# Project 2: Image Stitching

**B03902124 黃信元, B03902008 林煦恩**

In this project, we have used Multi-Scale Oriented Patches for feature detection and description. And used kd-tree for fast feature matching. Then we used RANSAC for robust image matching. After that, we have performed a global brightness shift. And we have implemented several blending techniques. Finally, we perform contrast limited adaptive histogram equalization via openCV.

## I. Software Usage:

We have written the project using C/C++, and has used openCV for imageIO and linear algebra calculation. Therefore the computer needs to have both openCV and CMake installed.

After installation, please type: `cmake .; make;` in the current directory. This creates several executables in this directory.The usages are as follows:

```
./MSOP <Image_Folder> <Temporary_Folder>
./blending_* <Temporary_Folder>
```

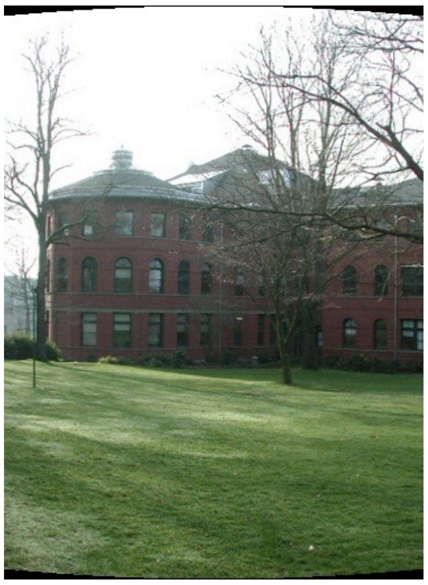
`<Image_Folder>` should contain all the images needed to create the panorama and a metadata named "`img_list.txt`" with first line denoting the focal length of the camera (measured in pixels), and subsequent lines denoting the image names. Note that the order of the image name is crucial. Basically it needs to be order from top to bottom such that the images are pasted from left to right, which is similar to how iPhone panorama works. `<Temporary_Folder>` is the folder name that will be used as the input to `blending_*`. All the executables starting with `blending_` will create a panorama (named "`panorama.jpg`") using the corresponding blending strategy.

## II. Image Warping:

To make different pictures fit more perfectly, we have applied cylindrical warping. This is first done by finding the focal length of the camera in terms of pixels. This can be done in several ways, and we have chosen to use the estimated output of **autostitch**.

After we have obtained the focal length, we used inverse warping to obtain the warped picture. Since inverse warping may result in floating point pixel, bilinear interpolation is used to solve this issue. The warped picture is shown in the middle figure. However, the inverse location of pixels in the warped picture may lie outside the original picture, so there are black regions in the middle figure. To make the later step more robust, we decided to crop the picture in this step. Note

Original Picture


Warped Picture

Warped & Cropped Picture

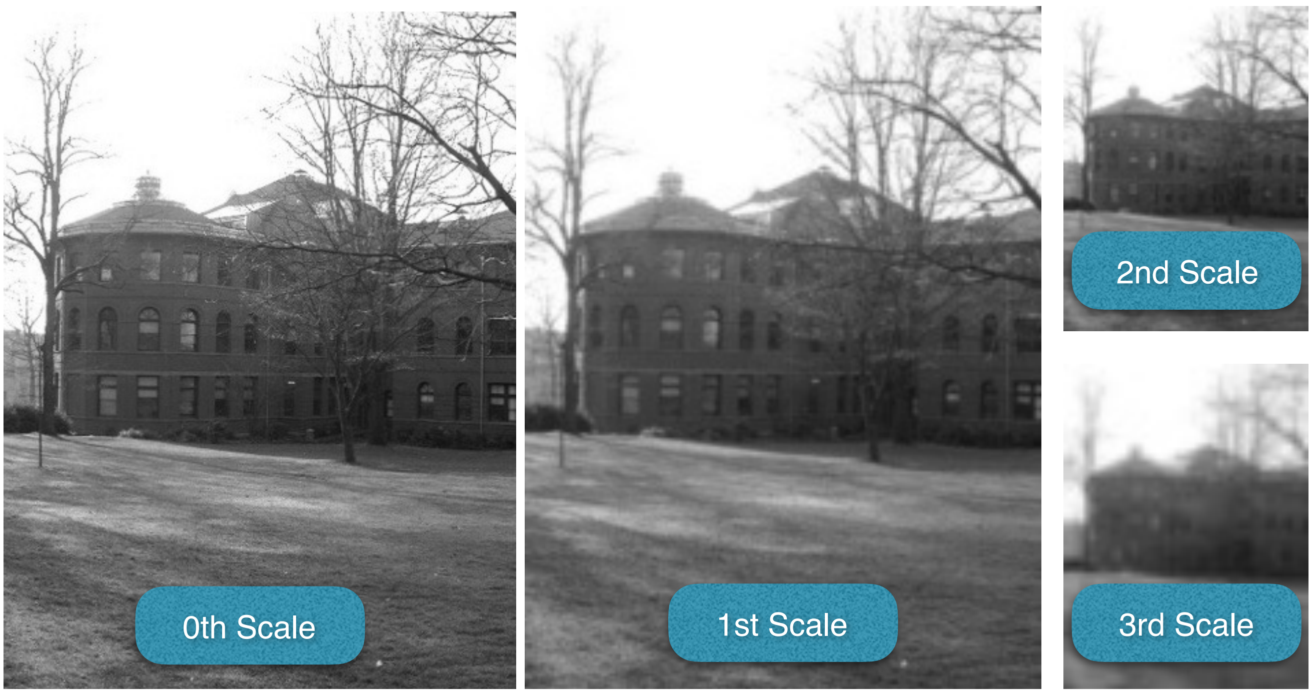
that including black region will not cause error in later steps, however we want to be able to distinguish originally black pixels and unavailable black pixels, so the black region in the picture's boundary is stored as (-1, -1, -1) in the matrix. And we suspect that storing negative value may cause error in later steps, so we decide to crop it. The cropped picture is shown to the right.

Now we have all the input pictures warped, we then apply the feature extraction method on the warped pictures in Section III. Another option we have considered is to first apply feature extraction and then apply cylindrical warping (warping the feature locations together with the image but using the original descriptor). We think this may not be suitable, since local information will also be warped and may not be able to perfectly match the features among different pictures (since different locations are warped differently).

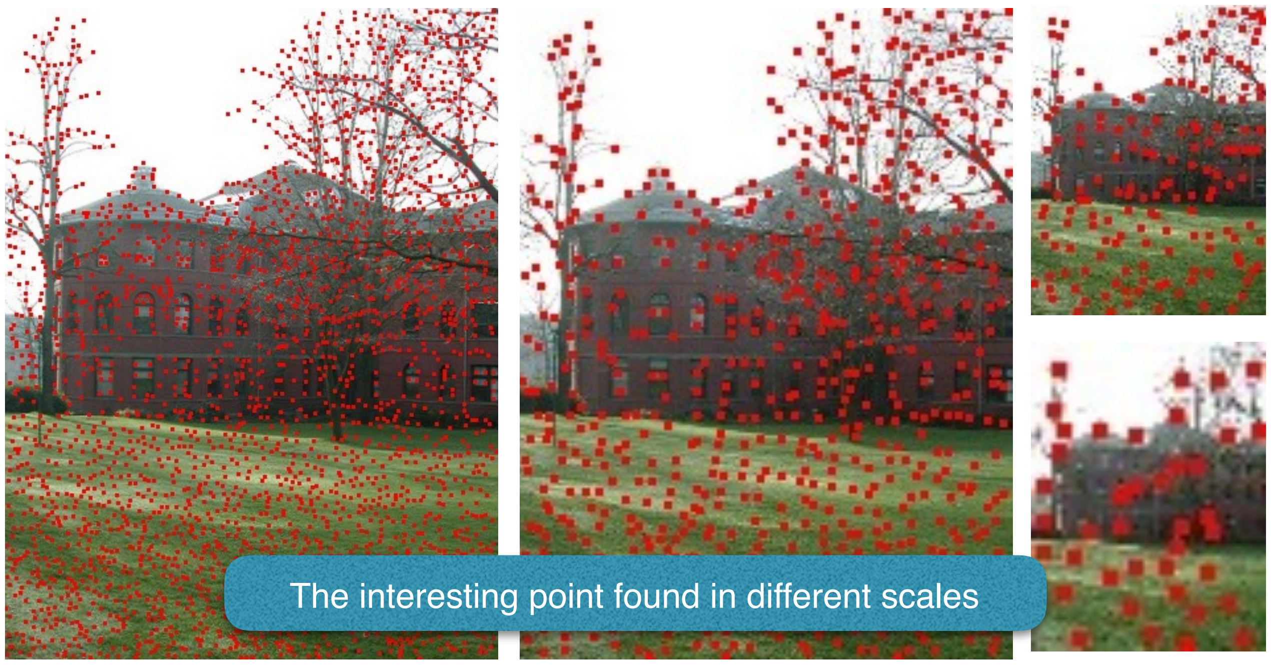
## III. Feature Extraction and Description:

For feature extraction and description, we have used the procedure in "Multi-Image Matching using Multi-Scale Oriented Patches" by Brown et al. [ref. 1].

First of all, multi-scale oriented patches have used different scale of the picture to capture the features. Each scale is obtained by Gaussian blurring the image and subsampling the pixels. For Gaussian blurring, we have used a finite support to speed up the process. We have used a square support with side length: (sigma - 0.35) / 0.15. This value may be a floating point number, so we have taken the smallest odd number greater than the above formula. In each stage, the subsampled image will be 1/4 the size of the pre-subsampled image. The 0th scale uses the original picture, while the 1st scale is 1/4 the original size and 2nd scale is 1/16 the size … etc. We only consider 4 scales, because the last scale is already 1/64 the original size so is quite small and blurry. Note that we only consider the intensity map of the original picture in finding and describing the features. Different scales are shown in the following pictures (the size for 0th and 1st scales, 2nd and 3rd scales, are actually not equal, it is just for better visualization).



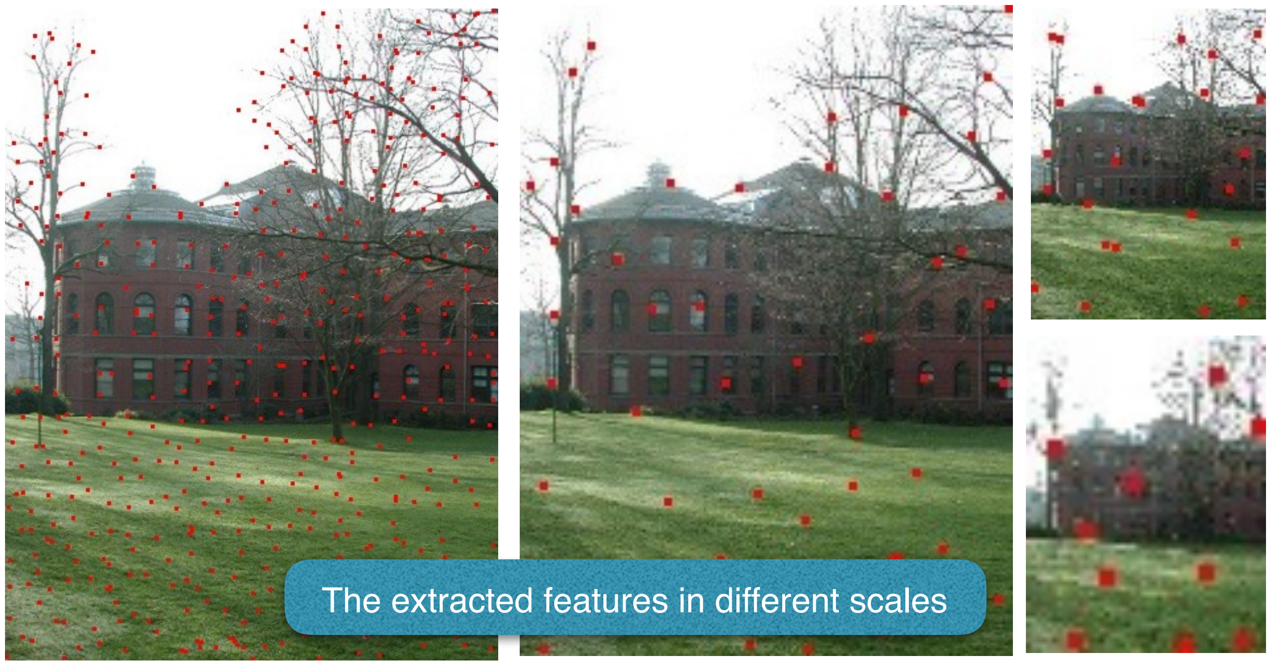
Then we find the features completely independently for the different scales. Each scale will find a predefined number of features. In our case, we have used 500, 50, 25, 10 for the 4 scales from 0th to 3rd. The following considers different scales separately. First of all, the goodness of a pixel is defined as the harmonic means of the pixel's Harris matrix. We then filter out pixels that is the local maximum around it's 3x3 neighborhood and have goodness > 10. The found pixels, called interesting points in the paper, for different scales are shown in the pictures next page.


