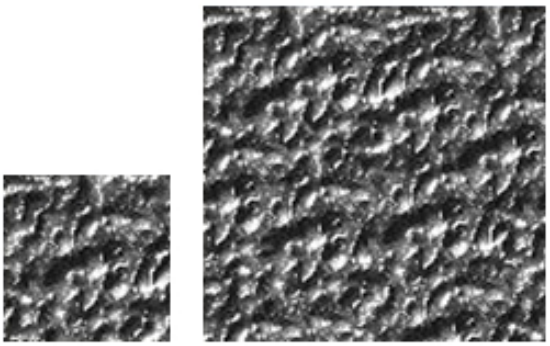

Figure 1: Texture Synthesis Example

The above example was created through Graph Cut texture synthesis [2001] which uses patches of texture to create its output image. In this paper we attempt to integrate this work along with other patch based texture synthesis techniques to help accelerate the creation of image analogies. We explain a simple algorithm that one can use to create them.

2 Previous Work

Image analogies were originally developed by Hertzmann et al. [2001]. The algorithm proposed produces extremely good results in a number of different circumstances and has a number of different applications including, but not limited to, traditional image filters, artistic filters, super-resolution, texture by numbers, texture transfer, and improved texture synthesis. It has also been shown to work well for image colorization [Welsh et al. 2002].

The algorithm takes in 3 input images: An unfiltered source image A, and filtered target image A', an unfiltered source image B, and produces a filtered target image B'. Building upon the texture synthesis work of Wei and Levoy [2000] and Ashikhmin [2001], it uses a multiscale representation of all the images, refining the output image B' in each successive level. As in the Wei and Levoy [2000] and Ashikhmin [2001] texture synthesis algorithms, the

algorithm works by constructing the image pixel by pixel. This attention to detail leads to the algorithms only main drawback, which is that it takes a reasonable amount of time to complete. The time of execution varies depending on the input and the desired goal. They note that on a 1GHz PC that their algorithm's execution time can take anywhere from a few seconds for texture synthesis to a few hours for artistic renderings.

Since the publication of the Image Analogies paper, faster methods of texture synthesis have been developed. These methods usually build their output images with patches of texture as opposed to building them pixel by pixel at many different resolutions.

Image Quilting [Efros and Freeman 2001] is one such method. In Image Quilting, the output image is generated block by block in raster scan order. New blocks are placed next to old blocks with an over lap of 1/6 of the block size. A dynamic programming algorithm is then used to determine which pixels from the new block will show up in the overlap region in the output image. This region of pixels forms a seam between the new block and the rest of the picture. If the block is a good match, the seam will barely, if at all, be noticeable.

Blocks being placed into the texture are all of the same size. A group of possible blocks from the input texture are selected based on how well their overlap regions match the region on the output image that they would overlap. One of these blocks is randomly chosen to be the next block in the texture.

Graph Cut Texture Synthesis [Kwatra et al. 2003] is another method of texture synthesis that works with patches of texture. Instead of using a dynamic programming algorithm to determine the best cut between two images that are overlapping, it uses a min cut [Ford and Fulkerson 1962] algorithm to determine the optimal cut between two images. The overlapping region between two patches, lets say A and B, is set up as a graph of nodes where each pixel in the overlap is represented by a node. Nodes along the border next to a patch link back to a single node that represents that patch. This node is either the source or the sink in the min cut algorithm. Nodes that are adjacent to each other in the graph have arcs between them that are weighted based on the following equation:

$$M(s, t, A, B) = ||A(s) - B(s)|| + ||A(t) - B(t)||$$

Here s and t are adjacent pixel positions and A(s) represents the color of pixel s in patch A, and B(t) represents the color of pixel t in patch B. After the graph is set up, running the min cut algorithm will yield the optimum cut between the two patches. Extremely fast min cut algorithms and implementations have been developed [Boykov et al. 1999].

Figure 3 gives an example of the set up for finding the optimal cut. Nodes bordering the A and B patches link back to it and all adjacent nodes have arcs between them. Here the pixel overlap is only 9 pixels. A red line indicates a possible min cut. The pixels on the left of this cut would end up coming from patch A, while the pixels on the right of the cut would end up coming from patch B.

