David Lariviere
Rendering Final Project
Professor Belhumeur
Spring 2006

Panorama Generation from Multiple Views of a Single Viewpoint

Introduction:

My project was to create a program capable of merging multiple pictures taken from a camera being rotated on a tripod and create a new panoramic image which combined multiple images which were properly perspective-warped. This has the effect of generating images with a field of view that was larger than that of the original camera.

Theory:

In order to combine multiple images which differ only by a rotation about the optical axis, it is first necessary to establish a base reference frame which will represent the image plane that is going to be simulated by merging multiple images.

Given two image planes, I_1 and I_2, choose the viewpoint of I_1 as the base image frame. It is now necessary to calculate the homography, H, which maps every point $p_2(x_2, y_2)$ in I_2 to its corresponding point $p_1(x_1,y_1)$ in I_1. Mathematically, H is defined such that $Hp_2 = p_1$ [1]. The homography, in effect, maps the point where the same light ray intersecting I_2 also intersects I_1. (See figure 1).

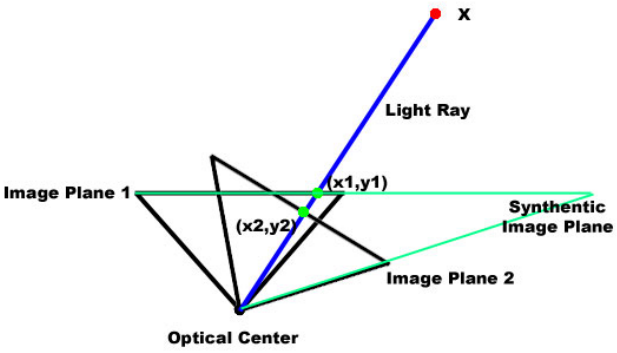


Figure 1

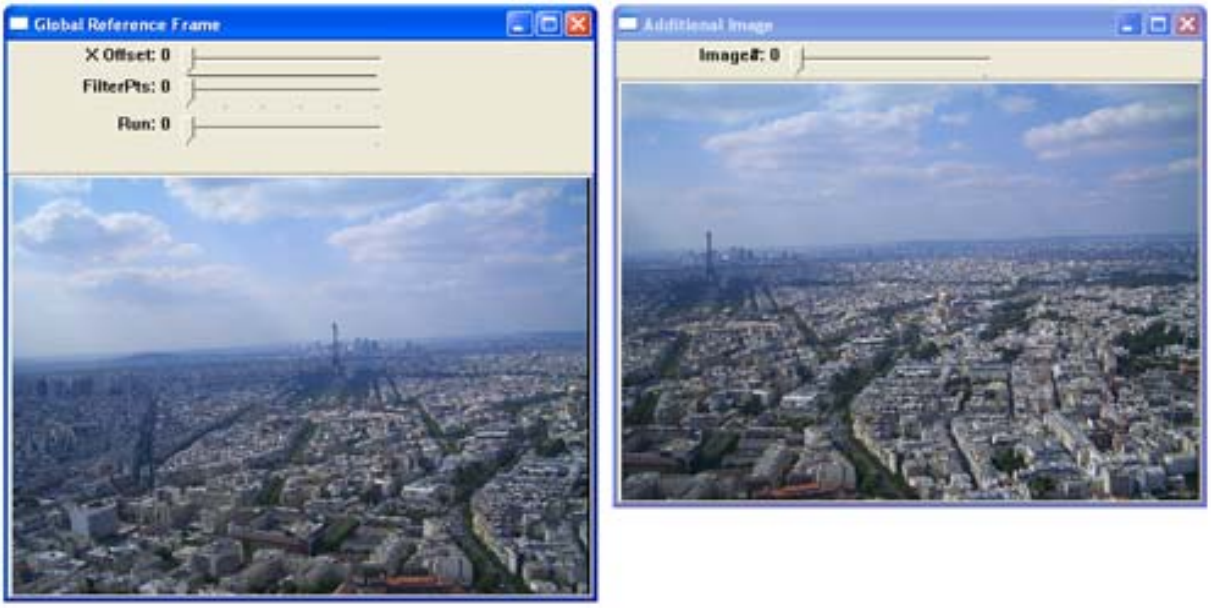
The homography is a 3x3 matrix with 8 degrees of freedom, since the correspondence can only be defined up to scale. In order to calculate the homography, at least 4 point correspondences between I_1 and I_2 need to be established [1]. Because of noise and error in the point mapping, however, many points are used in calculating the best homography which minimizes some error metric. In this project, the lowest average reprojective error per correspondence point is used.

Implementation:

Four main goals were targeted when designing the program.

1. Point Correspondences, when possible, will be established automatically or with as little user input as possible.
2. The user will be allowed to manually specify point correspondences which are of the same accuracy as the automatically established points.
3. The running time of the algorithms will be short (no more than a few seconds).
4. Incorrect point correspondences will be removed or the effects of their errors mitigated.