The interesting point found in different scales

As suggested in the paper, to make the features more spatially separated, we have applied adaptive non-maximal suppression. This is done by first calculating the suppression radius for each interesting points (given in the following formula). We have used a simple brute force method

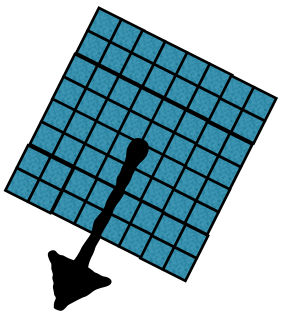
$$r_i = \min_j |\mathbf{x}_i - \mathbf{x}_j|, \text{ s.t. } f(\mathbf{x}_i) < c_{\text{robust}} f(\mathbf{x}_j), \ \mathbf{x}_j \epsilon \mathcal{I}$$

to find the radius so the calculation can be time consuming. Therefore we first select 20 * (predefined # of features) of the interesting points with the highest goodness to speed up the calculation. After then we pick predefined # of features having the largest suppression radius over each levels. As suggested in the paper, we have also conducted sub-pixel correction. Since we only need the inverse of a 2x2 matrix, simple formula for the inverse is used. The extracted features are shown in the following pictures.


The extracted features in different scales

Now to describe each feature, we first calculate the orientation for each feature points. This is defined to be the direction of the smoothed gradient at each feature points. However, since we will only consider translational camera model in later steps, this orientation is actually not needed.

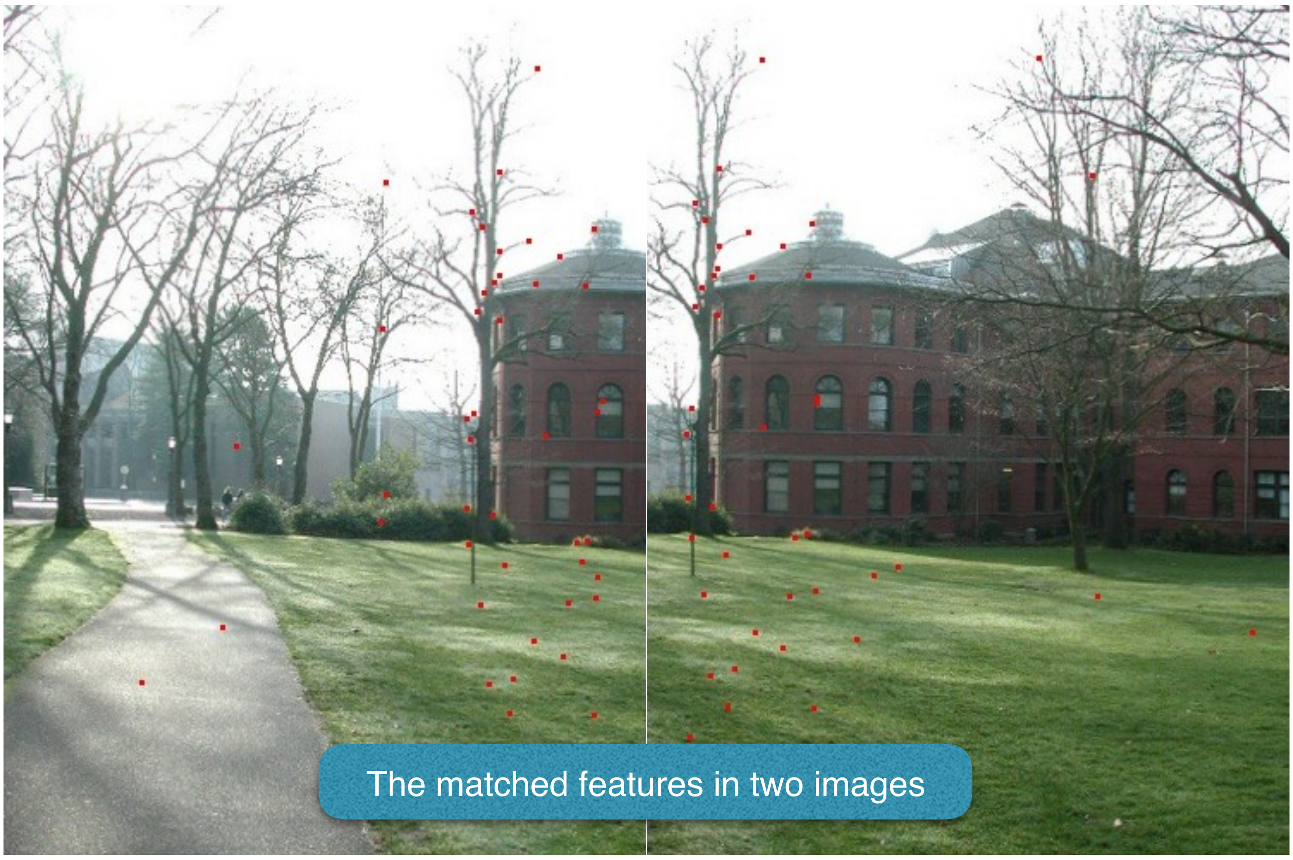
Then we form a 8x8 grid around the feature location (with sub-pixel correction). This is illustrated in the picture to the right. The center is the location of our feature point. For each of the 64 cells, the intensity of each cell is defined to be the intensity at the cell center. Similarly we have used bilinear interpolation to get the intensity if the pixel index is a floating point number. The final step is to apply normalization to this patch of intensity (a 64D vector), so the patch has mean = 0 and standard deviation = 1. Thus the feature will be invariant to affine changes in intensity.



## IV. Image Matching:

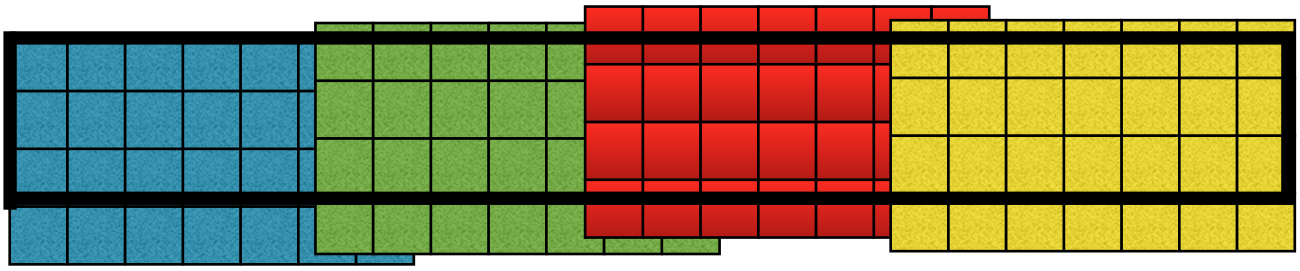
In this section, we used the features found in the Section III to match the images. As we have mentioned in Section I, we consider the images to be sorted. Thus successive images will be matched together with the later being paste roughly to the right of the former image. (similar to iPhone's panorama) However, we still consider the general 2D camera translation model (instead of a 1D model), so the pictures can be more perfectly matched.

Now we focus on the matching of two images. This is first done by forming a kd-tree of the features in one image (image A). We used FLANN, Fast Library for Approximate Nearest Neighbors [ref. 2], to construct the kd-tree. Now we search for each feature X in the other image (image B) that matches any features in image A. This is done by finding the two nearest neighbors in all features of image A. If the distance to the closest feature (feature Y) < 0.6 * the distance to the second closest feature, then we say feature X in image B matches with feature Y in image A. The matches are illustrated in the following images.


The matched features in two images

After we have found the matches, we apply RANSAC [ref. 3] to robustly find the displacement between the two images (since we are considering translational model). This is done in 300 iterations. For each iteration, we randomly select 5 matches and find the mean displacement (translational model M) of the 5 matches. Then we count the number of inliers under model M, where inlier is defined to be the matches that are displaced by no more than 5 pixels under the translational model M. Among the 300 iterations, we used the model M that result in the highest number of of inliers.

With the relative displacement between any two successive images at hand, we then crop each images to form a list of images that has the same number of columns and rows, but displace only in horizontal direction (a simplified version of the original case). This makes the blending step much easier and will always result in a rectangular panorama. This step is illustrated in the following figure. The original image is simplified to have size 4x7. The thick black line shows the

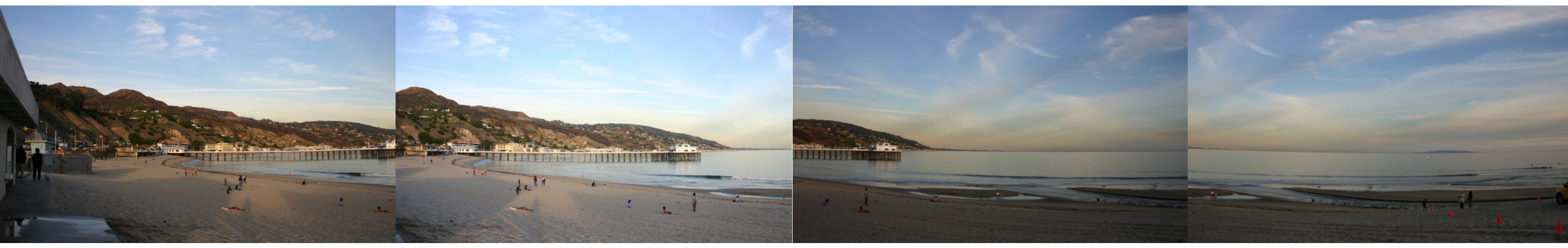


The cropping of the list of images

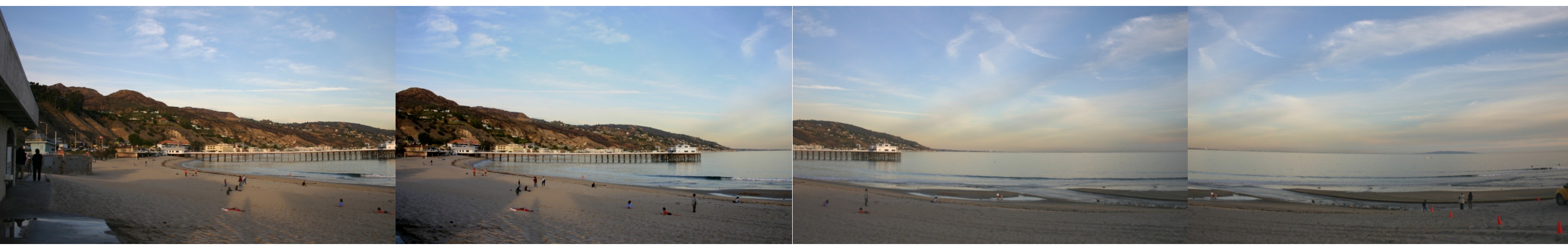
boundary to be cropped. Note that due to floating point displacement, we have to use bilinear interpolation for the cropping of  each image. After cropping, we will result in 4 images each of size 3x7. And is stored in the `<Temporary_Folder>` in Section I. The horizontal shifting is stored in the metadata file named “`metadata.txt`” inside the same folder. The final panorama will be formed in Section V using various blending strategy.

