Pyramid-based Image Synthesis - Theory

Shuguang Mao and Morgan Brown[1]

INTRODUCTION

Motivation

In the context of this paper *image synthesis* is the process of transforming an uncorrelated image to one with the same textural qualities as a known "traning image" (**TI** for short). The traditional applications of image synthesis have been in computer graphics, but same ideas will prove useful for earth scientists, due to the nature of their experimental measurements.

To create the synthesized image (**SI** for short), we first compute some of the TI's key one-point and two-point statistics and then impose these statistics on the SI. To overcome the difficulties in estimation induced by scale variance, our method utilizes the "Laplacian Pyramid" decomposition, a simple and well known tool for multiscale image analysis.

The Laplacian Pyramid

The general class of linear transform decomposes an image into various components by multiplication with a set of transform functions. Some examples are the Discrete Fourier and Discrete Cosine Transforms, the Singular Value Decomposition, and finally, the Wavelet Transform, of which the Laplacian Pyramid and other subband transforms are simple ancestors.

Real-world digital images are in general both scale-variant and highly nonstationary in space. They contain a variety of objects and features (lines, shapes, patterns, edges) at different scales, orientations, and spatial locations; features which the ideal image transformation should independently extract into easily manipulable components (**?**).

The Laplacian Pyramid decomposition, originally developed by Burt and Adelson (**?**), is illustrated in Figure 1 for a two-level pyramid. The following pseudocode describes the simple process for a pyramid with an arbitrary number of levels.

[1]**email:** mao@geo.stanford.edu, morgan@sep.stanford.edu

g = `makeGaussianFilt`()

do(i = 0 : nScales-1) {
  $l_i = g * f_i$
  $h_i = f_i - l_i$
  f_{i+1} = `subSamp2`(l_i)
}

output: $f_{\text{nScales}}, h_1 \cdots h_{\text{nScales}}$

Notice from the pseudocode that the pyramid consists of `nScales` "highpass" bands h_i and a "lowpass" band, f_{nScales}. The h_i contain most of the image's important textural features, at different scales. The Laplacian Pyramid is named as such because the process of computing h_i by subracting a blurred copy f_i from f_i is equivalent to convolving f_i with (approximately) the Laplacian of the Gaussian blurring filter.

