Scene Completion with a Million Photographs Revisited

Joshua Kalapos
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh PA 15213
jkalapos@andrew.cmu.edu

Abstract

In this paper we revisit Scene Completion with a Million Photographs, a paper written by James Hays and Alexei Efros. The purpose was to take images that have missing/unwanted regions and replace these regions with new, convincing visual data. To complete these images, we rely on finding semantically matching images and then combining them with our original photo the best we can along optimized cuts. With our new implementation, we recreate scenes from the original paper, and possibly some new scenes as well.

1. Introduction

Sometimes when we have an image there are often portions of it that we later wish were removed. For example, a telephone pole might be ruining a pristine picture of the mountains, or a stranger is photo-bombing your family photo in the background. In other cases, the photo might simply be damaged or missing information. Both of these problems lead us to the task of image completion. The goal of image completion is to take these photos and replace any regions with new information such that this replacement is imperceptible.

In this report, we recreate the results of the 2007 paper, Scene Completion with a Million Photographs [2]. When presented with an image with regions to fill in, Hays et al. attempt to fill in those regions with information that *could have been there*. This definition leaves a lot open to human perception, as what *could* be there relies on the context of the image. An example would be putting a cruise liner in a pharmacy parking lot. No matter how beautifully blended in it could be, the cruise liner could not have been actually there.

While other techniques at the time sought to resolve the regions in images by using information from the same image, Scene Completion with a Million Photographs takes in a semantically similar image and fills in the region with that new data. The advantage here is that image completion using only the original image assumes that all the visual data needed to complete any gaps is available in that same image. This, Hays et al. believed, left the problem too constrained, and limited the number of actual problems that could be tackled. For cases in which there is little useful texture for the region that needs to be filled, these other techniques will suffer.

To find a new image to use, Hays et al. search through a very large database of photographs. The insight is that, though the space of all possible images is essentially infinite in size, the space of semantically similar scenes is small. Given some image, we will be able to find a set of similar images, and can use them to fill in regions with appropriate visual information. Hays et al. describe this as semantic scene matching and use gist descriptors of all the images to find the closest images in the database to the query image. While interesting, this portion did not get implemented in this project, and we implement what comes after finding the right image to use.

2. Overview

After deciding which new image to use, it is time to do form a single complete image. To be clear, our input is the original image, a mask that indicates what part of the original image we want replaced, and the new image. It's recommended to keep the mask limited to one contiguous region, because stretching a single new image to fit more than one (although similar) contexts would confuse the image alignment.

The two images using the mask are combined together using Poisson blending. However, to improve the blend and to avoid artifacts we employ two preprocessing strategies. To start, the matched image gets scaled and translated to best align with the original image. Oftentimes, we noticed that images found to be semantically matching come pretty well aligned with the original image, but in some cases an additional explicit alignment step helps "alright" matches.

Masks defined by the user are often not perfect, as they can be either too coarse or do not follow along a good cut to avoid drawing attention from attentive human eyes. With

a graph cut segmentation that starts with the original mask, we can find a new boundary for the Poisson blending that has the minimum image gradient magnitude. Meaning at the seam we have a small intensity difference that takes advantage of human perceptual sensitivity. In Section 4 we will later see the advantage of using the new graph cut mask.

3. Implementation

Everything in this section was done in Matlab.

3.1. Image Alignment

Before any work is done on the two images, we first translate the matched image to best "fit" with the original. This is accomplished by doing template matching between the two images. By focusing around the region around the mask (about 80 pixels), we create a small template that we will translate the matched image until we find a good alignment. This good alignment is found by calculating the cross correlation for each possible shift, and then finding the translation that delivers the best correlation. For speed, rather than starting out matching the full resolution images, we perform pyramid alignment. With pyramid alignment we find the correct alignment at a smaller resolution, and use that translation as a starting point for our higher resolution image. We eventually scale back to the resolution of our original images.

