Abstract

There are many traditional algorithms that fix patches or holes in pictures based on texture and or other features found in the same picture. In our project, we replace these patches with information from other pictures, thereby extending our sources and reducing the limitation in information that the original photo holds. Ideally, these features should blend in well and be contextually valid. We adapt the Scene completion algorithm by Hays and Efros [2007], which uses millions of photographs, for this project. However, we limit the number of photographs to a more manageable quantity that can be processed on a single computer by pre-labeling our stock images, so that the average person will be able to do perform this form of image editing on one machine. This would be like an additional 'What Could Have Been' function on the home version of Adobe Photoshop or Gimp.

1 Introduction

How many times have we been on that dream holiday, only to find out during photo development that the otherwise perfect picture is marred by a finger imprint of an inexperienced photographer. There are also instances when even a good photographer cannot manipulate the surroundings to capture exactly what he wants. If only the scaffolding weren't covering half the building, if only the crane weren't in the middle of that wonderful sunset, if only that stranger weren't in my group photo. Occasionally, we might also just want a change of landscape, or to combine the 'best of' elements in different photographs. If only I could have that backdrop from that photo in this photo instead. If only, if only... These creative alternatives form the motivation behind our project.

Finding an alternative for the repairing of the image not only gives us the freedom to inject new elements into the picture, it is also a better way of hole patching compared to interpolating existing pixels in the original image, or to duplicate elements in the original image. Interpolating and then blending can result in a blurry patch that is far from natural. Meanwhile, duplicating existing features in the image, while contextually valid, is easy to spot since human sight is sensitive to recurring patterns.

Figure 1: Man obstructing part of picture

Figure 2: Poisson blending of hole in original image [Efros, Hays. 2007]
Figure 3: Duplicating existing features in original image to fill hole [Efros, Hays. 2007]

Having said that, isolating only the contextually valid elements to insert into an image hole is one of the challenges when selecting from a wide range of sources. We address this by pre-labeling our database images when we download and save them. This reduces our selection process to locally matching the selected area of interest amongst a pool of contextually valid images.

2 Prior and Related Work

In Efros and Hays' work [2007], they utilized 2.3 million unique images on a cluster of 15 machines and used the gist scene descriptor developed by Oliva and Torralba [2006] to group and recognize their pictures. As with Wilczkowiak et al. [2005], their seam finding operation relaxed the use of the remaining valid pixels in the original picture as hard constraints, allowing said pixels to be removed and replaced by new content.

Other algorithms to fill an unknown area with new information are based on example-based texture synthesis. These make use of texture and contour information from adjacent pixels, which are then extrapolated to fill up the missing area. Authors of such algorithms include Wilczkowiak et al. [2005] and Wexler et al. [2004].

There are also methods that use multiple photographs of the same area in order to accurately reconstruct the image, such as those developed by Agarwala et al. [2004].

3 Preliminaries

All images used were scaled down to a size of 512 by 512 pixels using the bicubic interpolation method. In addition, images that were too small and images that were duplicates were discarded.

We obtained our database of images using Google Image Swirl, which “organizes image search results based on their visual and semantic similarities”. When we saved those images, we labeled and categorized them according to their underlying themes, e.g. 'Road', 'Sky', 'Lake' etc. Doing this offers us two advantages:

One, the element chosen to fill the hole will have the relevant context. After all, local image matching is limited to local gradient information and pixel-by-pixel information, with no regards for the big picture. Pre-labeling ensures that we still retain information of the context of the picture as a whole, reducing instances where images have extremely out-of-place elements.

Next, pre-labeling sifts out the irrelevant images, thus reducing the number of images to be considered for local image matching. Since local image matching is the most computationally demanding step in our program, this confers us significant time savings.

For each image category, we have about 20 pictures. Themespaces, which capture the main characteristics of the scene, of each of these categories were constructed. Given a new image, its projection onto these themespaces can be compared with the actual image to determine how well it fits into a particular category.

Each database image was first reshaped to a column vector. 20 such vectors from the 20 pictures in a category were then used to form a matrix, A. The mean of A was subtracted, and the first 5 singular vectors from this resultant matrix were found. The themespace, corresponding to a certain theme category e.g. sky, was then formed from the linear combination of these 5 singular vectors.
4 Description of Algorithm

A. Selection Of Area

The user first selects the region where he wants replaced using a program like Paint, where a large degree of freedom is given for selection of an irregularly shaped area. A binary mask of the selected area is then created.

[Image: Binary mask of selected area]

B. Classification

The input image is classified via matching of its projection onto each of the themespace as mentioned in the Preliminaries section.

This projection is given by:

\[ x_{\text{proj}} = \text{mean image} + \sum_{p \in S} \alpha^{\text{opt}} \mu_i \]

where

\[ \alpha^{\text{opt}} = \arg\min_{\alpha} \| (x - \text{mean image}) - \sum_{i=1}^{r} \alpha_i \mu_i \|^2 \]

The input image is projected to each of the themespace and the resulting error calculated. Images from the 3 categories which yielded the least error are then used for local context matching with the input image.