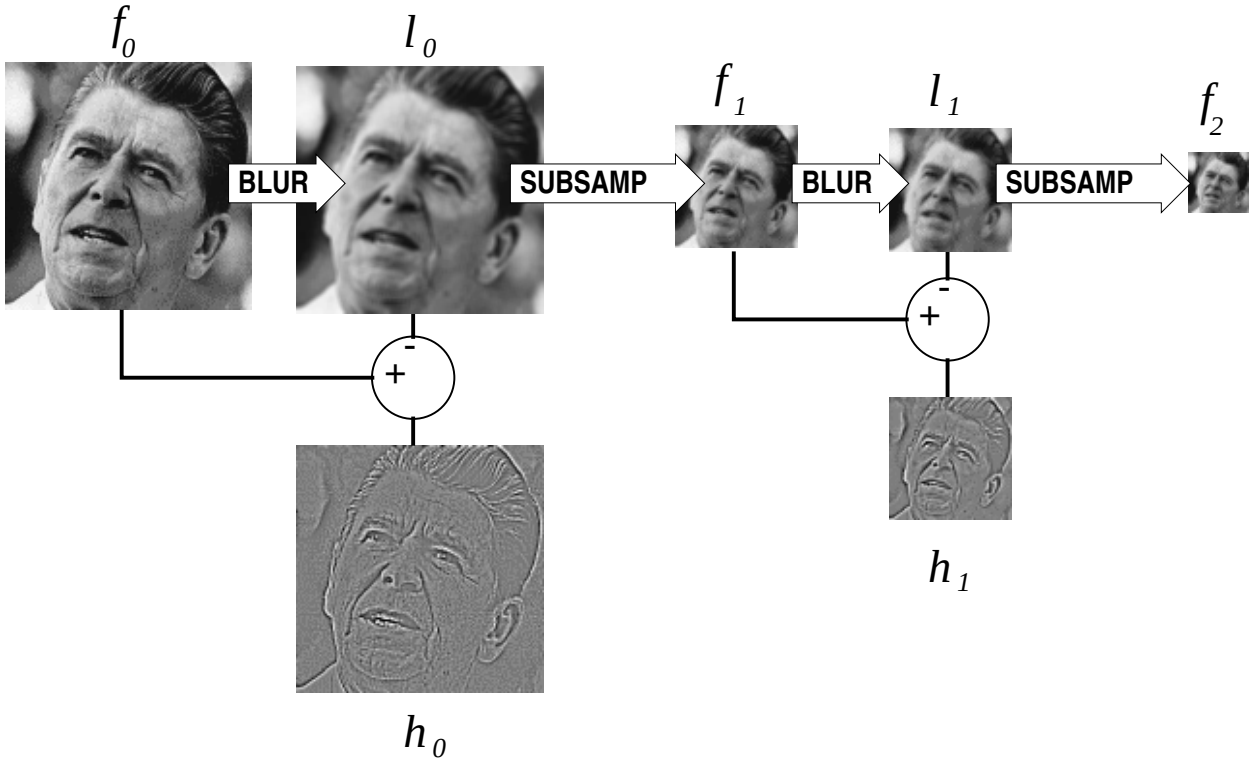


Figure 1: Decomposition step for two-level Laplacian Pyramid. The finished pyramid consists of the two "highpass" bands, h_0 and h_1, and the "lowpass" band, f_2. lapl-pyr-decomp [NR]

The reconstruction step for a two-level Laplacian Pyramid proceeds in preditictable faction, and is illustrated in figure 2. The following pseudocode describes the process for a pyramid with an arbitrary number of scales.

g = `makeGaussianFilt`()

do(i = nScales:-1:2) {
  $l_{i-1} = g * \texttt{upSamp2}(\ f_i)$
  $f_{i-1} = h_i + l_{i-1}$
}

output: f_0

The function `upSamp2()` simply inserts zeros between the samples of f_i, raising its size by a factor of two.