## V. Blending:

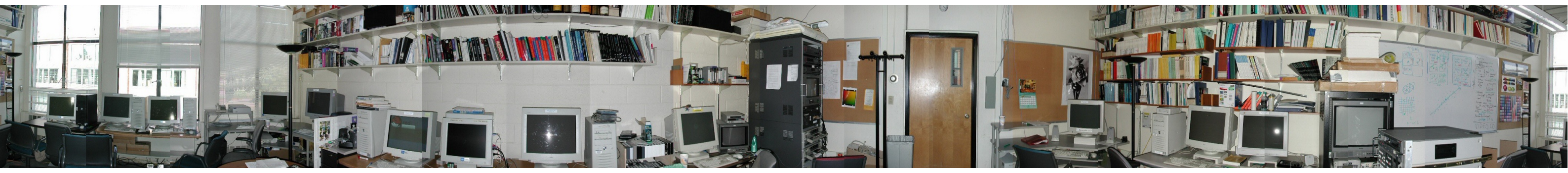
We first conduct a global brightness equalization. This is crucial as pictures are often taken under different exposure or the scene simply has a wide dynamic range. This is done by considering the overlapping area of consecutive images. I only focus on the boundary of the overlapping area, because there may be many ghost lying inside the picture. I then compute the mean of the intensity shift for each pixel. The mean intensity shift is then used to shift the global brightness of the following image (so the first image remains the same). After this process, the brightness for each image will become roughly the same. This can be seen from the following experiments. The pictures are taken long time ago in USA. Clearly the pictures are more consistent after global brightness equalization. For the second set of images in the next page (the sample image: grail),

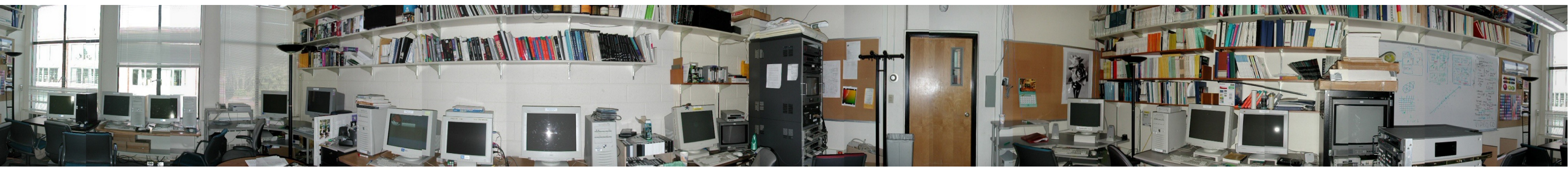


Before global brightness equalization



After global brightness equalization

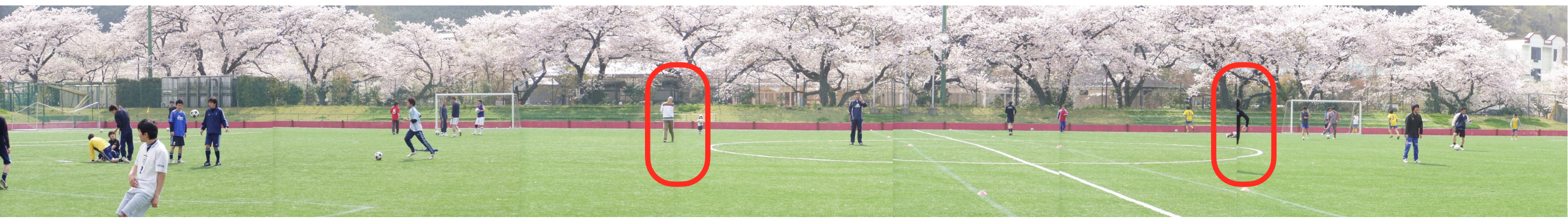

Before global brightness equalization

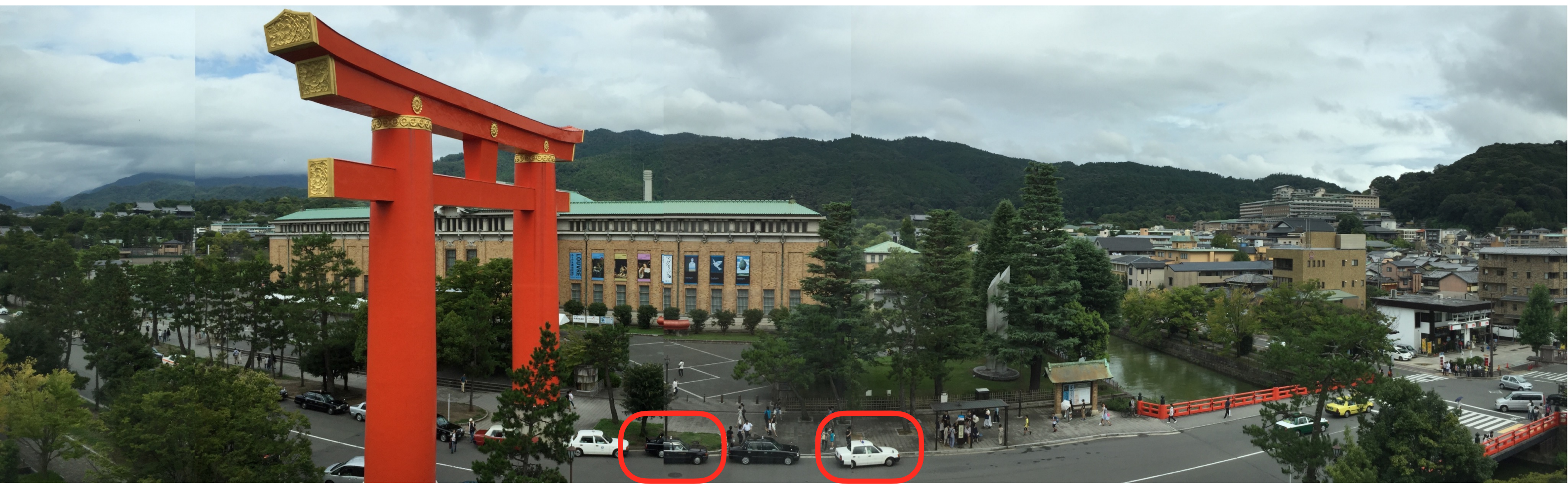

After global brightness equalization

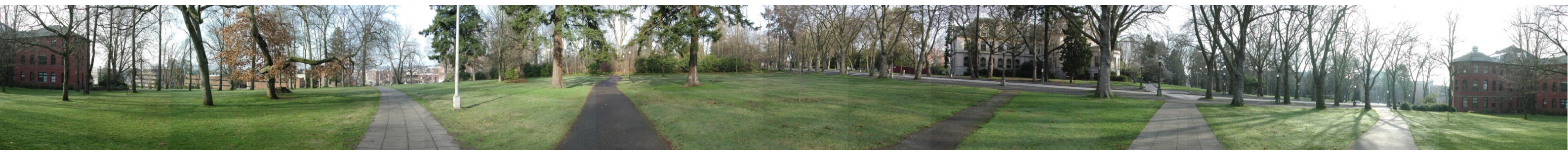
where direct blending is applied (explained later), the exposure difference on the wall is less visible. However since direct blending is applied, there exist lines that are not natural.

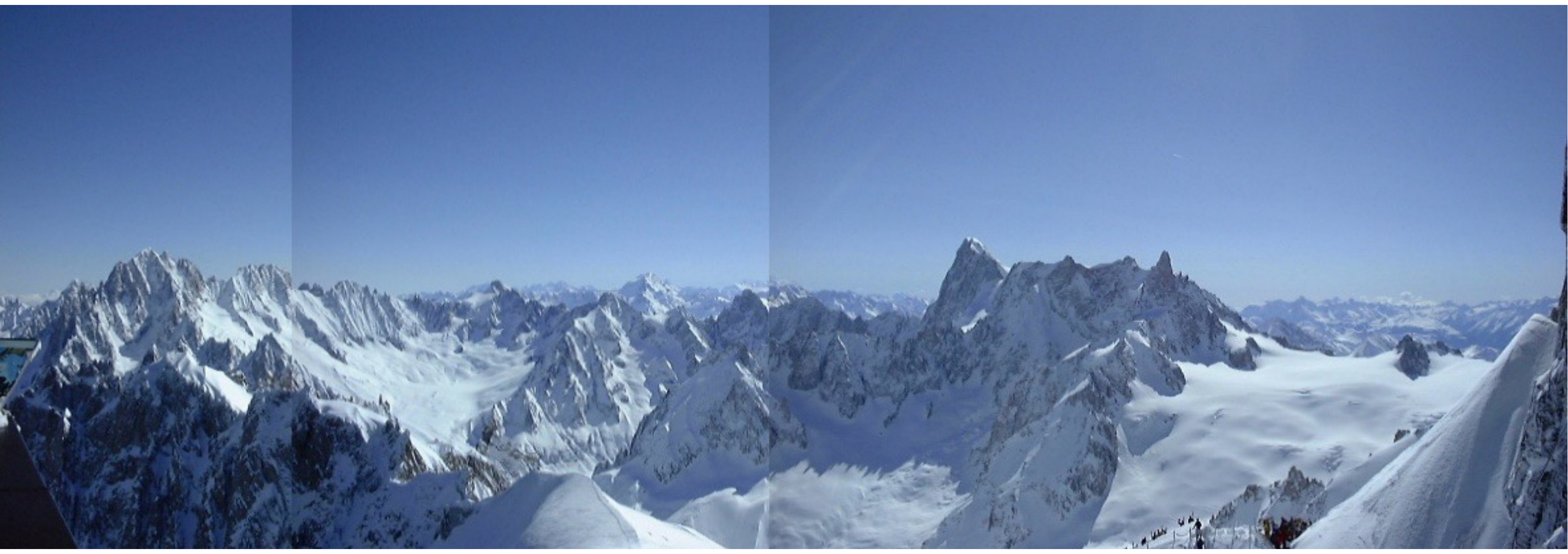
After preprocessing the images, we then consider blending the different images. We have implemented various blending strategy, and will later compare their results.