3.2. Graph Cutting

When combining the two images, we want them to merged along a boundary that has the minimal difference. To find this boundary, we use a strategy described by Kwatra et. al [3]. First we convert our image into a connected graph, with edge weights representing the difference between pixels. We then use this graph as an input to the max-flow/min-cut problem. As an output from the max-flow/min-cut problem, we will get a cut that will separate the graph into two components. These two components will then form the new mask for Poisson blending.

To create the graph, we represent it as a sparse matrix in which every entry represents the weight of an edge between two pixels. A sparse matrix must be used in implementation because the size of the matrix is extremely large (n pixels by n pixels), and the vast majority of the entries will be zero. Kwatra et. al suggests that this weight is the sum of the squared difference between the first image and the second image, plus the squared difference between the first image and second image at the neighboring pixel.

$$M(s,t,A,B) = ||A(s) - B(s)|| + ||A(t) - B(t)|| \text{ (Eq. 1)}$$

Where s is a pixel, t is one of its neighboring pixels, A is the original image and B is the matching image. Note that we have flattened our images to be vectors here. The flattening can be row-major or column-major as long as this decision is consistent. In addition to weighting using pixel intensity differences, one could also weight using gradient differences. The advantage being that color differences can still lead to passing through high frequency edges even when a constant color seam is found. Minimizing along the gradient seam can both find a constant color seam and avoid cutting high frequency boundaries.

In addition to labeling cost for separating pixels, we must provide sink and source edge weights to fully formulate the min-cut problem. The sink and source in this case each being masks for the original and matching image that determines which final pixels **must** come from their assigned image. Within our matrix, enforcing the sink and source comes by assigning a cost of infinity for assigning the pixel the wrong label. The source mask itself in our approach is what must come from the original image. This mask gets generated by taking the original mask, blurring it, then negating it. These simple steps provide a buffer region around the original mask that the cut is allowed to reside in and prevents the cut from going unexpectedly deep into the original image. The sink mask is the same as the original mask and shows all pixels that must come from the new image. We made this choice on the assumption that everything within the original mask must be replaced.

After we have formulated the costs in a sparse matrix, we use a Matlab wrapper for Boykov and Kolmogorov's min-cut/max-flow implementation [1] in C++. Using this heavily optimized version of the solution helps solving such a large graph problem happen within a reasonable amount of time.

3.3. Poisson Blending

With the help of a mask based on the optimized cut between two images, we can employ simple Poisson blending to actually put the two together. To blend, we want to impose our matched image A onto the target, original image B, but only within the masked area M. As opposed to simply copy pasting image A into image B ($C = A(1-M) + B(M)$), Poisson blending creates seamless blends by finding the solution to a Poisson problem. This requires us to put our images and mask in the appropriate terms.

Poisson blending is described by Perez et. al [4] and requires us once again to create a sparse n by n matrix. Where n is the number of pixels in the image. We have aligned our matched image and made it the same size as the original image by scaling or by setting pixels outside the region of interest to zero. The paper says that for every pixel under our mask (pixels we want to replace), it must satisfy this equation.

$$\forall p \in \Omega,$$

$$|N_p| f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in N_p \cap \partial\Omega} f_q^* + \sum_{q \in N_p} v_{pq}$$

(Eq. 2)

Where f^* is our source (matched) image, g is our target (original) image, p is the pixel, Ω is pixels under the mask, $\partial\Omega$ are the pixels who have at least one neighbor under the mask, N_p is the neighborhood of pixel p, f_p is the new value of p we are solving for, f_p^* is the value of pixel p in the soruce image, and v_{pq} is the gradient of the pixel to a neighbor in the source image g.

If we want to solve this equation for f_p, then we can simplify it to that each f_p is the average of its neighbors plus the gradient of the pixel to each of its neighbors. These neighbors can either also be new values (unknown) if they are under the mask, or known if we are on the border of the mask. Thus, elements of our sparse matrix must be set according to the following constraints.

- If a pixel with index i is not under the mask, mark $M[i][i]$ with a 1 and vector $b[i]$ with the target image's pixel.
- If a pixel i is under the mask, then set $M[i][i]$ to the number of neighbors it has (usually 4 unless on border of image). Then, the following must be done for each neighbor. $b[i]$ starts at zero here.
  - If the neighbor ni is under the mask then set $M[i][ni]$ to -1.
  - If the neighbor ni is not under the mask, we add the value of our target pixel to $b[i]$.