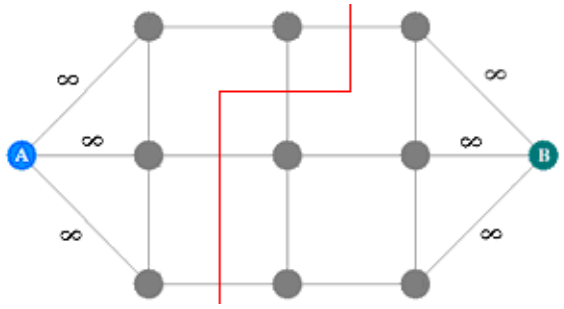

Figure 3: An example of a minimum cost cut between two texture patches.

3 The Algorithm

The algorithm presented in this section uses ideas from Image Quilting [2001], Graph Cut Texture Synthesis [2003], and the original Image Analogies paper [2001]. For input, we take in three images: An unfiltered source image A, a filtered source image A', and an unfiltered source image B. We output a filtered target image B'. The algorithm is as follows:

- Select a block size and block overlap size to be used. Tests seem to show that a block overlap of ½ works well. Small overlaps seem to produce blocky outputs while large overlaps seem to produce noisy outputs.
- The image B' is generated in a way similar to Image Quilting. We move through image B, examining it block by block with respect to the block overlap we have chosen. We analyze each block and find a block of pixels in image A that matches well with it. This can be done by randomly selecting a number of blocks from A,

or by selectively searching through blocks in A. To keep the output image from being overly repetitive, we find a group of blocks that matches well with the block in B, and then randomly select one of these blocks.

- The block in A' that corresponds to the block we selected in A is then chosen to be copied over to the output.
- The region the A' block is placed at overlaps existing pixels in B', the graph cut method discussed in Section 2 to select which pixels in the overlap are replaced with the new block's pixels.
- This process is repeated until the image B' is fully synthesized.

Figure 4 gives a visual example of this process. The red block in A is selected as matching well with the blue block in B. The corresponding red block in A' is then selected to be placed in the blue block area of B'. A cut is made to determine which pixels are copied to the overlap region.

As was done in the original image analogy algorithm, one can chose to only copy over certain pixel features from the pixels in A', such as luminescence. This is essential when the input images A and A' don't have all the colors needed to create an output image B'. Tests seem to indicate that output images whose results were generated by strictly by copying over the raw pixel data tend to look inferior to images generated in the same fashion which only had luminescence copied over.

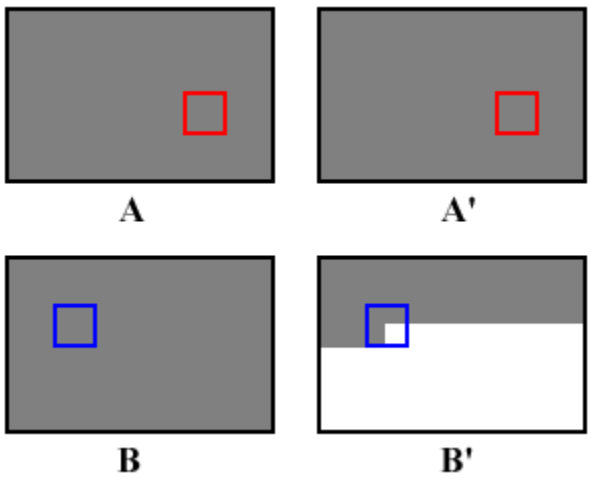

Figure 4: Analogy construction diagram. The dark gray areas represent filled in pixels. The white areas represent areas not yet filled in

To obtain even better results, we can use the pixels in the overlap region of B' to help find a better set of block candidates for the next block that will be written in B'. This can happen as follows: The pixels in blocks from B and A are compared, and weighted in with this comparison are how well the corresponding blocks from A' and B' match in the regions that have already been filled in. This will allow us to find the blocks that will produce the least noticeable seams in the output image.

3.1 Block Size

A hidden variable in this algorithm is the block size used in comparing and copying over pixels. The block sizes can vary depending on the size of the inputs and the features you want to capture. Too small a size and your output will look very pixilated. Too large a size and your output will simply look like a scrambled version of your input image A'. There is no set way of determining the block size, though from experimentation, block size between 30x30 and 40x40 pixels seem to work well.