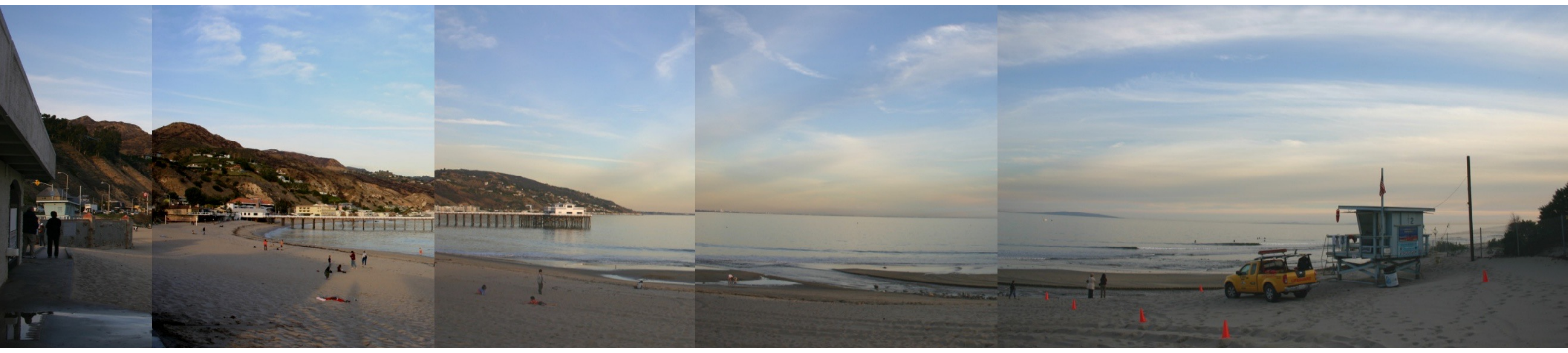
1. Direct Blending: This is the simplest form of blending. Each image is simply pasted onto the previous images. Some results are shown below (some of them are taken by me long time ago, while some of them are from this website: http://www.photofit4panorama.com/gallery.html). Basically it creates many visible lines due to the exposure difference and the difficulty to match the images (since the camera may have rotation motion as the pictures may be taken by hand). Also moving people may be accidentally cut out as shown in the red circles.


Result for Direct Blending

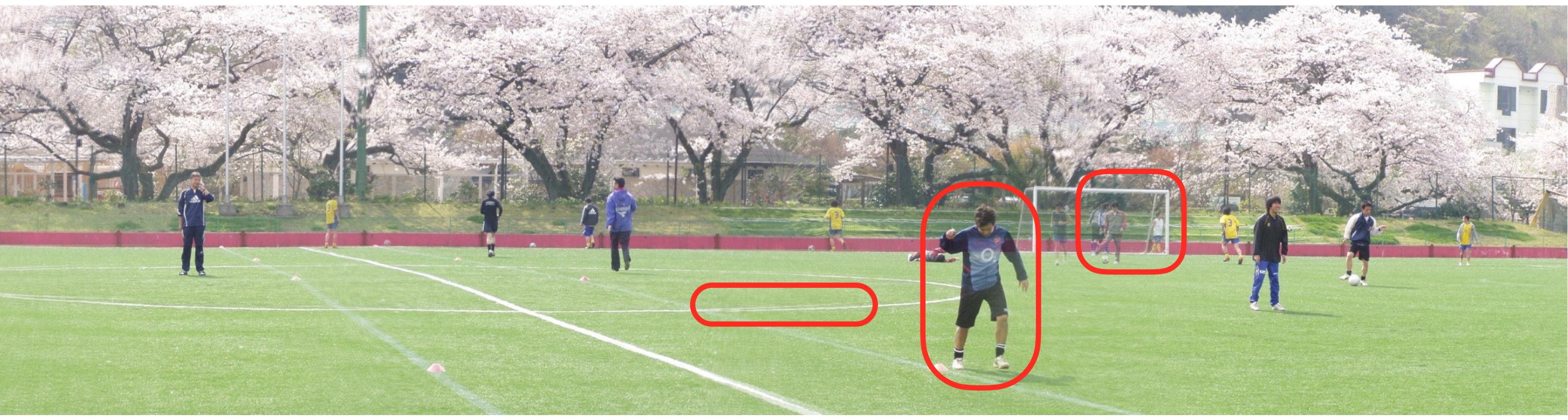

Result for Direct Blending (taken in Kyoto)

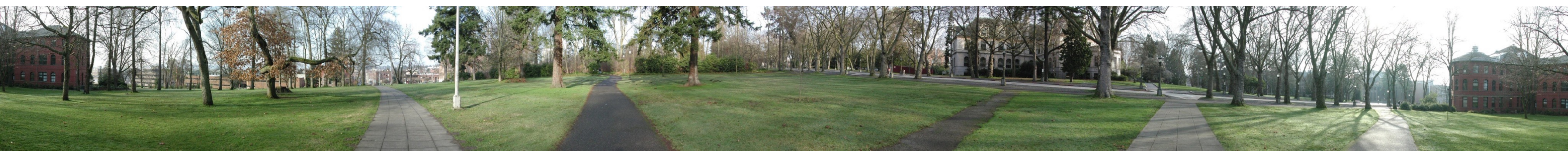

Result for Direct Blending

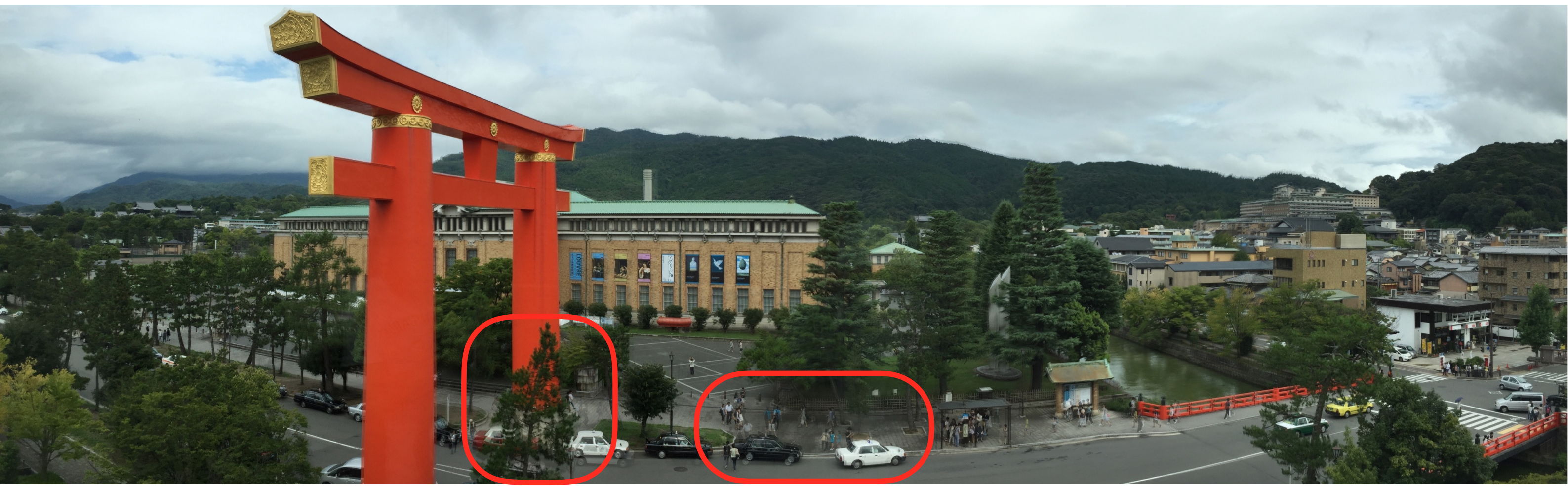

Result for Direct Blending

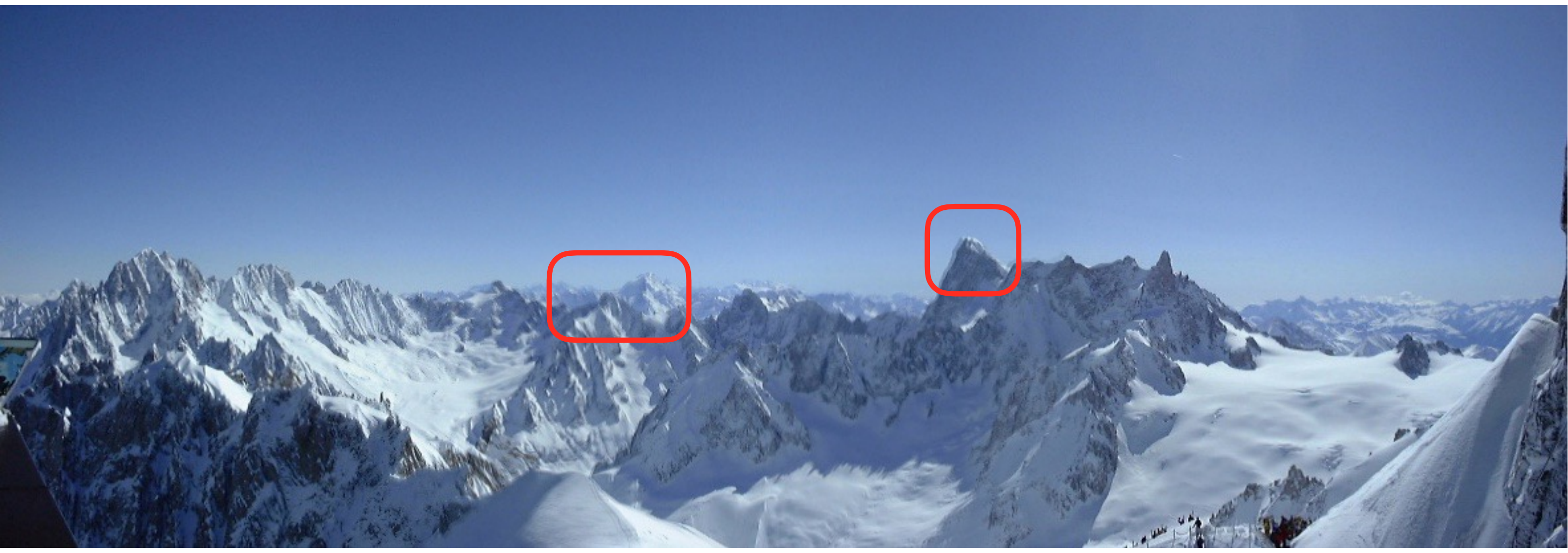

Result for Direct Blending. This one creates a very bad panorama. This may be due to the fact that these pictures are taken by hand a long time ago.

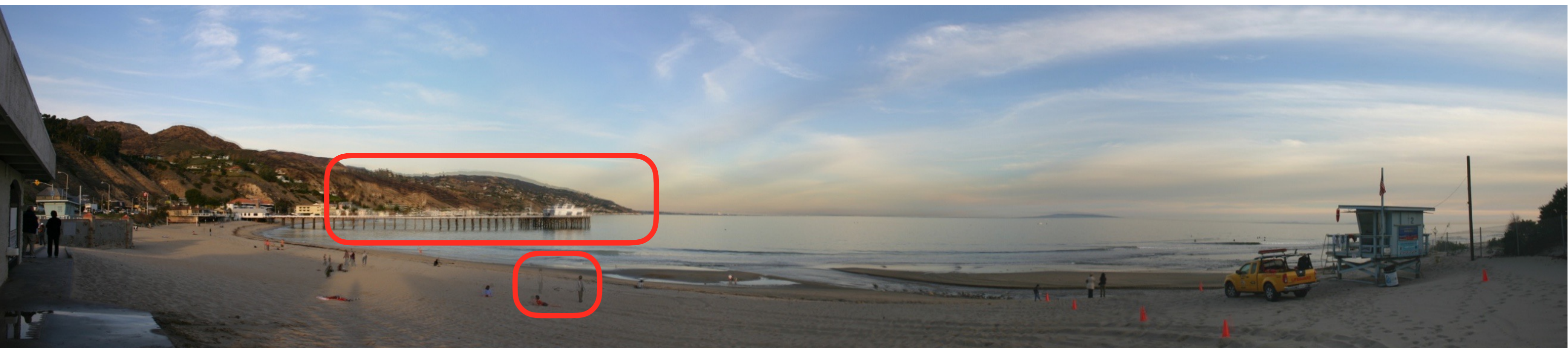
2. Alpha Blending: It is another equally simple approach. The color in the overlapping region uses the weighted average of the two overlapping images. The one that is closer to the pixel will assign a bigger weight. Therefore the color will transition gradually from one image to another without creating the visible seam. Some results for alpha blending are shown below. Although no visible seam is created, the edges may be blurred and the moving people will become transparent as shown in the red circles. But if the picture is taken carefully using tripod and the scene contains no moving objects, the result for alpha blending can be pretty good, e.g. in Parrington shown below.