Figure 2 displays the GUI of the program. It consists of two windows: on the left is the window holding the global reference frame which all other frames will be mapped into, and the window on the right which contains the additional image which is to be mapped into the global image.



(Figure 2) Global Reference Frame (Left), and Additional Image Frame (Right)

High Level Program Usage:

1. **Offset Estimation:** The user slides the "X Offset" slidebar to indicate approximately where in the Global Reference Frame the left-most edge of the Additional Image is. A vertical black bar is drawn onto the Global Reference frame to aid the user in positioning the offset.
2. **Point Correspondence**:
    a. The user invokes the automatic point correspondence algorithm by setting the "Run" slider to 1. All of the corresponding points between the two images will then be drawn on the images.
    b. If many of the point correspondences are incorrect, the user can utilize the FilterPts slidebar to invoke a statistically based filtering algorithm which

minimizes the standard deviation of point correspondence's disparity. The number on the "FilterPts" slidebar indicates the number of iterations that the filtering algorithm will run.

c. The user has the ability to manually remove obviously noisy points by clicking on a point in either window in order to remove both the point, and the corresponding point in the other window.
d. The user can also add additional point correspondences manually by first selecting a point in either window and then selecting the corresponding point in the other window.

3. **Generate Panorama:** Once point correspondences have been properly established, the user slides the "Run" slidebar to 1 in order to invoke the warping algorithm which will calculate the best homography and lastly generate a new image with a field of view of both images combined.

* *Note: The use of sliders in invoking commands was used only because it is the only type of GUI control provided in OpenCV's simple window GUI for displaying images.*

Implementation Details:

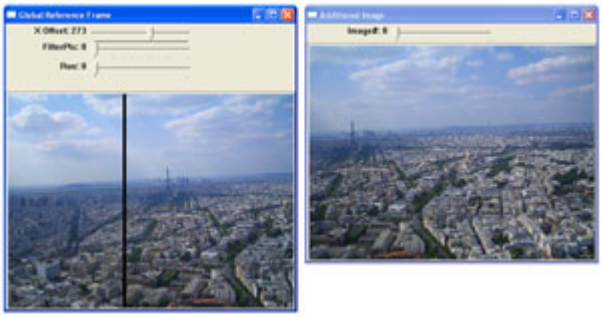
1. Offset Estimation:

Originally, attempts were made to more or less completely automate the warping process, such that the program was given two images and assumed only that the second image was taken after being rotated to the right and that the overlap between the two images was at worst only 50% (meaning that given two 600 pixel wide images, it was assumed that the left most points in the second image (0,y), were to be found somewhere between (0,y) and (300,y) in the first image). For some images, this worked quite well. However, on more difficult images which were taken many years ago and had an overlap of less than 50% and thus required a larger window (300-500 pixel deviation between the two images), it often resulted in too many erroneous point correspondences.

A second attempt was made to calculate corresponding points iteratively. First, a set of point correspondences was established using only the assumptions above. Then statistical filtering was used to hopefully eliminate outliers (to be discussed in section 2). Lastly the average disparity was used as the "X offset" and the algorithm was run again, this time with a smaller search window taken from the average of the previously calculated corresponding points. Unfortunately, with the initial window having to be too large, the filtering algorithm could not calculate a useful average disparity to use as the new search window, so this method was abandoned.

A third attempt at automatically calculating the rough disparity was attempted by simply taking a very large portion of the Global Reference Image as a template, and attempting to find the point in the second image which had the best correlation with the large template. Unfortunately, this method also did not generate sufficiently accurate results, since particular images (especially high-altitude photos of cities) contained too much noise and suffered from large perspective distortions between images.

Therefore, in the end, it was decided to let the user manually indicate the offset by using a slider (see Figure 3 to the right, where the vertical black line represents the offset). The search window for corresponding points is then derived by searching a window +/- 100 the X offset specified. Vertical disparity was assumed to be negligible (no more than +/- 60 pixels between successive images). For example, if the offset between the first and second image is set as 300, then a particular point in image 1 at (500, 300) will be matched against all points within the rectangle defined by (100,240) and (300,360).



By allowing the user to manually specify the offset, both the speed and accuracy of the automatic point correspondences improved dramatically.

2. Point Correspondence

2a. Automatic Point Correspondence:

Corners are good features to track between successive views of the same scene. This is because they are, in general, invariant to projective warpings and can be calculated to subpixel accuracy [1].

