Heeger-Bergen Steerable Pyramid-Based Texture Synthesis Problem Set

Fall 2023 CS 283 Advanced Computer Vision Final Project

Sree Harsha Tanneru
Harvard University
sreeharshatanneru@g.harvard.edu

Annabel Yim
Harvard University
annabelyim@g.harvard.edu

Abstract

Texture synthesis is a notable application of image pyramids, which are multi-resolution representations used for analyzing images across various spatial scales. These pyramids are utilized to generate coherent textures corresponding to a source texture. Among the different types of image pyramids, steerable pyramids offer several advantages. They provide a rich representation that includes information about image scale and orientation. Additionally, steerable pyramids are multi-scale, oriented, translation-invariant, non-aliased, over-complete, and self-invertible [6].

During our class lectures, we explored the properties of steerable pyramids and their role in texture synthesis. To further enhance our understanding of the practical applications of steerable pyramids, we propose an implementation-focused problem set. This problem set is designed to delve deeper into the subject, involving exploration of the Heeger-Bergen algorithm, used for synthesizing grayscale textures. It extends the algorithm to adapt the synthesis of grayscale textures to different image sizes from the source texture, explores color textures, and addresses the handling of edges in textures.

Our work is in the following GitHub Repository.

1. Introduction

The Heeger-Bergen algorithm assumes that the first-order statistics of steerable pyramids can capture all spatial information characterizing a texture image. The process begins with a white noise image and alternates between histogram matching in the image and steerable pyramid domains. The aim is to match the output histogram with that of the input texture [2].

Although an ANSI C implementation of the Heeger-Bergen algorithm exists, there has been no publicly accessible Python implementation that utilizes existing Python packages [1]. For our problem set, we have developed a Python implementation of the Heeger-Bergen algorithm. This implementation leverages Pyrtools—a Python package for multi-scale image processing, adapted from Eero Simoncelli's matlabPyrTools—for steerable pyramid decomposition and image reconstruction [5]. Our goal is to provide a more accessible tool for the computer vision community, as many recent projects are Python-based. Furthermore, this Python implementation is designed to integrate seamlessly with existing Python packages, thereby enhancing its utility and application in various computer vision tasks.

1.1. Heeger-Bergen Algorithm

The Heeger-Bergen algorithm for texture synthesis is presented as follows:

Algorithm 4: Heeger-Bergen texture synthesis algorithm for grayscale images (without extension)

Input : Number of scales P, number of orientations Q, texture image u of size $M \times N$ such that M and N are multiples of 2^P, number of iterations N_{iter}
Output: Texture image v of size $M \times N$

1. **Input analysis:**
2. Compute and store the steerable pyramid with P scales and Q orientations of the input texture u.
3. **Output synthesis:**
4. Initialize v with a Gaussian white noise.
5. Match the gray-level histogram of v with the gray-level histogram of the input u.
6. **for** iteration $i = 1$ **to** N_{iter} **do**
7. Compute the steerable pyramid of v.
8. For each of the $PQ + 2$ images of this pyramid, apply histogram matching with the corresponding image of the pyramid of u.
9. Apply the image reconstruction algorithm to this new histogram-matched pyramid and store the obtained image in v.
10. Match the gray-level histogram of v with the gray-level histogram of the input u.
11. **end**
12. Return v.

Figure 1. Heeger-Bergen Algorithm [1]

A fundamental component of this algorithm is the steerable pyramid decomposition and image reconstruction. Although the original paper provides a detailed mathematical description of the construction of the steerable pyramid algorithm, we have opted to use Pyrtools for the steerable pyramid decomposition and image reconstruction in this problem set, due to the complexity involved in manually constructing them. For students wishing to gain a better understanding of the steerable pyramid decomposition, we have included an implementation that we created using Python at the end of the problem set.

In Pyrtools, the orientations (height) and scales (order) of the steerable pyramid can be specified as follows:

```
pyr = pt.pyramids.SteerablePyramidSpace
      (source_image, height=4, order=3)
pyr.pyr_coeffs
```

In this example, we obtain the steerable pyramid coefficients of an image, totaling $4 \times 3 + 2$ (the additional 2 accounts for the high and low-frequency residuals), resulting in 14 sub-bands for an image. Furthermore, Heeger-Bergen algorithm also utilizes image reconstruction. Since the steerable pyramid is designed to be self-inverting, we can reconstruct an image from its pyramid using the same filters employed in the decomposition process. This reconstruction can also be achieved using Pyrtools, as demonstrated below [5]:

```
res = pyr.recon_pyr()
```

1.2. Histogram Matching Algorithm

In addition to steerable pyramid decomposition and image reconstruction, another fundamental component of texture synthesis is the histogram matching algorithm. There are several methods for implementing this algorithm.