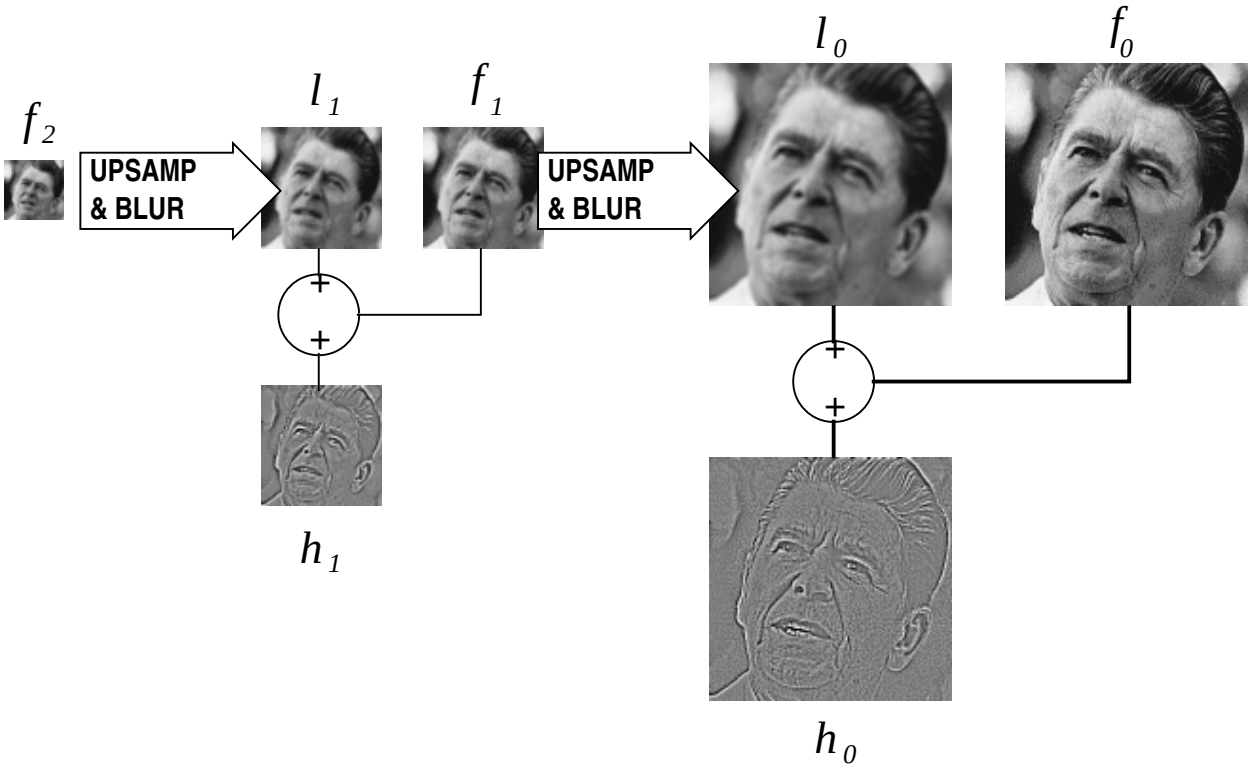


Figure 2: Reconstruction step for two-level Laplacian Pyramid. The process begins with the two "highpass" bands, h_0 and h_1, and the "lowpass" band, f_2, and then perfectly reconstructs the starting image, f_0. lapl-pyr-recon [NR]

Statistics Matching

Image synthesis is the process of first measuring the key statistics which characterize the texture of the TI, and then imposing these statistics on the SI. Below we describe the procedure used to perform the matching of the key statistics used to characterize the TI: the histogram and the autocovariance.

- **Histogram Matching**

  The process of modifying the histogram of an image to best match the histogram of another image is a mature subject with many optimized approaches already in the literature (**?**). Since our code is in MATLAB^{TM}, we rely on the canned routine `histeq()`. Given standardized input and "target" images, `histeq()` works by finding the point transformation which minimizes the error between the CDF's of the target image and of the transformation of a flat histogram.

- **Autocovariance Matching**

  The autocovariance of a function $h(x,y)$ is the inverse Fourier Transform of its amplitude spectrum:

$$C(x,y) = \mathcal{F}^{-1}\{H(k_x,k_y)H^*(k_x,k_y)\} \tag{1}$$

  The amplitude spectrum of a 2-D function contains much of its pertinent spatial correlation information, and hence, much of its structure. The goal is to force another function, $g(x,y)$, to possess a similar structure as $h(x,y)$, without directly copying. First write $g(x,y)$ as follows:

$$g(x,y) = \mathcal{F}^{-1}\{|A|e^{i\phi}\} \tag{2}$$

  Our approach is simple; substitute the Fourier Transform of $C(x,y)$ for the amplitude of the Fourier Transform of $g(x,y)$:

$$g_{\text{out}}(x,y) = \mathcal{F}^{-1}\{|\mathcal{F}\{C(x,y)\}|e^{i\phi}\} \tag{3}$$

  The amplitude spectrum of a function contains the "textural" features of an image (shapes, trends, orientations), while the phase spectrum localizes these features in space. Note that in creating $g_{\text{out}}(x,y)$ from $g(x,y)$, the phase is left unchanged. If $g(x,y)$ is a random image, it will have the general appearence of $h(x,y)$, with a different (random) phase.

Algorithm

Now that the foundation for the Laplacian Pyramid and statistics matching has been laid, we present our algorithm for texture synthesis. In a statistical context, the goal is to compactly parameterize the TI's textural features in terms of a set of statistics, and then to impart these statistics on an uncorrelated image, thus producing an SI with a similar texture. However, as mentioned above, the TI's textural features are generally scale-variant, making the characterization of all features at once quite difficult. Ideally, the subbands of the Laplacian Pyramid each contain unique image features at different scales.