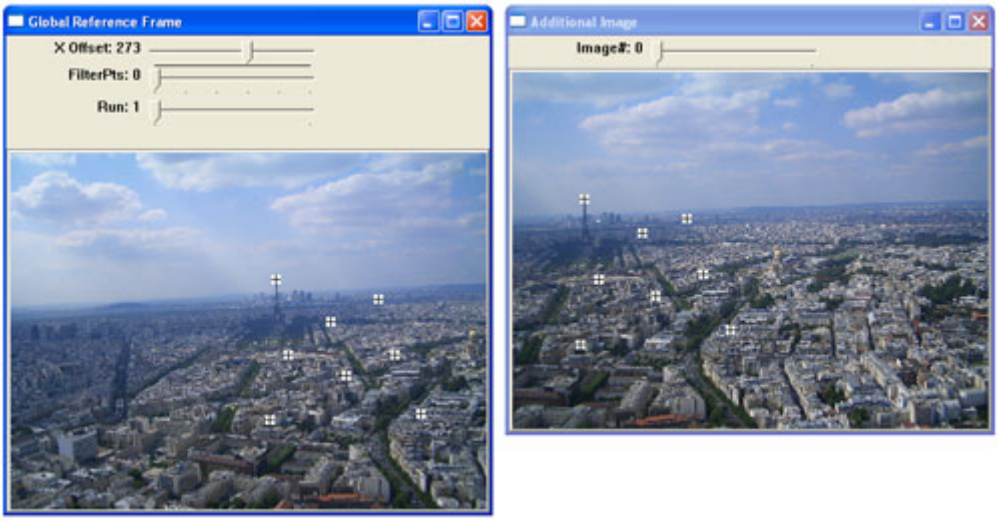
In calculating corners, first a sub-image is extracted from Global Reference Frame, taking into account the X Offset previously described in section 1. For example, given an image of dimensions 640 x 480 and a user-set X offset of 300, a sub-image from the rectangle defined by (200,0) and (640,480) is extracted. Remember that the X Offset is assumed to be within +/- 100 pixels of the actual value. Therefore the first possible point in the first image that can actually be found in the second image would be @ (200,0). This is another example of where the user-set estimated offset is useful, as it limits the region searched for corners and does not bother detecting corners that cannot possibly be found in the other image.

OpenCV implements two popular corner detecting algorithms, the harris corner detector and the eigenvalue-based corner detector. In this implementation, I utilized the eigenvalue based corner detector to locate corners within the extracted subimage from the Global Reference Frame. Once a sequence of corners at pixel level accuracy was determined, another OpenCV algorithm was used to more accurately locate all of the points to an accuracy of .1 subpixels.

Now that a list of corners within the Global Reference Frame has been established, it is then necessary to calculate these corresponding corners in the Additional Image. A template matching approach is used. A template of size 20x20 is extracted, centered about the desired corner in the Global Reference Frame. Then a search window determined by the X Offset +/- 100 is extracted from the Additional Frame. The point in

the extracted window with the strongest correlation with the template, utilizing the normalized sum of squared difference, is selected as the corresponding point in the Additional Image. In order to quickly filter very poor matches, only corresponding points with a correlation meeting a certain threshold are used. Lastly, a subpixel-accurate corner refinement is again applied to more accurately locate the corresponding corner in the Additional Image.

Once the corresponding points have been found, they are displayed in the two image frames, marked by white rectangles with black crosses on them.



(Figure 4) Automatic Point Correspondences

2b. Filtering Point Correspondences:

A statistically based method is used to filter the corresponding points. First, it is assumed that the disparity between a point in the first image and its corresponding point in the second image is roughly the same across all pairs' corresponding points. Experimentally, this seems to hold true, although it has not been rigorously mathematically derived. Intuitively, my guess is that this only holds true if the image "planes" were cylindrical, rather than planar, but since the field of view is relatively small, the plane is a sufficiently accurate first-order approximation of a cylinder, making the assumption acceptable.

The filtering method works as follows. Given a list of n corresponding points (p_i), the average x and y disparities (x_{ave} & y_{ave}) and the average deviation (x_{dev} and y_{dev}) are calculated, where:

$$x_{ave} = \frac{1}{n}\sum_{i=1}^{i=n} x(p_i) \qquad \text{and} \quad y_{ave} = \frac{1}{n}\sum_{i=1}^{i=n} y(p_i)$$

$$x_{dev} = \frac{1}{n}\sum_{i=1}^{i=n} |\, x(p_i) - x_{ave} \,| \quad \text{and} \quad y_{dev} = \frac{1}{n}\sum_{i=1}^{i=n} |\, y(p_i) - y_{ave} \,|$$

Then a cutoff of acceptable deviations are chosen, such that the x and y disparities must be within a range of the average disparity +/- average deviation. The filtering procedure can be repeated several times to iteratively reduce the deviation in disparity values. Experimentally, this method proved to be very good at filtering corresponding points.