3.2 Comparing blocks of pixels

In order to find a reasonable block of pixels to use from our input images, we use a cost function that tells us how well a particular image block matches with another image block. A sum of squared differences between blocks of pixels works fine for such cases [Kwatra et al. 2003]. However, the sum of squared differences function is computationally heavy, and to keep time down it is wise to only check a random number of offsets for possible match ups. Checking only around 2000 offsets can usually lead to reasonable results, as seen in Figure 5. If one wishes to do a full patch search of every possible block, in order to find the best possible candidates, one can use block matching techniques that are accelerated by the Fast Fourier Transform. Such techniques [Kwatra et al. 2003; Kilthau et al. 2002; Soler et al. 2002] can lead to speeds 13%-29% of an exhaustive search [Soler et al. 2002]. Full patch searches do have their draw back though, and they will be discussed in the results section.

3.3 Sets of Analogies

Often times, an example analogy will not contain all of the features one needs in order to accurately capture the results in the B' image. A simple enhancement is to have multiple example analogies as an input in creating the output. One would then

simply consider the best block offsets from the input sets in determining the patch to copy over. Figure 7 gives an example of an analogy that was created with an analogy set.

4 Results

Following the Reference section of this paper, you will find a number of result images. Figure 5 gives four image analogies that were created using random patch offsets to find the best possible candidate patch. As you can see, the resulting B' images come out fairly well for being made completely out of pixel chunks from their input A' image. Since the possible pallet of pixel chunks for B' comes entirely from input A', we chose to use input B images that were very similar to input A images.

Each of these results took between 45 seconds and 3 minutes to generate on a 2.01GZ PC. The first analogy shown, of the tree on the hill, took 45 seconds to generate while output for the same inputs took over 3 minutes to generate using the software provided by Hertzmann et al. [2001] for the original algorithm. It should be noted that the original size of these images was 200x313 pixels.

Figure 6 shows a comparison between images generated using the original algorithm and our algorithm on an example set of size 512x400 and an input B image of size 512x383. In this case, a FFT block matching speed up was used to find optimal patches. Using a full patch search without the FFT speed up it took slightly over 10 hours to generate an output image for this set. Using a full patch search with the FFT speed up it took only 1,017 seconds to generate an output image. However, using the original algorithm, it took only 863 seconds, and produced a better output. Thus the FFT speed up, though an incredible speed up over full patch searching, it is not fast enough to beat the original algorithm's speed.

It should be noted that both algorithms used luminance copying in this case to produce the colors in their output images. A black and white image is shown in Figure 6 as an intermediate step that was taken by the original algorithm in creating its output. Our algorithm, though it also used luminance copying, was not able to achieve the same level of detail.

Here the features of the input image cannot be expressed well by the input analogy of a forest. This shows the importance of the need for an input analogies pair to be similar to the input image B. The greater the difference between the input images A and B, the harder it will be for image B to be captured in the output.

This brings us to one of the algorithms limitations: the image analogies that can be emulated are somewhat limited. The input image B must be similar to the input image A in order for an accurate B' image to be generated. Also, since pixels are transferred in blocks from image A' to image B', some detail is lost in the output image. Traditional filters such as the blur filter are not emulated well at all. In fact, any kind of image transformation that requires precise details will not fair well.

However, there are plenty of cases where the output images seem to come out nicely. Through testing it seems to be shown that this process works well for artistic filters that do not require precise details and have a certain amount of noise in their filtered output. Texture transfer also comes out nicely. This can be done by setting images A and A' equal to the texture one wishes to apply to image B.

Finally, Figure 7 gives an example of an output that was generated using an analogy set. Blocks from both example sets end up being used in the creation of the output image. Two outputs images are shown, one where pixels were copied over directly, and one where only luminance was copied over. These two images were generated on the same run of the program, so they are made up of the same patches. The only difference between them is in what was copied over.

5 Conclusion

In this paper we have described a possible alternative to creating image analogies. Test results indicate that the algorithm works well for inputs that are similar and that resultant images can be generated in a fraction of the time of the original algorithm if only a subset of patch offsets are considered. Full patch matching using the FFT can produce images that have better candidate patches, but resultant images will take longer to produce than the original algorithm.

5.1 Acknowledgements

We would like to thank Hertzmann et al. [2001] and Boykov et al. [1999] for making their software available for use. We would also like to thank John

Shaw for letting us use his “Darkclouds” and “Swawn” images (used in the first two analogies in Figures 5, as the example analogy set in Figure 6 and as second analogy set in Figure 7), Rachel Dodge for letting us use her flower image in Figure 6, and thank Hertzmann et al. [2001] again for letting us use some of their images.

References

Michael Ashikhmin. Synthesizing Natural Textures. *2001 ACM Symposium on Interactive 3D Graphics*, pages 217–226, March 2001.

Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *International Conference on Computer Vision*, pages 377–384, 1999

Alexei A. Efros and William T. Freeman. Quilting for Texture Synthesis and Transfer. *Proceedings of SIGGRAPH 2001*, August 2001.

Lester R. Ford, Jr. and D. R. Fulkerson. Flows in Networks. *Princeton University Press*, 1962.

Aaron Hertzmann , Charles E. Jacobs , Nuria Oliver , Brian Curless , David H. Salesin, Image analogies, Proceedings of the 28th annual conference on Computer graphics and interactive techniques, p.327-340, August 2001

Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts, *ACM Transactions on Graphics (TOG)*, v.22 n.3, July 2003

Kilthau, S.L., Drew, M., and Moller, T. 2002. Full search content independent block matching based on the Fast Fourier Transform. In *ICIP02, I*: 669–672.

Cyril Soler, Marie-Paule Cani, Alexis Angelidis, Hierarchical pattern mapping, *ACM Transactions on Graphics (TOG)*, v.21 n.3, July 2002

Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller, Transferring color to greyscale images, *ACM Transactions on Graphics (TOG)*, v.21 n.3, July 2002

Li-YiWei and Marc Levoy. Fast Texture Synthesis Using Tree-Structured Vector Quantization. *Proceedings of SIGGRAPH 2000*, pages 479–488, July 2000.