Result for Alpha Blending: Only the right part is shown.


Result for Alpha Blending (Parrington)


Result for Alpha Blending (Kyoto)


Result for Alpha Blending


Result for Alpha Blending

3. Graph-cut based blending: This approach is motivated by [ref. 3]. Their idea is to create a seam line that cut through the overlapping region. This cut will dissect the overlapping region into two subregion, one to the left and one to the right. Each one will use its corresponding image. The first method can be seen as a special case of this graph cut method, where the cut is a vertical line. But to make better panorama, graph cut is used to minimize the cost of the cut, which can be for example the difference between the boundary color. However, applying graph cut can be very time consuming. Thus we have think of a greedy approach to solve this problem in O(nlogn), where n is the number of pixels in the overlapping region.

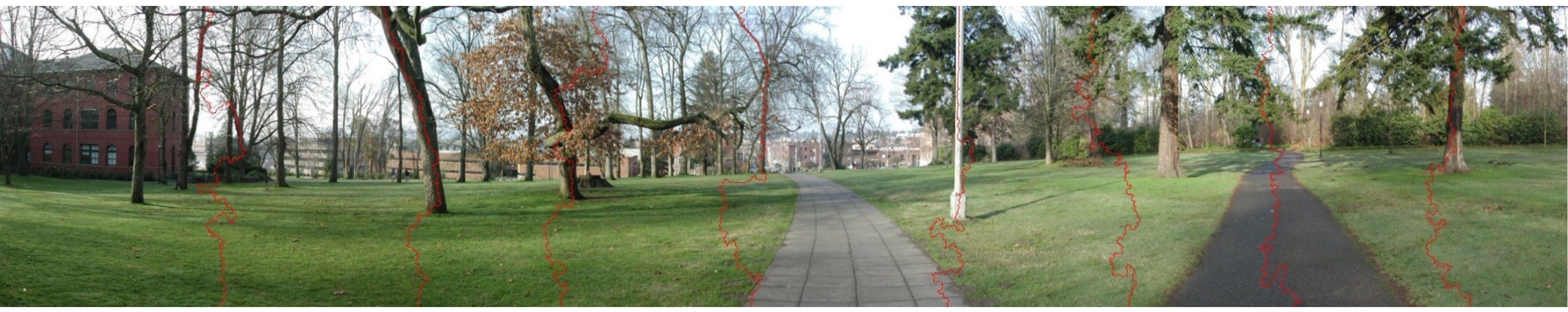
   We first set the boundary of the image equal to the corresponding image (left side uses the left image, right side uses the right image) as shown in the leftmost figure (the yellow image corresponds to the overlapping region). Then we grow the labeled pixels into the overlapping region. This is done by first growing into the pixel that has the largest maximum difference in the RGB channels. We used priority_queue to find this pixel in O(logn) time. After every pixel has a label, the procedure stops, as shown in the rightmost figure. The final boundary will have a small maximum difference. Although this may not be the optimal seam, the result is actually surprisingly good. We will demonstrate this in the following experiments.

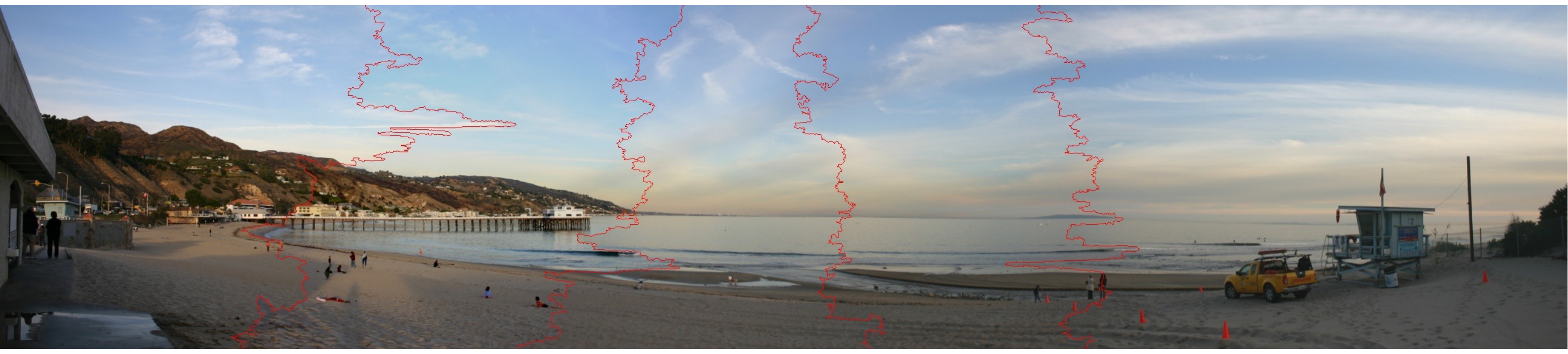
| | | | | | | |
|---|---|---|---|---|---|---|
| -1 | | | | | | 1 |
| -1 | | | | | | 1 |
| -1 | | | | | | 1 |
| -1 | | | | | | 1 |

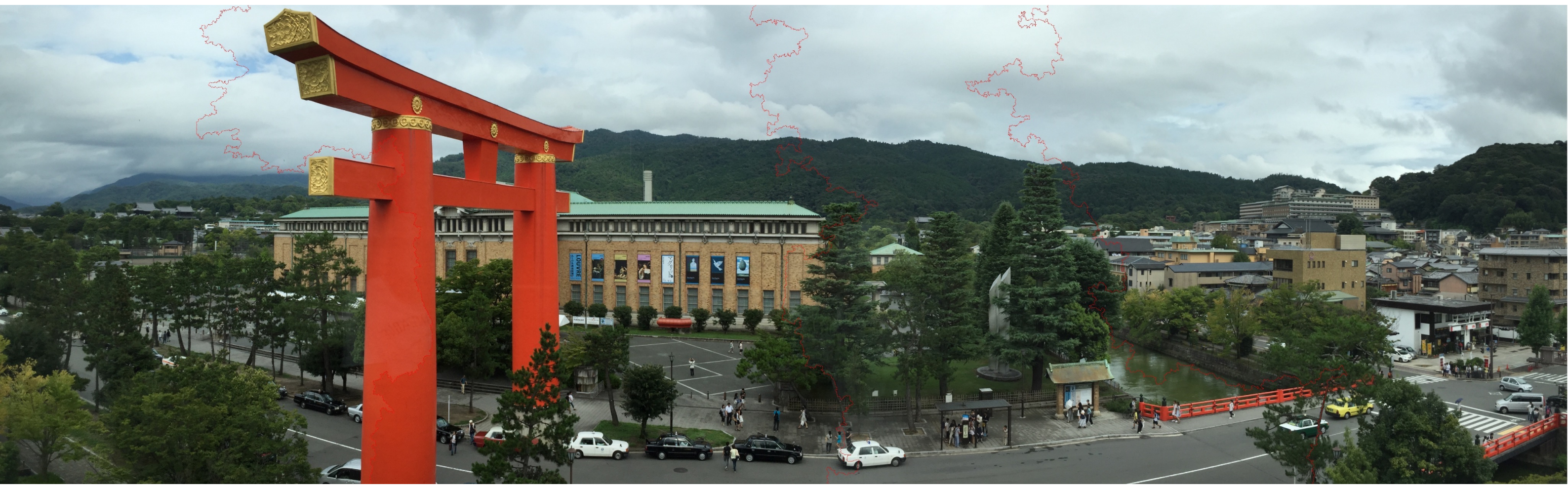
| | | | | | | |
|---|---|---|---|---|---|---|
| -1 | | | | | | 1 |
| -1 | -1 | -1 | -1 | | 1 | 1 |
| -1 | | -1 | | 1 | 1 | 1 |
| -1 | | | | 1 | | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| -1 | -1 | 1 | 1 | 1 | 1 | 1 |
| -1 | -1 | -1 | -1 | 1 | 1 | 1 |
| -1 | -1 | -1 | 1 | 1 | 1 | 1 |
| -1 | 1 | 1 | 1 | 1 | 1 | 1 |

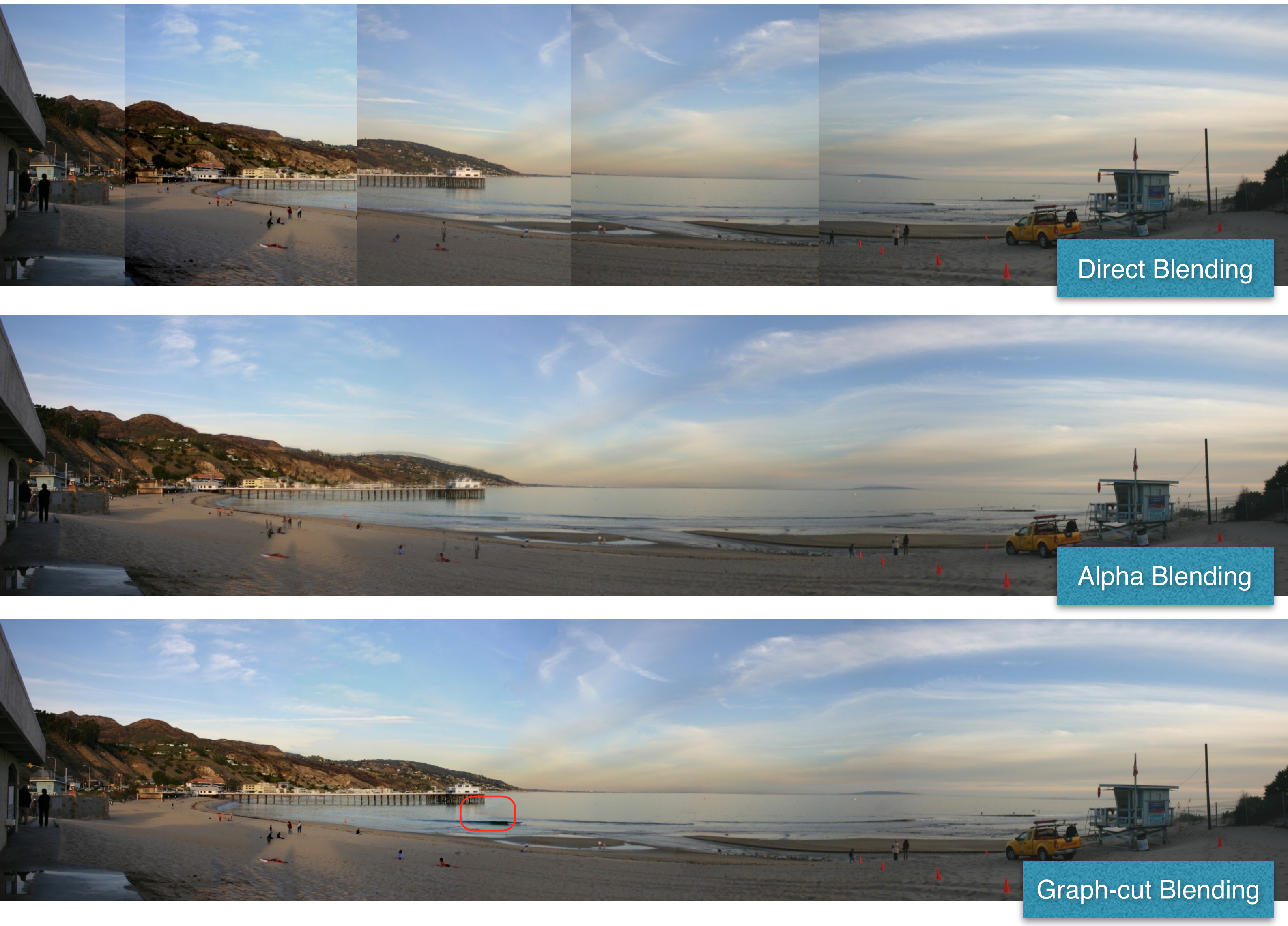
First of all, the following panoramas show the found cut in red lines. You can see that the curve can turn around sharply finding for a good partition that result in nearly invisible seams.