The only problem that arose was how many times to filter. It ended up being highly dependant on the amount of bad correspondences. With a preset number of iterations, depending on the image, sometimes either too many erroneous points would be left or too many points would be filtered out. Numerous schemes were attempted to automate the process of choosing the optimal number of filtering iterations based on the number of points left and their deviations. In the end, however, it was decided to provide the user with a slider to manually invoke the statistical filter method the number of desired times (in a range of [0,5] iterations). This way the user could see the effect of filtering in near real time, and choose the optimal number which gave accurate point correspondences. Further, if the user wanted to choose all point correspondences manually, setting the number of iterations to 5 was a quick way of eliminating virtually all points, as opposed to having to manually click each one to remove them.

2c/d: Manual Removal/Addition of Corresponding Points:

The manual editing of points is fairly straight forward and will therefore not be discussed in much detail. The only thing worth noting is that after the user selects corresponding points, OpenCV's sub-pixel refinement algorithm with a search window of 6x6 is run. This way the user can specify corners that are often as accurate as those found automatically without having to actually click the corresponding points within sub pixel accuracy. The only problem that may arise using this technique is when different subpixel accurate corners within the 6x6 pixel search window are selected. I decided that the benefit of getting subpixel accurate corners outweighed the potential errors that might result, given that so long as most points were correct, the error-handling homography algorithm should still yield better results. Further, if the calculated sub-pixel accurate corners were obviously wrong, the user can always remove the points upon seeing the results.

3. Generate Panorama:

3a. Calculate Homography:

In order to extend the field of view of the Global Reference Frame to include the pixels included in the Additional Image, it is first necessary to calculate the homography mapping pixels in the Additional Image to their proper location within the Global Reference Frame.

OpenCV's camera calibration code contains an implementation of a homography-calculating algorithm ported from Bouguet's widely used Matlab camera calibration toolkit [3]. Given that the algorithm was implemented with clearly defined and very accurate checkerboard corners in mind, it is not surprising that the algorithm does not always handle a sometimes very noisy set of corresponding points very well. Therefore, it

became necessary to implement a RANSAC-inspired algorithm on top of OpenCV's FindHomography function.

RANSAC stands for RANdom Sample Consensus [1]. Applied to calculating homographies, the basic principle is as follows:

1. Select a random sample of 5 pairs of corresponding points and then calculate the homography based just on those 5 pairs.
2. Using the calculated homography from (1), calculate the average reprojection error for all known corresponding points.
3. Repeat the algorithm some number of times, recording the smallest known average reprojection error and the homography matrix that produced it.

Note that only 4 points could have been randomly selected (the minimum number required for calculating homographies), but experimentally it was found that using 5 points produced better results. Also, in the formal version of RANSAC, filtering is used whereby those points with a reprojection error above a certain threshold are discarded. However, given the small number of corresponding points typically had, it was felt that doing this reliably would be extremely difficult, given the relatively large effect erroneous points can have on a small set of correspondences. I did implement an algorithm to keep track throughout all of the RANSAC iterations of each pair's total average reprojection error, but upon examining it, no pairs seemed to stick out significantly more than others, so the idea of filtering out points during RANSAC was abandoned. A more detailed analysis, however, would likely yield a robust method of filtering during RANSAC.

In the implementation of the program, the number of iterations was set to 1000. This seemed to strike a nice balance between speed (only a few seconds on a P4 3GHz machine) and accuracy (getting best average reprojective error between .5 and 2 on many image sets).

3b. Rendering the Panoramic Image:

Once the homography mapping the Additional Image to the Global Reference Frame is known, it is possible to merge the two images in order to generate a panorama combining all of the information visible in either view into a single image.

The first attempt at combining the two images was to first copy the entire Global Reference Frame into the new merged image, and then add the additional points in the Additional Image not found in the Global Reference frame to the new merged image. This had the advantage of being relatively simple. The downside was that if the two images had different exposures or coloration of the same points, then a distinct line is clearly visible (see Figure 5) showing the boundary between the Global Reference Frame and the Additional Image once they are both cast into the new merged image (representing the new Global Reference Frame)..

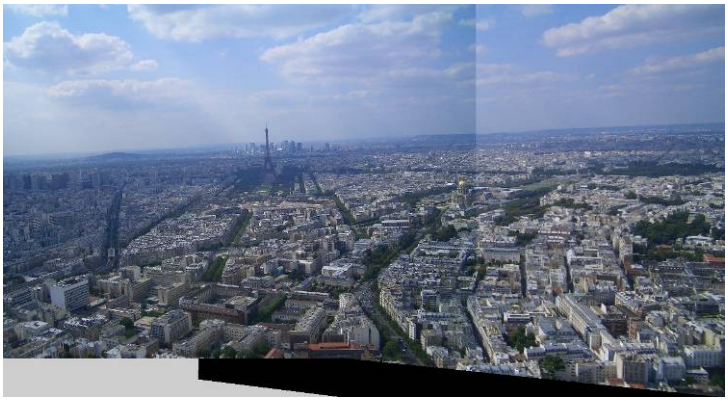


Figure 5: Shows the line artifact that results from simple merging algorithm

Borrowing an idea from the work of Debevec in [4], another attempt was made to blend the images together in the regions where the two images overlapped. A linear weighting was used such that an image's contribution to a certain pixel in the final merged image was weighted by where in the overlapped region the pixel was.