The following pseudocode paints a clearer picture of the procedure. The SI (`synthImg`) begins as a random image.

```
synthImg = makeRandomImg()
synthPyr = makeLPyr(synthImg)
trainPyr = makeLPyr(trainImg)

do( i = 1 : Niter) {
   synthPyr = matchCovariance(synthPyr,trainPyr)
   synthPyr = matchHistogram(synthPyr,trainPyr)
}

synthImg = reconLPyr( synthPyr)

output: synthImg
```

The SI begins as random numbers. We decompose both the TI and the SI into their respective Laplacian Pyramids, and iterate, performing the aforementioned covariance and histogram matching technique between corresponding subbands of each pyramid. Finally, the synthetic pyramid is reconstructed and output.

RESULTS

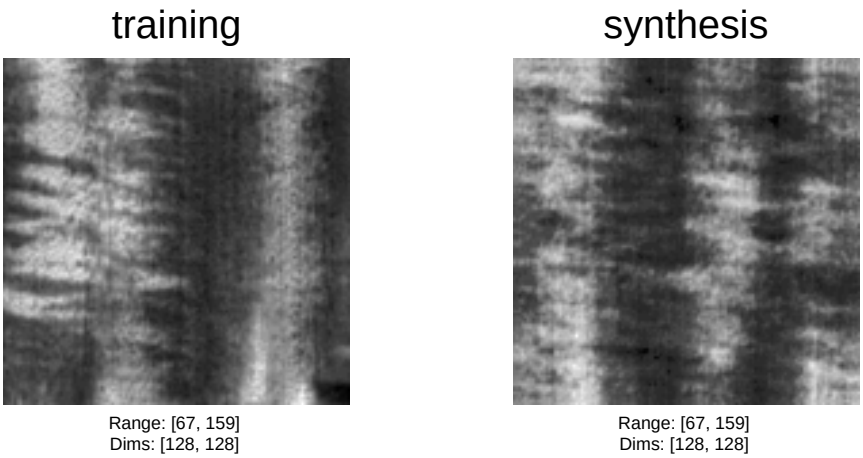
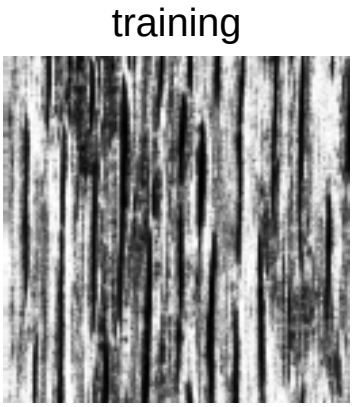


Figure 3: This method does a marvelous job of synthesizing the sepele bark image. Though it obviously has features with different characteristic size, the image is roughly stationary upon inspection, and its spatial features are not sharply localized in space, and is thus quite easy to synthesize. sepele [NR]

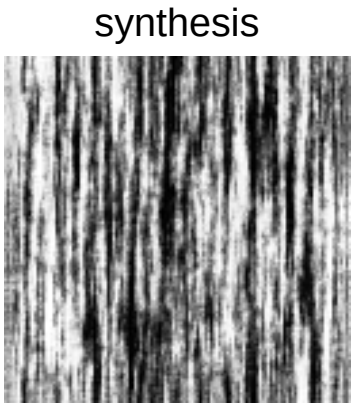
CONCLUSIONS

ACKNOWLEDGEMENTS

REFERENCES

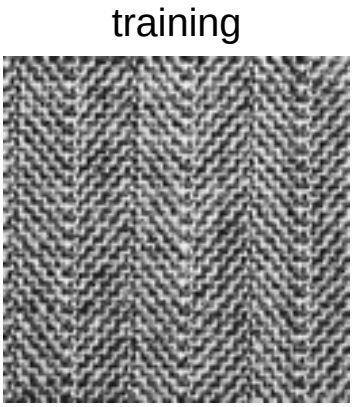


Range: [0, 255]
Dims: [128, 128]

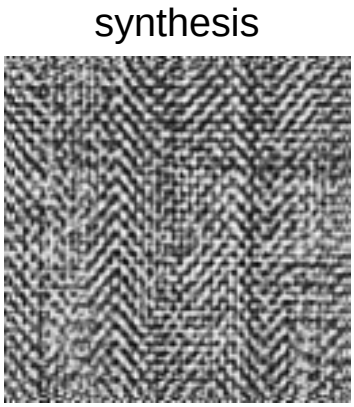


Range: [0, 255]
Dims: [128, 128]