Result for the cut found

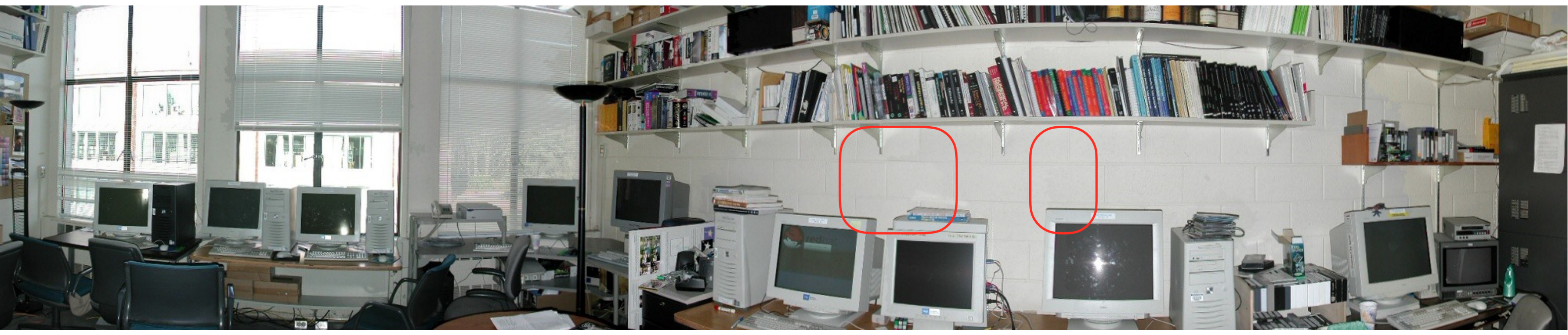

Result for the cut found

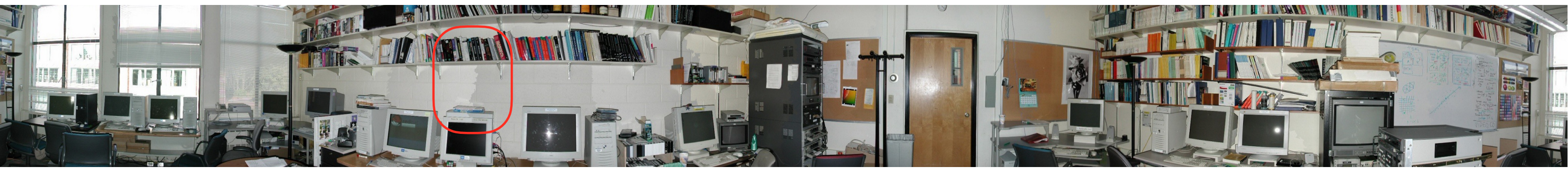

Result for the cut found

We then compare the performance of graph-cut motivated blending with the above two blending techniques using the beach panorama taken several years ago. From this comparison you can see that the benefit of graph-cut based blending is that (1) it do not blur the edges and does not produce ghost as in alpha blending. The first property also holds for direct blending, but (2) it does not contain visible seams that may pierce through walking people as in direct blending; and (3) even though the image is not perfectly stitched (due to non-translational camera motion), it still creates an astonishing panorama. However, by carefully looking at the image, you can see that there still exists seams that are visible (shown in the red circle).



We present several other panoramas created using this blending method. For the grail panorama, you can see that there exists some slightly visible seams. The result if we did not apply global brightness equalization is also shown. As expected, the seams become much more visible when global brightness equalization is not used.

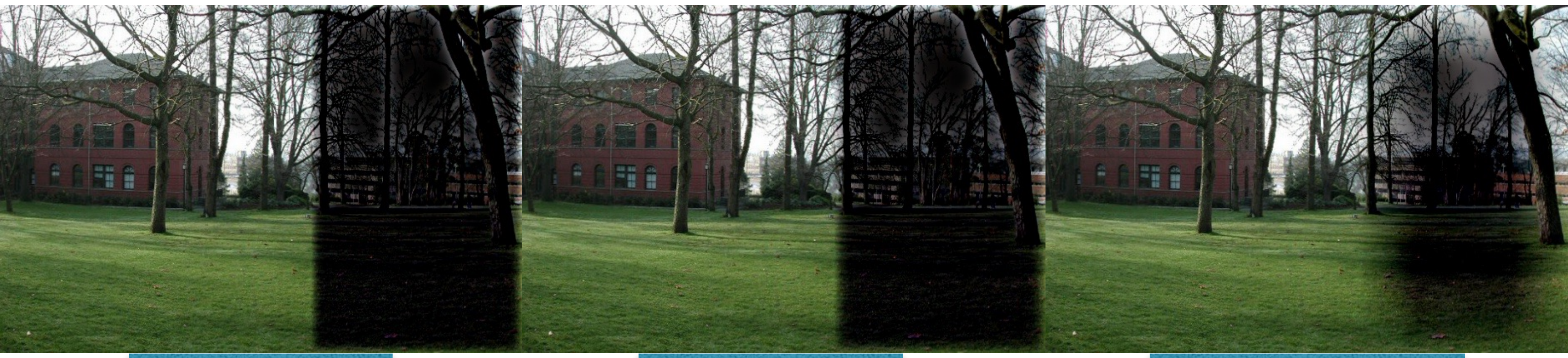

Result for graph-cut motivated blending (right half of grail)


Result for graph-cut motivated blending (grail without brightness normalization)

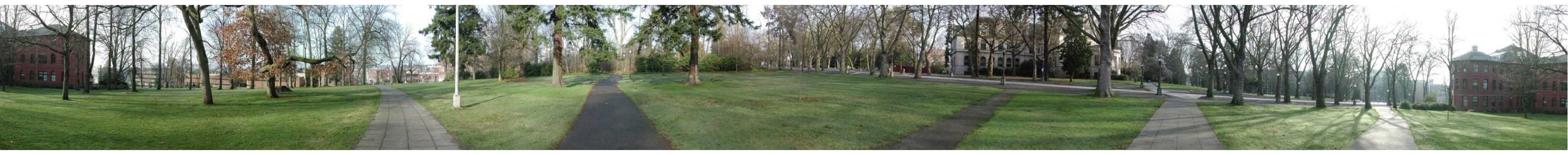
Result for graph-cut motivated blending

4. Poisson blending: We have also implemented Poisson blending based on [ref. 4]. While graph-cut based blending will not pierce through the image, it can still leaves visible seam. The reason is due to the exposure difference between the two consecutive image. This is the problem Poisson blending wanted to solve, the exposure difference between different images. The idea is to cut off the region we want to paste our second image B. For that region the gradient of image B is used, but the color value on the boundary uses that of the original image A. Now we want to assign the color value for the pixels inside the cut off region. This is done by minimizing the absolute difference between the pasted gradient field and the gradient of the assigned color value. By a simple mathematic technique often used in Electrodynamic, it is equivalent to letting the laplacian of the assigned color value to equal the divergence of the pasted gradient field. This can turns easily into a linear system Ax = b. However, the difficult part is that the size of A is actually of (# of pixels) * (# of pixels). Thus some numerical method has to be applied to solve this problem.

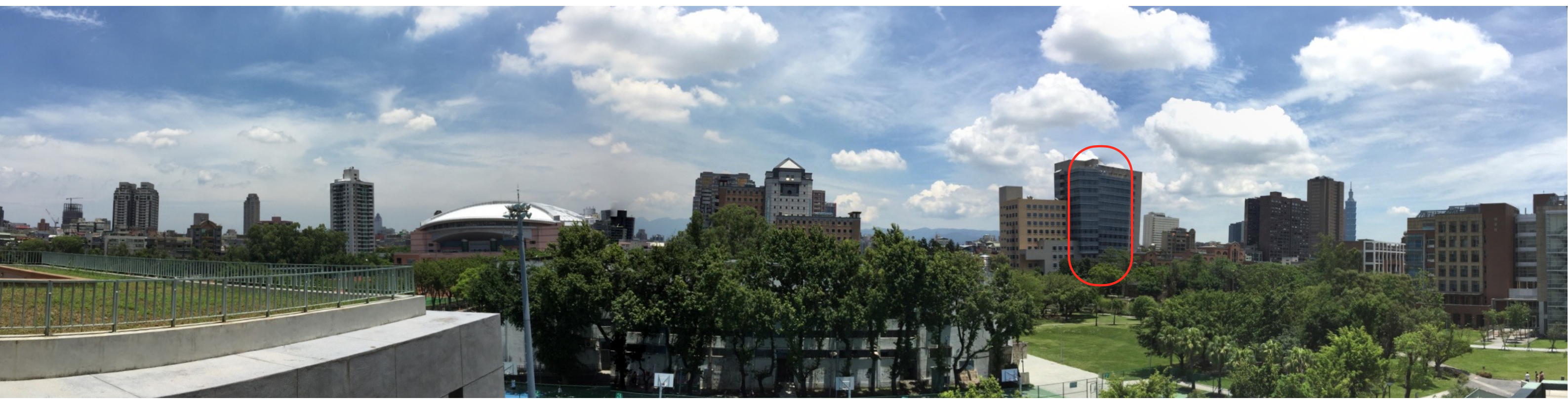
   The good thing about A is that it is very sparse. Thus it can be solve by iterative methods. We have implemented three numerical methods (*completely written by ourselves* without resorting on numerical libraries). We have first written Jacobi method. However the convergence rate is very slow. It would take about (# of pixels) iterations to converge to the optimal solution. Thus we then consider using Gaussian-Seidal method, this is twice faster than Jacobi method, but is still too time consuming. So we have written a much faster version using Conjugate Gradient Descent. (all of the code for these numerical methods to solve Ax=b are in


Jacobi Method Gaussian-Seidal Conjugate Gradient

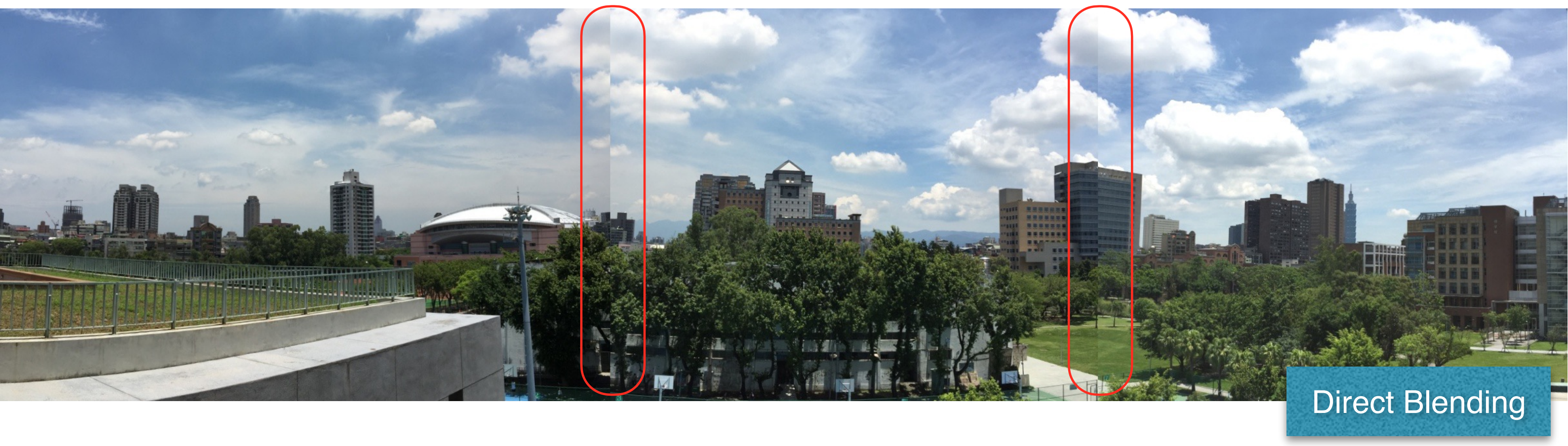
`blending_poisson.cpp`) The pictures in the previous page shows the result after 100 iterations for these three methods. Basically Gaussian-Seidal is only slightly better than Jacobi, where Conjugate Gradient is significantly better than both of them. But 100 iterations is not enough for convergence, basically 500+ iterations is needed for conjugate gradient. We have also create a video, showing the optimization process using CG, named ***poisson_ntu.mp4***. Below images shows the final panorama for a few set of pictures. The running time using CG is still too slow, so we did not show the result running on large images. We then compare the 4 blending techniques on the resulting panorama for ntu (taken by hand).