First to calculate the overlapped region, a pixel on the left edge of the Additional Image, (0, height/2), is mapped to its corresponding location in the Global Reference Frame, yielding (leftEdgeX, leftEdgeY). The overlapped region is then taken to be any pixel whose x value lies within [leftEdgeX,GlobalReferanceFrameWidth].

The weighting function for pixels (x_0,y_0) in the Global Reference Frame and Additional Image are then defined as:

$$weight_{GRF} = \frac{GRFWidth - x_0}{GRFWidth - leftEdgeX}$$

$$weight_{AI} = \frac{x_0 - leftEdgeX}{GRFWidth - leftEdgeX}$$

This blending method worked well in removing the image boundaries line which resulted from different exposures, but it had one drawback, blurred images. When manually specifying point correspondences, an emphasis was put on placing correspondences near the border between the two images, the idea being that if these areas were unaligned, the effects would be most noticeable. With blending, however, all regions of the image that do not overlap perfectly produce a noticeably blurred image (see Figure 6). This is especially problematic when the two images overlap significantly.

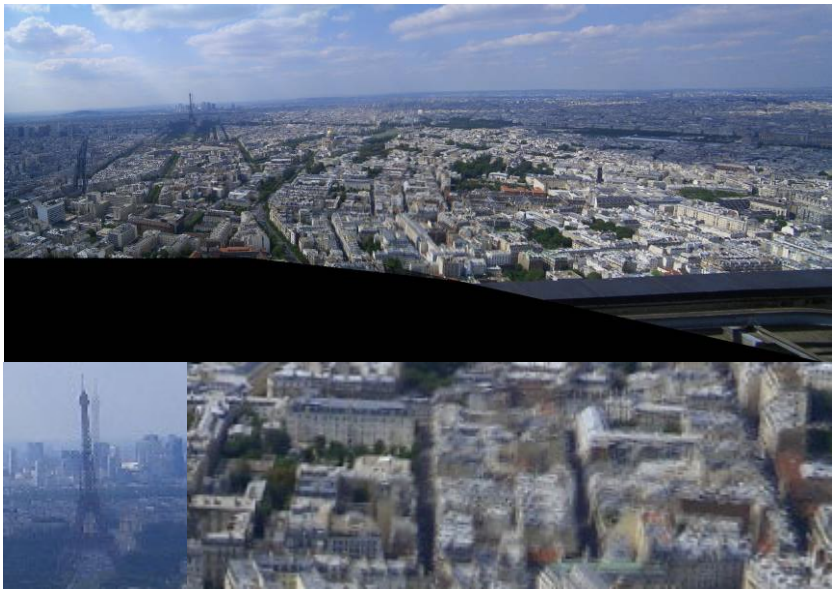


Figure 6: Result of Blending (Top),
Two portions extracted portions of scene demonstrating the negative effects of blurring (Bottom).

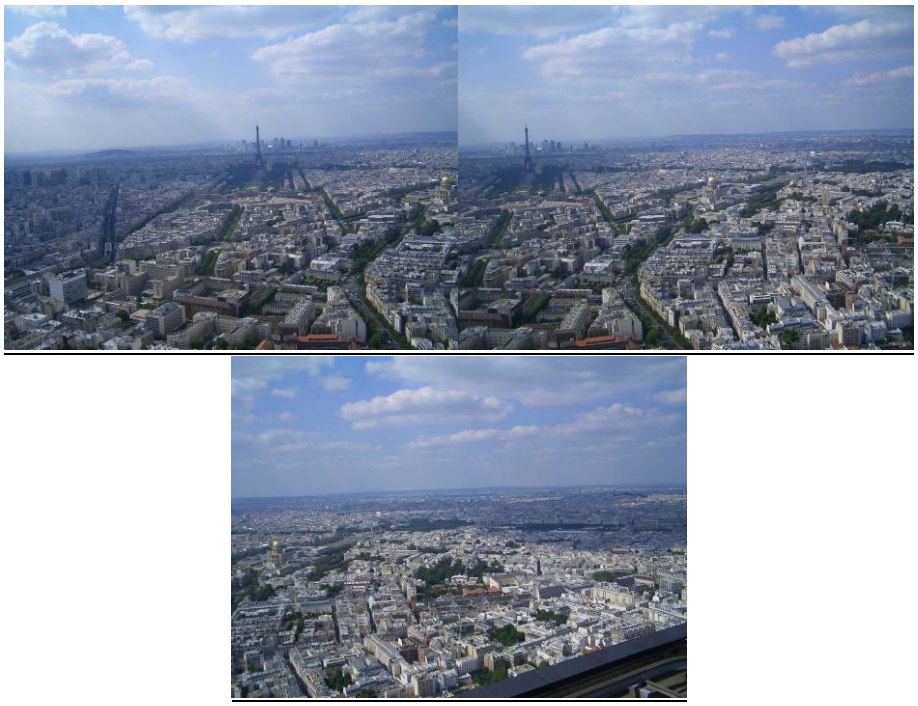
In order to mitigate the effects of blurring, but still gain the advantage eliminating the border-line between images, a final modification to the blurring algorithm was used. Rather than use weighted averaging to combine all pixels within the overlapped region, it was decided to only merge those pixels that lay within some window about the center of the overlapped region. The same blending procedure as previously described was used, however the weights were now determined by the distance between the center of the overlapped region +/- 50 pixels, in other words simulating that the overlapped region between images was at most 100 pixels wide. This change helped to mitigate the blending errors described above (see Figure 7).