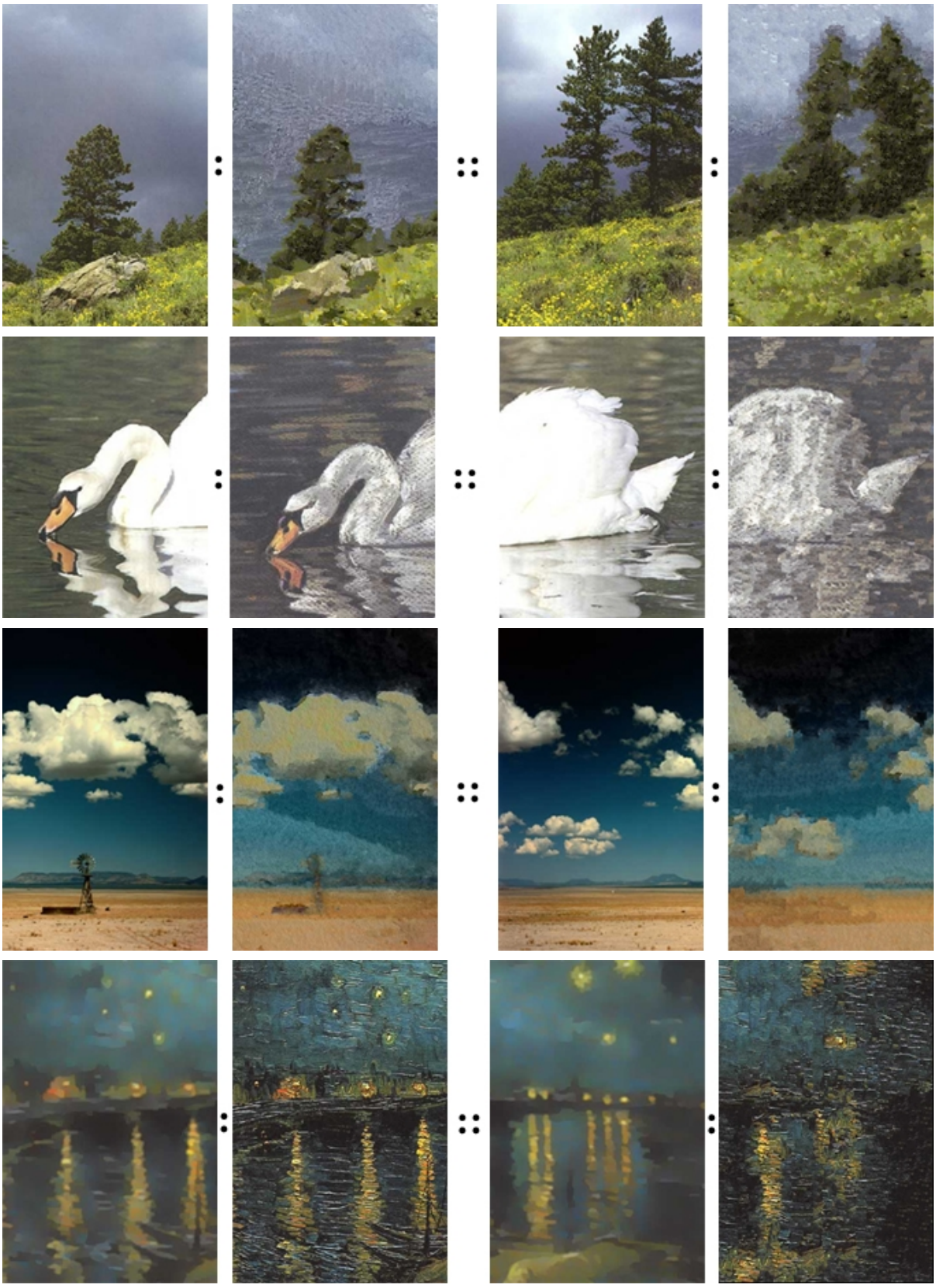

Figure 5: Images analogies created using a subset of patch offsets

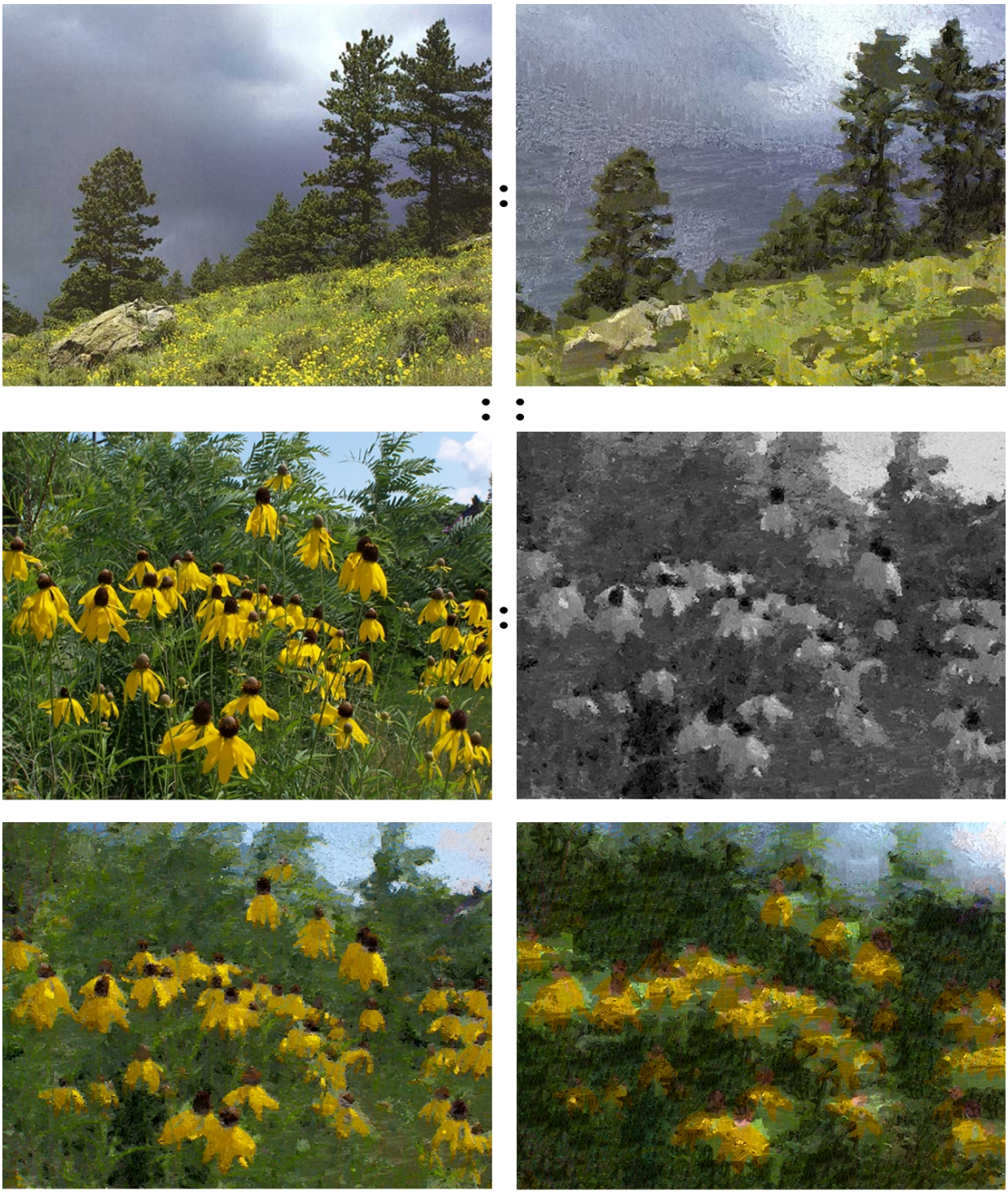

Figure 6: A comparison between the original and the patch based method