  For all neighbors, we also add to $b[i]$ the gradient between the starting pixel and the neighbor pixel.

Using these constraints gives us the matrix problem of the form $Ax = b$. So, we can solve the equation to get our final output image x. In our implementation the color channels get separated and we solve a Poisson problem for each channel to get our final result.

4. Results

To obtain our results we obtained images and masks from the dataset provided by Hays et. al from their paper's website. These images were used as the "original" images, with the masks indicating which region should be replaced. For our semantic matches, they were found in one of two ways. First, as the paper demonstrated some example matches that were found, we used those same images. For the second approach, we found matches by looking at other provided datasets or by looking up specific themes in the image online. Both of these strategies produced outputs of similar quality.

Figure 1. Example of result from our implementation. Top left is original photo and bottom left is the output.

For the first result, we take a scene of an ocean beach as seen in Figure 1. In our original image (top left), our scene is disrupted by the roof of the house. Arguably, the picture is ruined. But, with the help of our scene completion, we can indicate the region we want to be replaced (the roof) and find an image that matches our original scene. In this case, like in the original paper, we use a body of water with boats in it. The bottom right is our combined output. When combining the two bodies of water together, our graph cutting step was able to find the best place to count as the boundary for the Poisson blend. With water, the visual artifacts of not doing the graph cut are minor so we will draw from a different example.

With our next example, we have a sky full of color but a poorly lit town on the bottom. It's pretty subjective on whether the town should be removed, but say we replace the town with the information from a different sky with a sunset. Using our mask, graph cut and Poisson blending gives us our final result on the bottom left of Figure 2. For contrast, without graph cuts, although also minor, there are now artifacts that would hint to the viewer quickly of the image's falsity. In Figure 3, near the sun Poisson acts up and produces a green color. Likely because of some strange blend along the boundaries and the large gradients in different color channels. Additionally, on the right side of the image, we can identify the location of where the tower from the original image was removed. The cut of the original mask was not situated at the best location, so Poisson was not able to seamlessly combine the two images as well as it

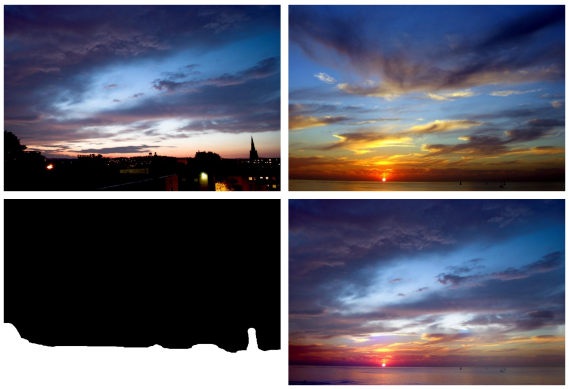

Figure 2. Removal of town landscape with sunset.

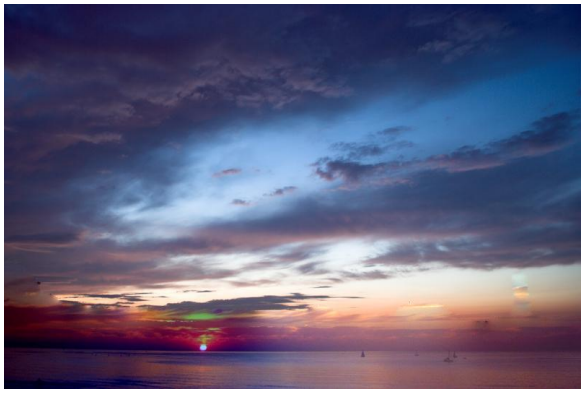

Figure 3. Same nigh sky landscape as Figure 2 but with no graph cut beforehand.

could have.