Figure 7: Notice the complete removal of ghosting artifacts on the Eiffel Tower, as well as significantly decreased blurring on the buildings, despite still lying within the blended region.

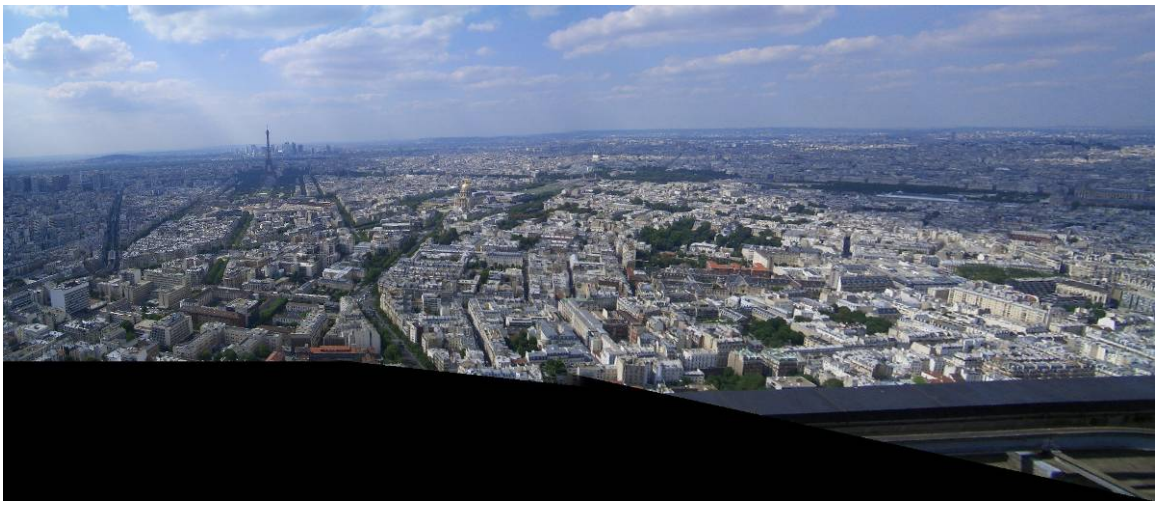
Note that in the case of noisy images (from video taken at night, for example), it may actually be desired to still use the full blending model, in hopes that the weighted averaging will remove noise from the images.

Results:

Paris, France. Taken Summer 2004 from atop Montparnasse Tower



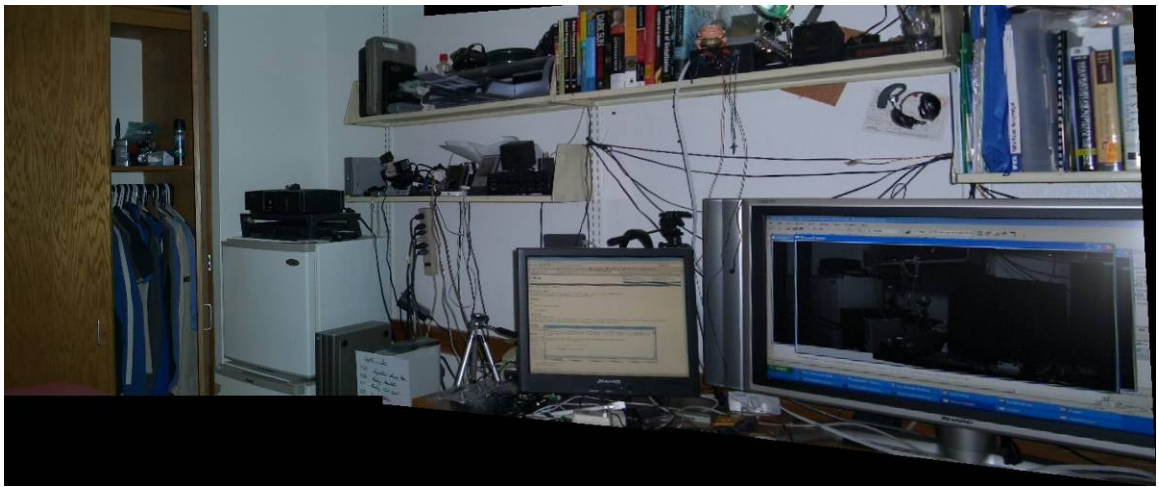
Source Images



Synthesis of Three Views
Note the Eiffel Tower in the upper left (as rarely seen from above).

My Room:

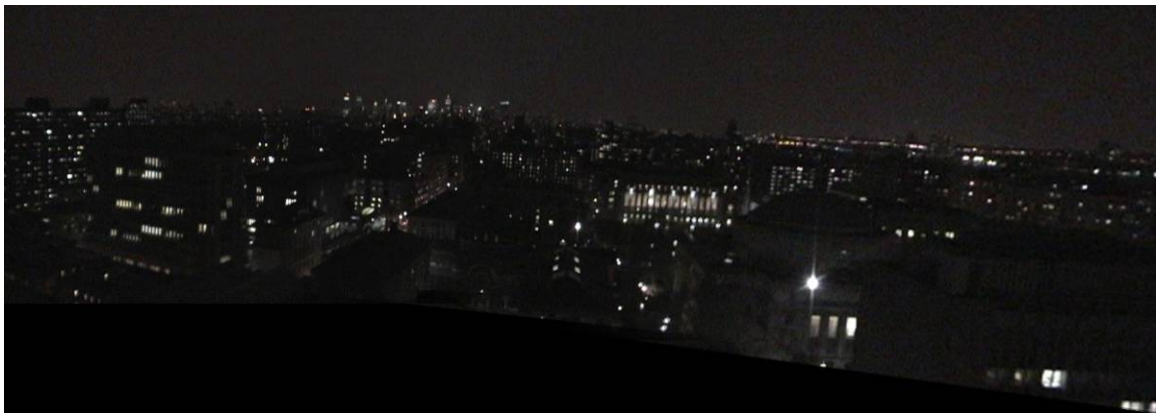
Source Images



Note the difference in color between the first image and the other two. The blurring helps a lot in reducing obvious lines between the two images.

Taken from the Roof of Mudd: (don't ask how ☺):

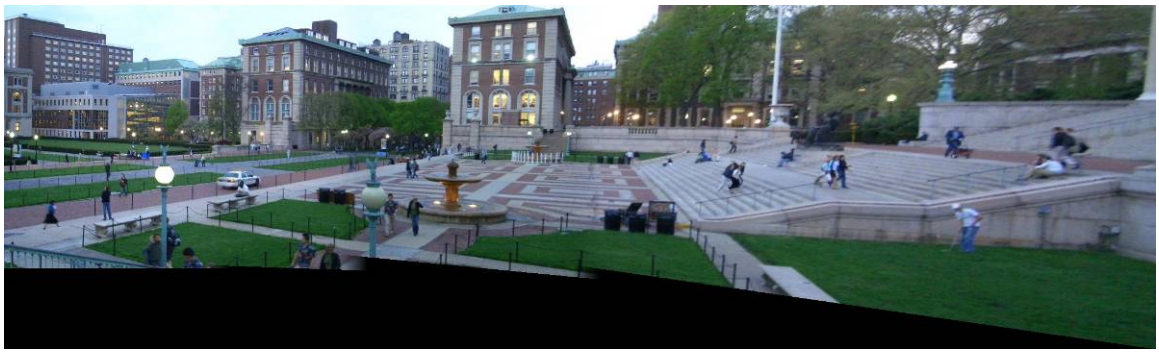
Source Images taken with Video camera



Resulting Panorama

Columbia Quad:

Source Images

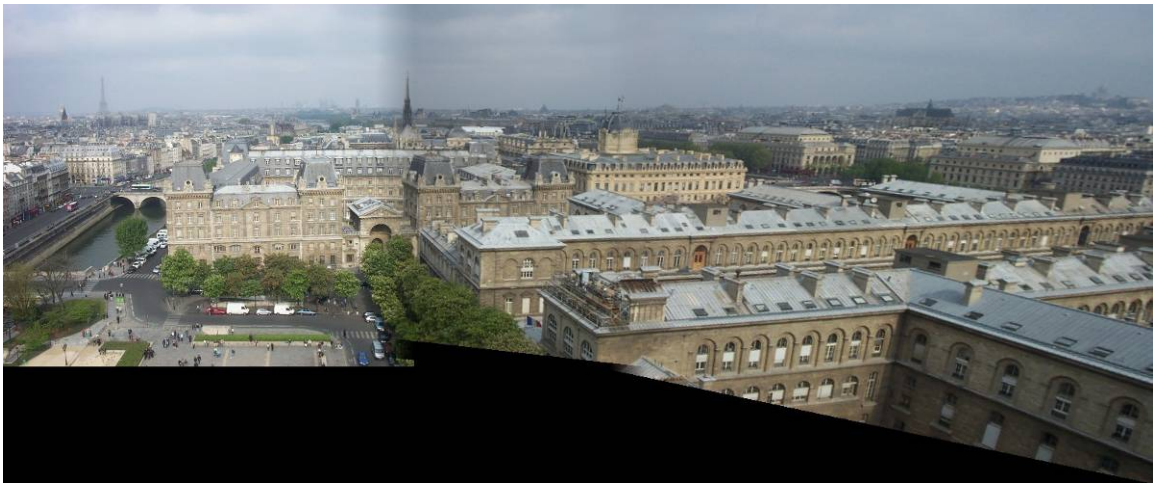


Note the consistency in the brick patterns in the center of the image.

Extracted from bottom left portion of result above. Notice the couple imaged multiple times as they were walking up the stairs.

Paris, France: Taken from atop Notre Dame in Spring of 2003.

Source Images



Resulting Panorama

Note that in this example, the differences in exposure between the first and second image are so great, that even with blending, one can still determine the separation between the two images in the result.

Discussion:

As demonstrated by the results presented, the objective of the program was reached. Using a combination of user input and automatic point correspondence, realistic panoramic images were successfully generated.

In relation to the target questions put forth in the official assignment, the primary phenomena studied was projective geometry and the process of rectification for the purpose of generating panoramas. We previously studied both the Fundamental matrix and the Essential matrix. The fundamental matrix assumes one knows neither the internal nor external parameters of the camera, and the essential matrix assumes one does not know any of the external parameters. The homography mapping, by contrasrt, assumes that all internal and half of the external parameters remain constant between images. The only free parameters are the rotation of the camera about its optical center.

Lastly, the issue of rendering scenes with images was also examined. A method similar to the one in [4] was utilized to blend multiple images together in a manner that, in many cases, generates realistic images.

Areas for Future Improvement:

Due to time constraints, numerous features which are believed to likely improve the results were not implemented. Utilizing a slow moving video camera would likely allow more point correspondences to be accurately calculated, given that the search window for corresponding points between successive video frames can be made substantially smaller.

The program described always used the first frame as the reference frame and warped all images with respect to it. Greater fields of view and images with less projective distortion could've been made if an image in the middle of the sequence were chosen as the reference frame, and additional frames could be added either to the left or right of it, instead of always falling to the right.

If one were to track both the homographies and inverse homographies of every image with respect to the global reference frame, it would also be possible to arbitrarily choose which frame to use as the global reference frame, simply by applying the inverse homography of the new GRF to the previous GRF, bringing its points into the new GRF, and then apply first the old homography followed by the inverse in order to map all other reference frames first into the old GRF and then into the new GRF. This would likely simulate very well the functionality of QuicktimeVR.

In automatically locating corresponding points, templates were always extracted from the global reference frame and compared against additional frames. Once corners in the global reference frame were coming from regions that were synthesized from perspectively warped images, the template matching procedure is less effective, as projective distortions continue to accumulate and decrease the similarity between the perspective-warped and additional images to be merged. A more efficient method would be to track corresponding points only between adjacent images, and then apply multiple homographies to eventually map a corresponding point into the global reference frame.

Lastly, points based on corners were used for calculating correspondences and evaluating reprojective error. In merging images, however, edges are the biggest cause for concern, since their misalignment is most noticeable. If manually specified edge correspondences were used for the RANSAC algorithm, better visual results may result. Also, edges could be taken into account in the blending algorithm, such that points are only blended if they do not lie on an edge, thus minimizing the blurring of edges that can occur. All correspondence point pairs were treated equally in determining error. Weighting the various points as a function of the certainty of the correlation as well as their proximity to the intersecting edge might also yield better visual results.

References:

[1] R. Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press. 2003.
[2] "Open Source Computer Vision Library." Intel Corporation. http://www.intel.com/technology/computing/opencv/index.htm
[3] J. Bouget, Camera Calibration Toolbox for Matlab.
[4] P. Debevec, C.J.Taylor, and J. Malik. Modeling and Rendering Architecture from Photographs:A hybrid geometry-and image-based approach. *SIGGRAPH 1996, pp11-20*