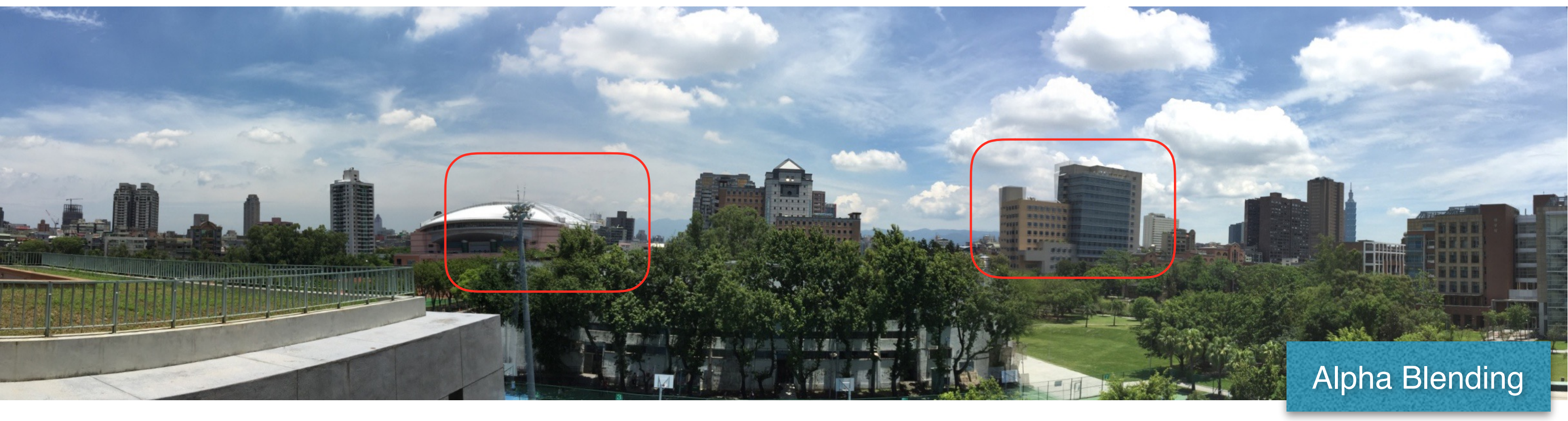


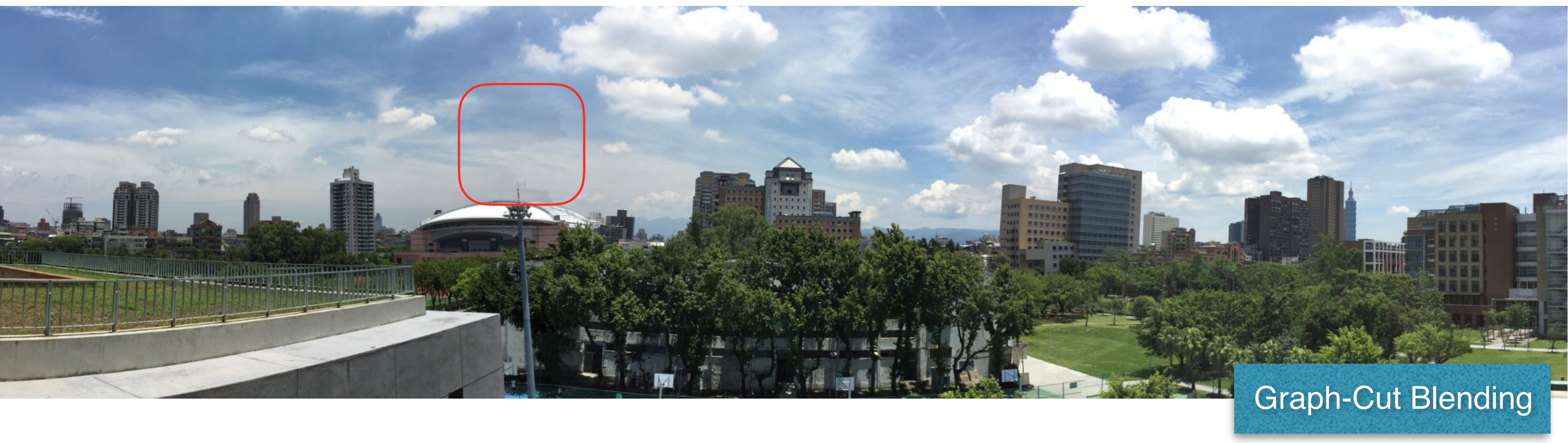
Result for Poisson blending



Result for Poisson blending (ntu)

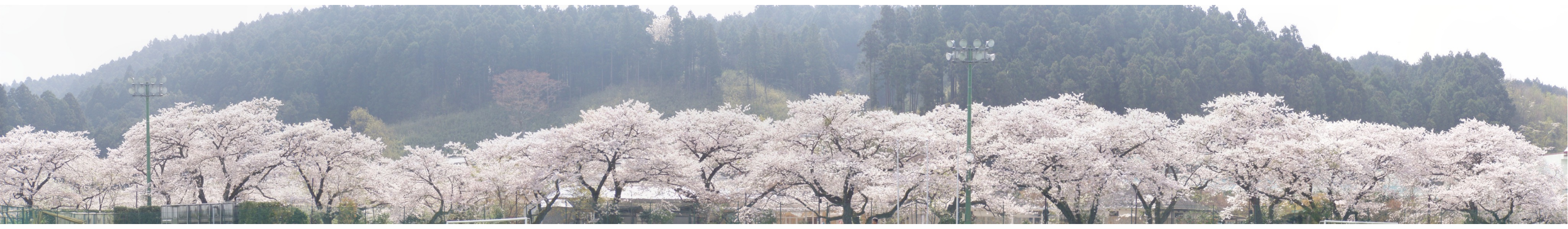


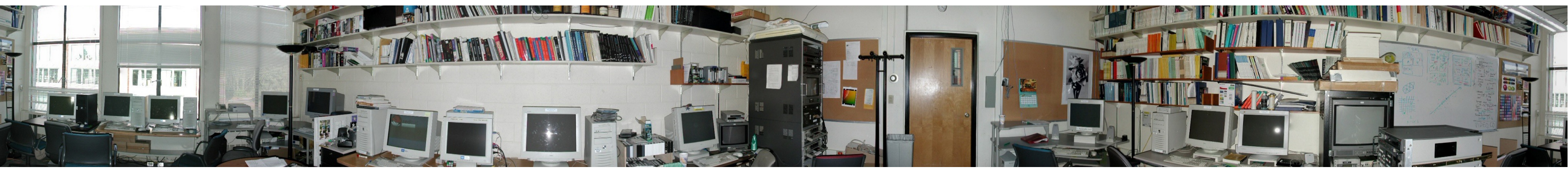


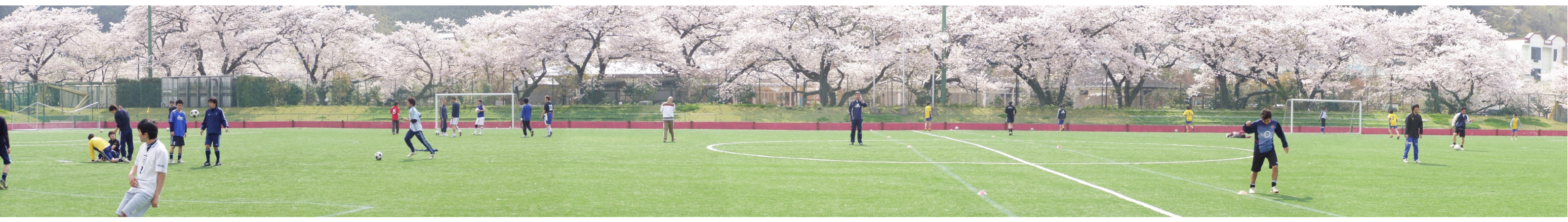


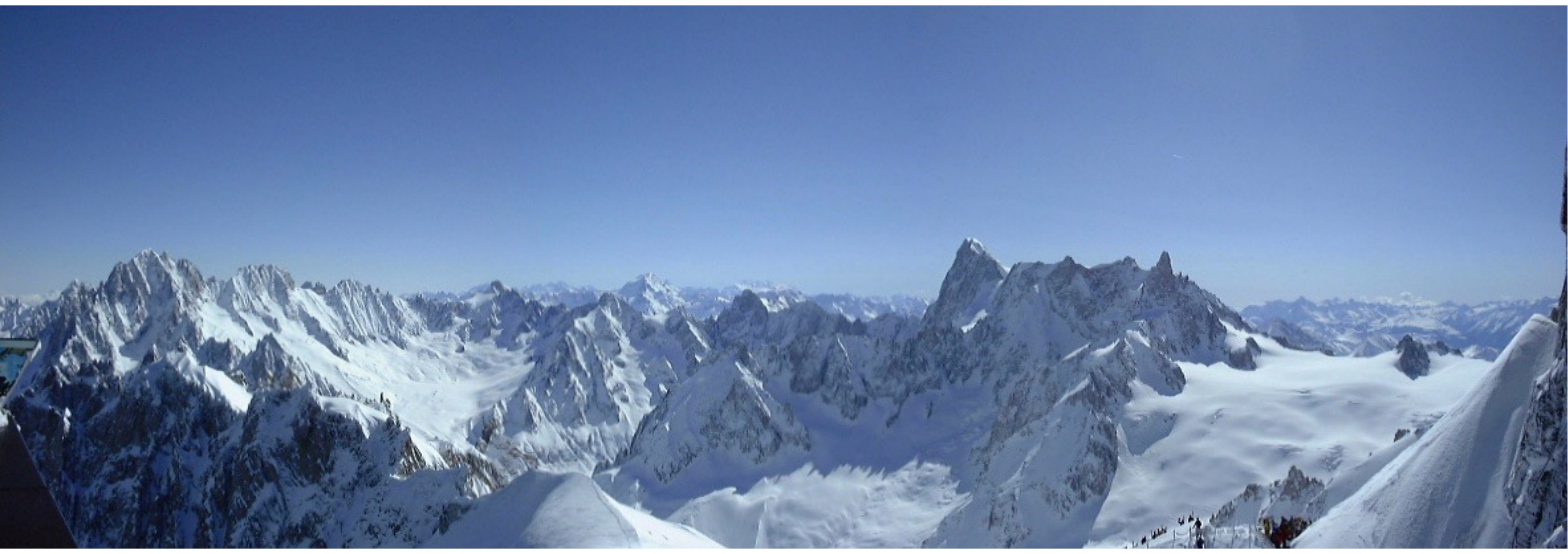
Basically, alpha blending will blur the edges, which is not desirable. Other three blending creates a sharp image by not performing weighted sum between different images. However direct blending creates visible seam line, because of exposure difference and the seam line may pierce through moving building due to camera motion. On the other hand, graph-cut blending will not pierce through moving building, but can create slightly visible seam line also due to exposure difference. And for Poisson blending, it fixed the problem of exposure difference, so no visible seam line is created, however it may create broken building because the invisible seam line may pierce through the building (or walking people) on the boundary. So basically they all have it's own advantages and disadvantages.