Worth noting also is scene completion with just copy pasting after solving for the graph cut mask. Such a result can be observed in Figure 4. The original photo was a town with a grassy hill background, and we have decided to replace the background with a lake and mountain range. By just creating a new image without the Poisson blending step, we can see a lot of benefit from the cuts not interrupting the details of the town. For the most part visually the eyes are not too bothered because our attention is not as drawn to the boundary where the two images meet. However, upon closer inspection, the viewer can notice issues such as the tree on the right containing a different color background behind its leaves, or the little imperfections along the unblended border with the water.

4.1. Issues

Even with graph cuts improving the quality of our Poisson blend, there are some unavoidable issues that this approach has with scene completion. The first problem would be that, though we attempt to cut along lower frequency

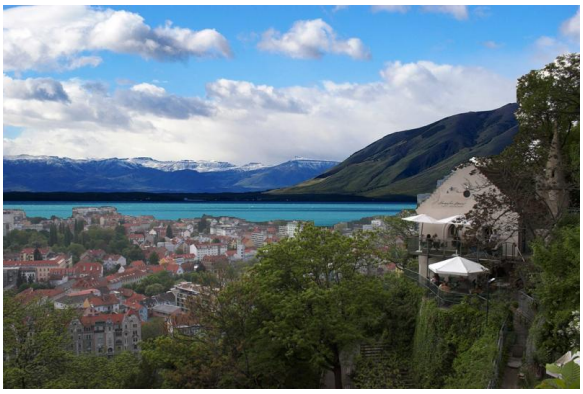

Figure 4. Town with background completed with just graph cut and copy paste.

Figure 5. Image completion of the right wall with a matched image may sometimes bring unwanted parts of objects.

edges between the two images, this does not account for cutting across complete objects. For example in Figure 5, blending an image with a car but only taking half of the vehicle results in a confusing ghost cars. Though there is some context built into the original search for matching images, the actual completion of the scene does not have an awareness of objects in the scene. This unawareness can often result in other unusual outcomes, such as partial people.

Another issue that arises is in scale. Hays et. al suggests that multiple scales can be tried during the alignment phase, but such maneuvers do not solve the algorithm's lack of knowledge about the scale of objects. Relative to the ocean

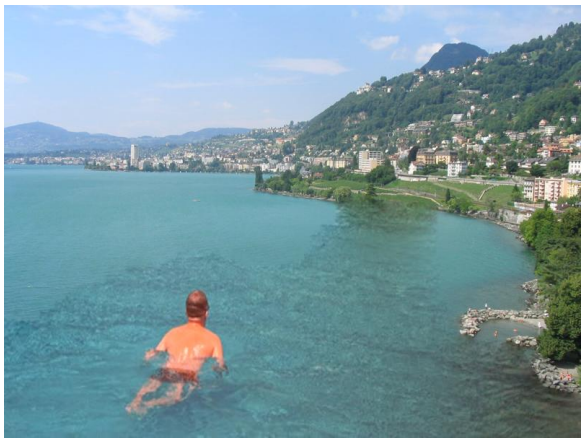

Figure 6. Monster swimmer approaches peaceful city.‘

scene from figure 1, people should be relatively small, but, with the right/wrong match, we can have outcomes like in Figure 6.

A final issue would be that sometimes the completed region’s perspective does not match that of the original image, giving the strange sensation of a warped reality. Though all of these issues can present annoying obstacles, all is not lost as, if one matched fails, there are many other images that may potentially work better. Even in their paper Hays et. al say that sometimes it comes to trial and error, which is why the semantic matching presents a multitude of best matching images, rather than just a single one.

5. Conclusion

For scene completion, the primary goal is to create artificial images that can trick the human eye into believing them to be real. Such a quality is not easy to measure, as the only source of that kind of evaluation is to ask a person. Still, even techniques that put a little consideration into the human mind (choosing semantically similar images, cutting along low frequencies), can go a long way in creating convincing outputs. Not flawless as discussed, as just about all strategies concerning human interpretation will find it hard to cover all cases. But, with this technique and some additional effort on the human side for choosing input, we come pretty close to accomplishing this imperceptibility of modification.

References

[1] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, Sept. 2004.

[2] J. Hays and A. A. Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), 2007.

[3] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003*, 22(3):277–286, July 2003.

[4] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, July 2003.