The original Heeger-Bergen histogram matching algorithm involves adjusting the histogram of an input image to align with a desired histogram shape. This is achieved using the cumulative distribution function (CDF) of one image and the inverse CDF of another. The algorithm calculates the histogram of the input image using 256 bins, providing a detailed representation of the image's intensity distribution. The CDF, derived from this histogram, maps the pixel intensities to a normalized scale. The inverse CDF performs the reverse operation, mapping from the normalized scale back to the original pixel intensity range. While this method allows for high precision in texture synthesis, it has a limitation due to unnecessary quantizations. The subband histograms of many "natural" images exhibit characteristic shapes, suggesting that fewer parameters than the 256 bins might be sufficient [2].

Consequently, we opted for a simpler histogram matching implementation by [2]. This method sorts the pixel lists of two images, u and v, and assigns to the pixel of u having rank k the gray-value of the pixel of v having the same rank. This algorithm ensures that the output has exactly the same histogram as the source image while preserving the relative rank of the input pixels [1]. The algorithm is illustrated in Figure 2.

Additionally, histogram matching can be applied to images of different sizes. When the gaussian white noise initialized input image has dimensions that are multiples of the texturized source image, modifications are made. For

Algorithm 3: Histogram matching

Input : Input image u, reference image v (both images have size $M \times N$)
Output: Image u having the same histogram as v (the input u is lost)

1. Define $L = MN$ and describe the images as vectors of length L (e.g. by reading values line by line).
2. **Sort the reference image v:**
3. Determine the permutation τ such that $v_{\tau(1)} \leq v_{\tau(2)} \leq \cdots \leq v_{\tau(L)}$.
4. **Sort the input image u:**
5. Determine the permutation σ such that $u_{\sigma(1)} \leq u_{\sigma(2)} \leq \cdots \leq u_{\sigma(L)}$.
6. **Match the histogram of u:**
7. **for** rank $k = 1$ **to** L **do**
8. $\quad u_{\sigma(k)} \leftarrow v_{\tau(k)}$ (the k-th pixel of u takes the gray-value of the k-th pixel of v).
9. **end**

Figure 2. Histogram Matching Algorithm [1]

example, if the source image v has size $M \times N$ and the input image has size $r_1 M \times r_2 N$, where $r_1, r_2 = 2, 3, \ldots$, line 8 of the histogram matching algorithm is replaced by [1]:

$$u_{\sigma(r_1 r_2 (k-1)+i)} \leftarrow v_{\tau(k)}, \text{ for } i = 1, 2, \ldots, r_1 * r_2$$

1.3. Color Texture Synthesis

For color textures, independently synthesizing each of the three color channels often leads to inaccurate colors. This is because the RGB components of a typical texture image are not independent; they are often correlated. To address this, Heeger and Bergen proposed using Principal Component Analysis (PCA) to decorrelate the color channels. By transforming the RGB color space into the PCA color space, the color channels become decorrelated and nearly independent. It's important to note that while the PCA channels are decorrelated in terms of their intensity values, they are not completely independent. This partial decorrelation is due to PCA focusing solely on color correlations and not on spatial correlations [2]. The Heeger-Bergen texture synthesis algorithm for RGB color textures involves the following steps:

1. Compute the PCA (Principal Component Analysis) in the color space of the input image u.
2. Determine the C=channels of u in the PCA color space.
3. Apply the texture synthesis algorithm shown in 1 on each PCA channel. This gives an output texture v in the PCA color space.
4. Convert the image v in the RGB color space by applying the procedure described below. The obtained RGB image is the output of the algorithm [1].

$$(v_{1,i}, v_{2,i}, v_{3,i})^T \leftarrow (m_R, m_B, m_G)^T + P(v_{1,i}, v_{2,i}, v_{3,i})^T$$

where P is the orthogonal basis matrix from PCA, and (m_R, m_B, m_G) are channel-wise mean values in original image.

The final output is an RGB image that results from this algorithm, representing a synthesized texture that maintains the statistical properties of the original while overcoming the limitations of color channel interdependence. In the problem set, PCA from sklearn.decomposition can be used to extend the Heeger-Bergen algorithm to color textures [4].