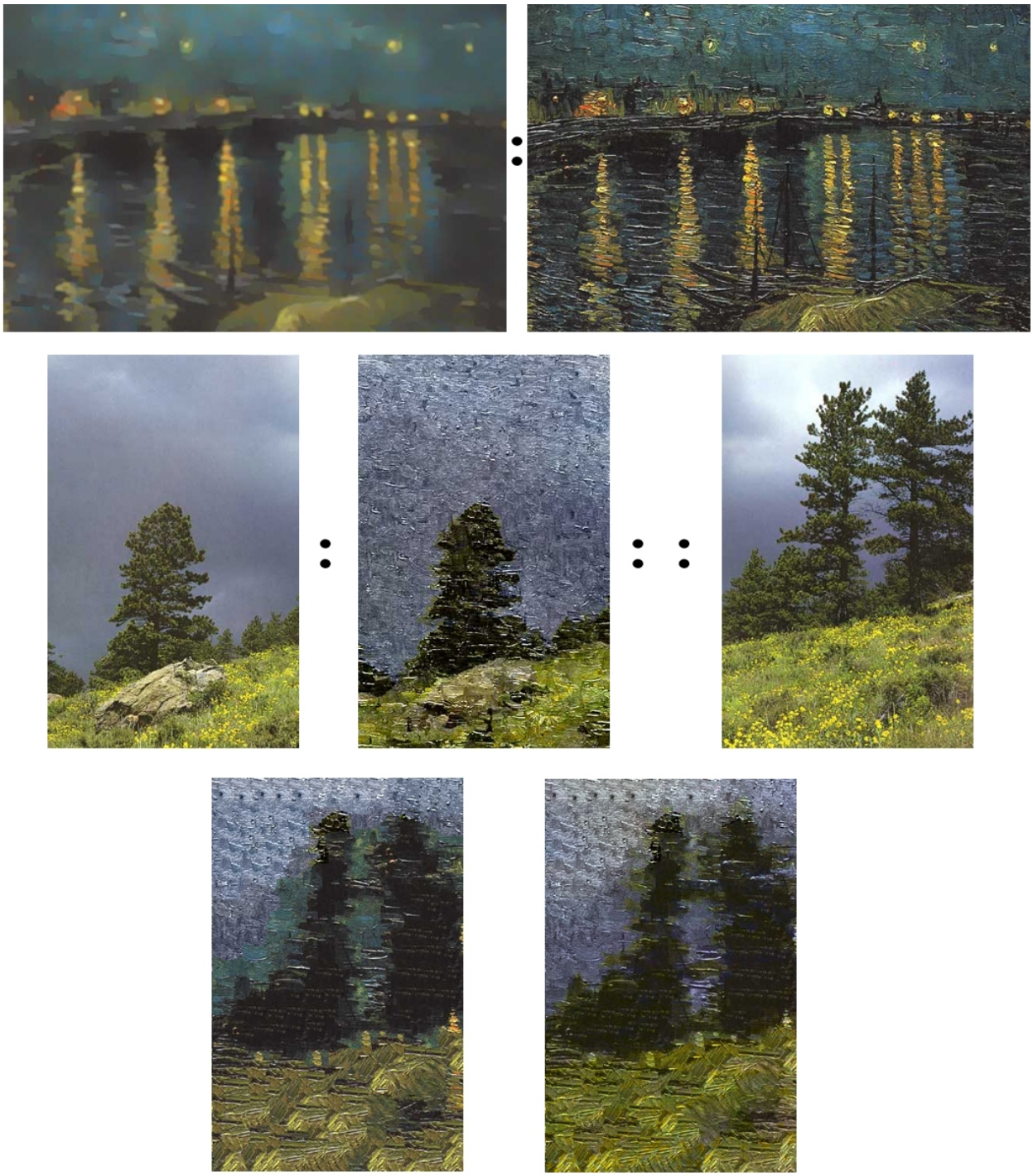

Figure 7: An image analogy example that uses sets of analogies. Two outputs are shown, one where raw pixel values were copied over, and one where only luminescence was copied over.