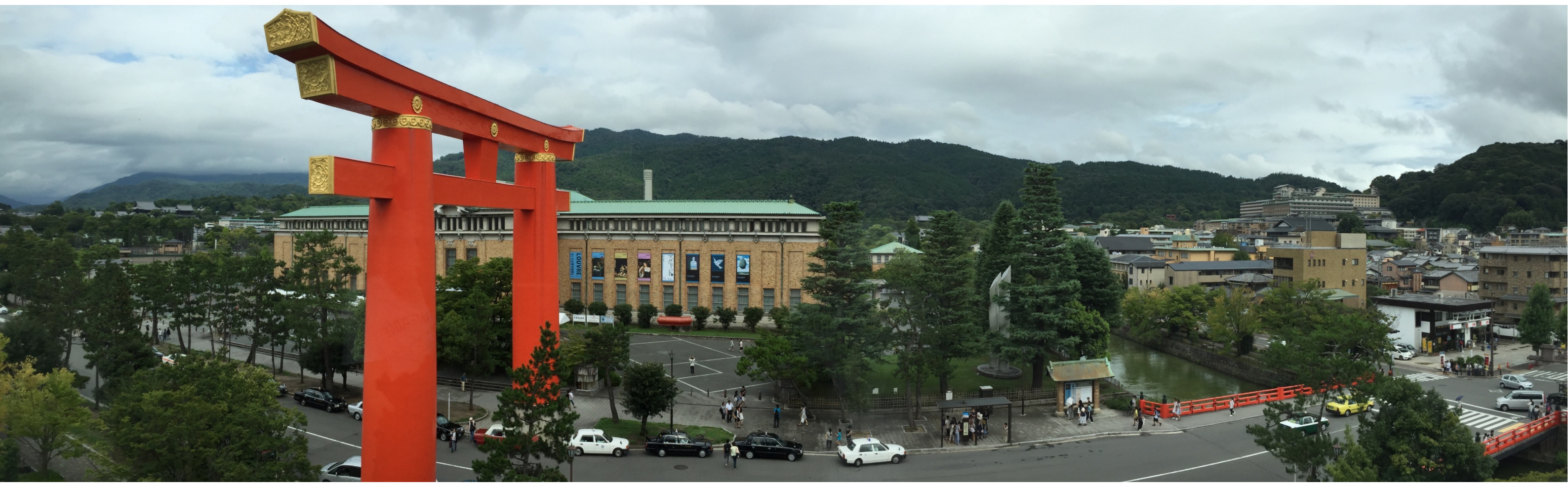
5. Photomontage-motivated blending: Although the above four blending techniques all have their pros and cons, it is possible to create an ultimate blending technique that resolve all the aforementioned issues. Our work here is motivated by [ref. 6]. The idea is to combine graph-cut blending and Poisson blending, so it will create invisible seam line that do not pierce through moving objects. We did this by also Poisson blending the two images. But instead of using a vertical boundary, we used the boundary chosen using the graph-cut method. However, previously we only implemented Poisson solver with rectangular region, but now we have to handle arbitrary boundary. Therefore we have extended our original Poisson blending solver to handle this situation. And we have discover that it is possible to speed up the optimization by using a good initial solution, which is by using the original picture inside the selected region. This creates a *significant speed up*. Several created panoramas are shown in the following images. The result can be seen to be quite good.

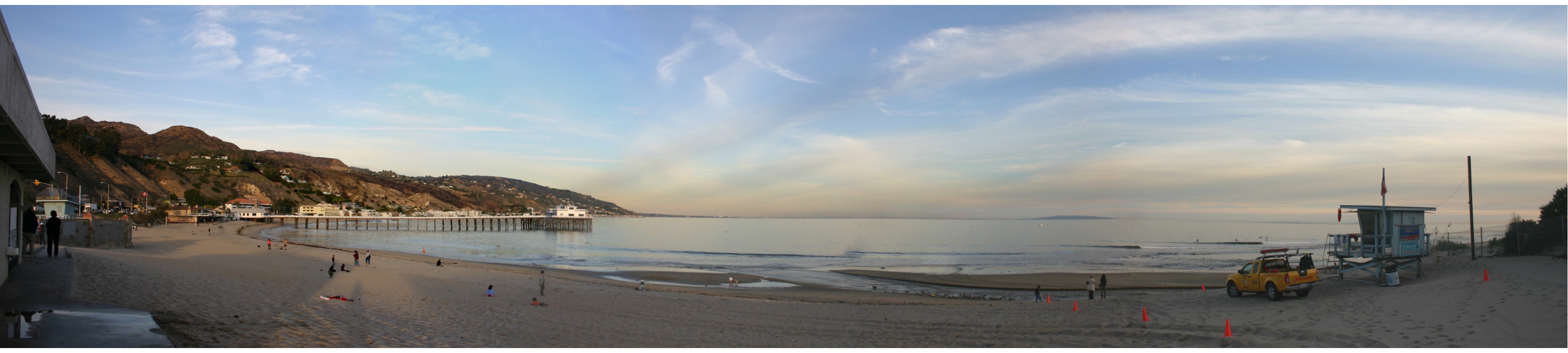

Result for Photomontage-motivated blending

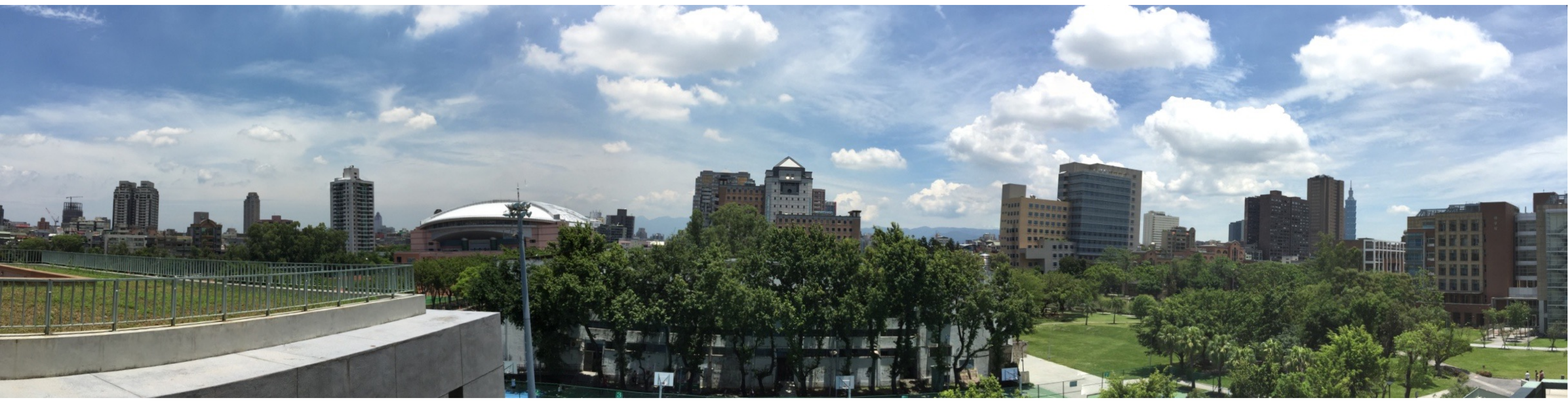

Result for Photomontage-motivated blending

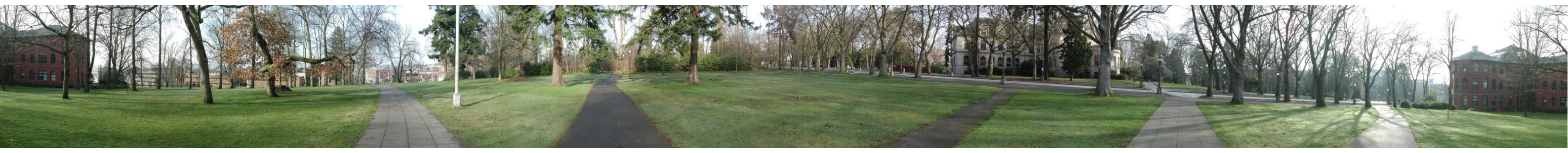

Result for Photomontage-motivated blending


Result for Photomontage-motivated blending


Result for Photomontage-motivated blending

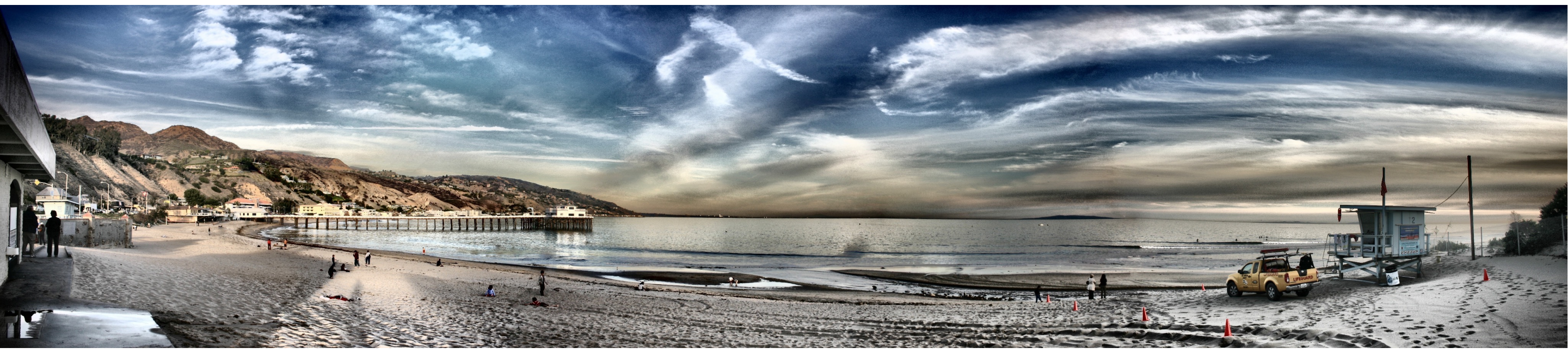

Result for Photomontage-motivated blending


Result for Photomontage-motivated blending



Result for Photomontage-motivated blending

After blending the images, we perform contrast limited adaptive histogram equalization (CLAHE) supported by openCV to create an artistic-looking panorama (CLAHE is only applied in `blending_photomontage.cpp`). Our final artifact is the following image.



Final Artifacts

## VI. Reference:

1. Multi-Image Matching using Multi-Scale Oriented Patches, by Brown et al., 2005.
2. FLANN - Fast Library for Approximate Nearest Neighbors, by Muja et al., 2009. (http://www.cs.ubc.ca/research/flann/)
3. Random Sample Consensus: A Paradigm for Model Fitting with Applications Analysis and Automated Cartography, by Fischler et al., 1981.
4. Poisson Image Editing, by Pérez et al., 2003.
5. Fast Approximate Energy Minimization via Graph Cuts, by Boykov et al., 2001.
6. Interactive Digital Photomontage, by Agarwala et al., 2004.