1.4. Edge Handling

Another extension explored in the problem set, related to the Heeger-Bergen algorithm, concerns edge handling. In the real world, textures are never truly periodic. However, pyramid decomposition, which is based on the Discrete Fourier Transform (DFT) that assumes input images are periodic, introduces discontinuity in edge handling. Proper edge handling is crucial as it ensures the synthesized texture appears seamless, without abrupt transitions at the edges. These transitions can lead to visual artifacts that disrupt the texture's coherence and uniformity. A straightforward method to mitigate this issue is to use mirror symmetrization at the borders. This technique works by reflecting the image across its borders, creating a repeating pattern, which effectively smoothens out the edges of the synthesized image. Additionally, although mirror symmetrization ensures continuity at the borders, it is not perfect, as it introduces artificial orientations in the input texture. Therefore, we will explore another edge handling method [3] by replacing the input texture with its periodic component. The period component approach aligns the texture pattern at the edges, making it useful for textures that have an inherent repetitive nature. The implementation details for calculating the periodic component are explained in the following Figure 3:

1. Compute the discrete Laplacian $\Delta_i u$ of u.
2. Compute the DFT $\widehat{\Delta_i u}$ of $\Delta_i u$.
3. Compute the DFT $\hat{p}$ of p by inverting the discrete periodic Laplacian:

$$\begin{cases} \hat{p}_{m,n} = \left(4 - 2\cos\left(\frac{2m\pi}{M}\right) - 2\cos\left(\frac{2n\pi}{N}\right)\right)^{-1} \widehat{\Delta_i u}_{m,n}, & \text{if } (m,n) \in \hat{\Omega}_{M,N} \setminus \{(0,0)\} \\ \hat{p}_{0,0} = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} u_{k,l} & \text{if } (m,n) = (0,0) \end{cases}$$

4. Compute p by inverse DFT.

Figure 3. Edge Handling: Periodic Component [1]

2. Methodology

In the problem set, we have formulated seven questions, each comprising several sub-questions, pertaining to the Heeger-Bergen Texture Synthesis Algorithm. The Google Colab Notebook for this problem set, along with its solutions and related texture images, is included with the submission for this project.

Q1.1 focuses on constructing the steerable pyramid decomposition. Assuming that Q1.1 is completed, Q1.2 then asks students to reconstruct the image from its steerable pyramid representation.

Q2 is concerned with creating the histogram matching algorithm, as illustrated in Figure 2. Q3 extends this concept by adapting the histogram matching function for scaled images, where the synthesized image and the input texture image do not have the same number of pixels.

Q4 relates to constructing the main architecture of the Heeger-Bergen Algorithm, utilizing the components developed in Questions 1-3. This process will follow the pseudocode presented in Figure 1.

Q5 aims to extend the approach in Q4 to color textures. It involves performing Principal Component Analysis (PCA) on the color channels, identifying the most significant component, conducting HB texture synthesis on this component, and then mapping it back to the color space.

Q6 and Q7 explore edge handling, guiding students to try out both mirror symmetrization and periodic components.

3. Results

In the problem set solutions, we can visually inspect how the synthesized textured images compare to the source textured images throughout the different algorithms that are used for texture synthesis.

4. Conclusion

In this problem set, we have explored the multifaceted applications and theoretical underpinnings of steerable pyramids in image processing. Through a series of practical exercises and theoretical discussions, we hope students have gained insights into the nuances of multi-scale and multi-orientation image analysis. The exercises were designed not only to provide hands-on experience with steerable pyramids but also to foster a deeper understanding of their significance in various image processing tasks. We hope this problem set enhances the students' technical skills in image analysis. As we conclude, we encourage students to delve deeper into texture synthesis by attempting to recreate our implementation of the steerable pyramid decomposition, which can be found at the end of the problem set.

References

[1] Thibaud Briand et al. "The Heeger & Bergen Pyramid Based Texture Synthesis Algorithm". In: *Image Processing On Line* 4 (2014), pp. 276–299. DOI: 10.5201/ipol.2014.79.

[2] D.J. Heeger and J.R. Bergen. "Pyramid-based texture analysis/synthesis". In: *Proceedings., International Conference on Image Processing*. Vol. 3. 1995, 648–651 vol.3. DOI: 10.1109/ICIP.1995.537718.

[3] L Moisan. “L. Periodic Plus Smooth Image Decomposition.” In: *J Math Imaging Vis 39, 161–179 (2011)*. 4 (2011). DOI: 10.1007/s10851-010-0227-1.

[4] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[5] Eero Simoncelli et al. *Pyrtools: tools for multi-scale image processing*. Version v1.0.2. 2023. DOI: 10.5281/zenodo.10161031. URL: https://github.com/LabForComputationalVision/pyrtools.

[6] Massachusetts Institute of Technology. *Lecture 7: Texture Analysis and Synthesis*. MIT Course 6.869: Advances in Computer Vision. 2019. URL: http://6.869.csail.mit.edu/fa19/lectures/notes_lecture_7.pdf.