Figure 4: Excellent performance. Both the texture of the wood and the lineations along its length are synthesized seamlessly. wood [NR]

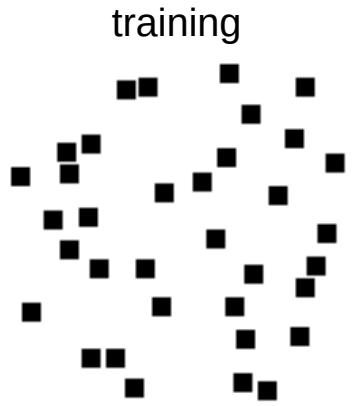


Range: [0, 255]
Dims: [128, 128]

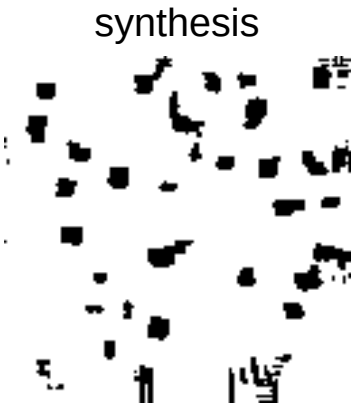


Range: [0, 255]
Dims: [128, 128]

Figure 5: This image is simply parameterized in terms of two-point statistics, which our algorithm uses, hence the excellent synthesis of the herringbone fabric image. This result effectively maintains the vertical homogeneity property of the TI. herringbone [NR]



Range: [0, 255]
Dims: [128, 128]



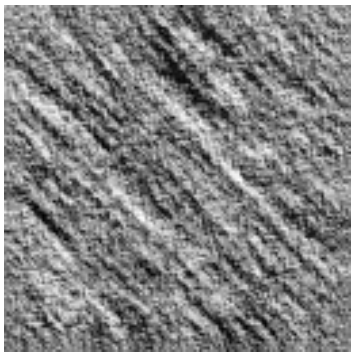
Range: [0, 255]
Dims: [128, 128]

Figure 6: The result is quite good. Many (not all) of the solid black areas in the output look like squares, and the relative density of squares is quite similar. rand-squares [NR]

training

Range: [0, 255]
Dims: [128, 128]

synthesis



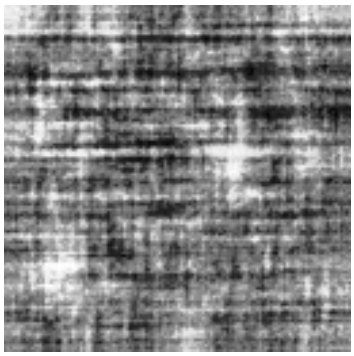
Range: [5.15, 255]
Dims: [128, 128]

Figure 7: At a first glance, one might expect a similarly striking result as the herringbone fabric synthesis. Unfortunately, the method was nowhere near as successful, though we don't characterize this result as a failure, either. The differences between this image and the herringbone are subtle: lower contrast in most regions, less regularity within patches of constant weave, some residual curvature in the lower portion of the image. fabric [NR]

training

Range: [10, 255]
Dims: [128, 128]

synthesis



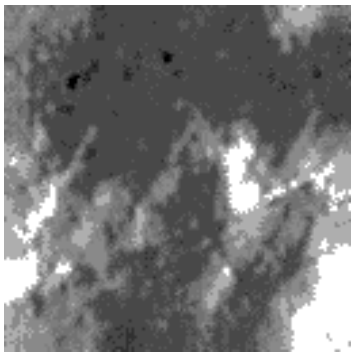
Range: [10, 255]
Dims: [128, 128]

Figure 8: Our method performs surprisingly poorly on this image. The complexity of the localization is severe; for instance, the randomly placed patches of constant gray are localized within individual bricks, and must be so in the SI. brick [NR]

training

Range: [0, 255]
Dims: [128, 128]

synthesis



Range: [0, 255]
Dims: [128, 128]

Figure 9: The simplistic appearence of this image is misleading, for it is highly difficult to synthesize. The features which we must synthesize are the edges, which occur at a multitude of different orientations. polygons [NR]