C. Local Context Matching

The local context used for matching is defined to be the 10 pixel band along the boundary of the seam formed by the binary mask. This local context is used for comparisons among the relevant images (60-80 images) as sifted out by the classification process.

The cost function associated with each replacement image is formulated based on three considerations. First, the distance of each pixel along the seam to the original binary mask defined by the user. Second, the magnitude of the gradient of the sum of squared differences (SSD) between the original image and the replacement image along the seam. The final consideration is the mean squared error (MSE) in intensity between the original image and the replacement image over a band of width 10 pixels from the seam.

The seam is found by obtaining the local minimum of the following cost function around the original binary mask defined by the user.

\[ C(S) = \sum_{p \in S} [C_d(p, M) + | \nabla SSD(p) |] \]

Where S are the pixels which form the seam, M is the original binary mask defined by the user and B is the 10 pixel band from the seam S.

\[ C_d(p, M) = (k \ast \text{Dist}(p, M))^3 \]

\( C_d(p, M) \) assigns a cost to each pixel from the seam that increases with distance from the original binary mask. The value of k was empirically set to 0.02.

At regions that are missing for the original image, \( C_d(p, M) \) is very large to prevent the seam from crossing regions removed by the user.
\(|\nabla \text{SSD}(p)|\) is the magnitude of the gradient of the sum of squared differences between the original and replacement image at pixel \(p\). Using the gradient in the cost function causes the seam to avoid paths of high frequency changes in color. The high frequency edges are difficult to blend and may appear unnatural in the blended image.

The seam is originally assumed to be the boundary formed by the original binary mask. The local minimum of \(C(S)\) is then found by shifting \(S\) towards pixels of lower cost.

The final cost function is evaluated by adding the MSE along the 10 pixel band along the seam to \(C(S)\). The individual components of the cost function are then weighted to give roughly equal contributions.

For each replacement image, the cost function is calculated and the four matches with the lowest scores are selected for blending.

D. Laplacian Pyramid Blending

After selecting the four lowest cost replacement images, we perform Laplacian pyramid blending to obtain the composite output image.

An \(N\) by \(N\) image can be represented as a pyramid of \(1\) by \(1\), \(2\) by \(2\), \(4\) by \(4\), \ldots, \(2^k\) by \(2^k\) images, where \(N = 2^k\). The image is sub-sampled as it moves from level 0 (512 by 512) to level \(k\) (1 by 1). Gaussian pre-filtering is applied to the image before sub-sampling.

In pyramid blending, first, two Laplacian pyramids \(L_A\) and \(L_B\) are built from the replacement and original images respectively. During implementation, the Laplacian pyramids are approximated using Difference of Gaussian pyramids.

Next, a Gaussian pyramid \(G_R\) is built from the selected region \(R\) represented by the local minimum seam, \(S\).

A combined pyramid \(L_S\) is formed from \(L_A\) and \(L_B\) using nodes of \(G_R\) as weights:

\[
L_S(i,j) = G_R(i,j) \cdot L_A(i,j) + (1 - G_R(i,j)) \cdot L_B(i,j)
\]

The \(L_S\) pyramid is then collapsed to get the final blended image.

E. Mode And Results Selection

The program will have available, two modes from which the user can select from. Normal mode, which would search amongst images that are contextually similar. Random mode, if the user would like to see more possible combinations that may, or may not be semantically valid. This mode will search for good local context matches among a larger group of images, generating more possibilities, but also taking up more computation time.

Four blended composite images with the lowest scores are then presented for the user to choose from.
Experimental Results

We have attached some results from our test runs on page 6. The user is able to choose among four images that look different from one another.

For our best results, the composite image has elements that fit within the general context of the picture, is well blended and looks natural.

Meanwhile, the less-than-desirable composite images may contain elements that are obviously out of scale with the rest of the picture, have visible blending artifacts, have elements with unnatural cut-offs and/or strange illumination patterns.

The Matlab implementation of this algorithm takes approximately three to five minutes to run on a single computer for one input image with a database of about 100 images.

Conclusion

This project, in summary, covers the local context matching section of the paper by Hays and Efros [2007], since the semantic scene matching section is done via pre-labeling of downloaded images. This makes it a more manageable task that can be done with limited computational resource. However, the database can be expanded to include a much larger number of images, which could give us combinations with much lower scores compared to what we currently have with a smaller set of images.

Another area that can be further developed would be to take the scales of both original and replacement images into account. When the replacement element is interleaved into the original, it may so happen that the new element is of a much larger or smaller scale compared to other elements in the background, making the composite look unnaturally disproportionate.

Finally, a less local method of selecting areas from the replacement image could be looked into. This would prevent instances when part of the replacement object is cropped off. An object preserving component could be included that defines the edges that make a particular feature 'whole'.

Bibliography


Appendix

Tian Kai: Section A, Section B, Section C, Code testing and editing

Huimin: Preliminaries, Section D, Section E, Report drafting and editing
Figure 7: Example images. Input images and replacement images are blended to form composite output images. Top four matches associated with each input image are displayed.