

**CSC2530F Visual Modeling**  
***Panoramic Mosaicing Project Report***  
**By: Shahzad Malik and Pablo Sala**  
*October 28, 2002*

## 1 Assumptions

In our implementation we made the following assumptions:

- The set of input images will correspond to images with small inter-frame motion.
- Each image must have a height and width that corresponds to a power of 2. If the images don't conform to this restriction, then they will be internally cropped to the closest power of 2 from the center outward. The power-of-2 restriction allows generating and traversing our pyramids more simple.
- All images in an input sequence should be of the same size and pixel depth, and should be captured with the same focal length.
- All images should have the same brightness from frame to frame (i.e.: no gain control adjustment, white balancing, etc.).

## 2 Implementation Details

We implemented the cylindrical panorama mosaicing technique presented by Szeliski & Shum in their *SIGGRAPH* paper [SZE97] and technical report [SHUM97]. This section will describe the various phases of our implementation in more detail.

### 2.1 Tools and Framework

Before describing the details of the alignment system, it is worth mentioning the system, tools, and development environment that facilitated the implementation.

**Implementation Platform:** The software was developed entirely in Windows, using the Microsoft Visual C++ 6.0 compiler.

**Basic Image I/O:** We used the freely available Portable Image Library (PIL) by Anthony Whitehead. This library provides a common interface for loading and saving images in a variety of popular file formats.

**Matrix Math:** The Newmat library that was provided as part of the mini-assignments for this project was used throughout our code, mainly to solve linear systems of equations but also for general-purpose matrix operations.

**Image Processing:** The Intel Image Processing Library (IPL) was used to manipulate images in a wide variety of ways, since the library provides built-in functionality for image filtering, colour conversion, etc.

**Graphical User Interface:** Rather than relying on a command-line interface, we used the FLUID/FLTK library to design a graphical front-end for our mosaicing software.

**Graphics Display:** The OpenGL graphics library was used in conjunction with FLUID to display 2D graphical components in the user interface window (such as the final mosaic image, input sequences, etc.)

## 2.2 Laplacian Pyramids

Our alignment system makes use of a coarse-to-fine search strategy, and thus requires the generation of Laplacian pyramids. The IPL library provides built-in support for generating Laplacian images, as well as image resizing operations. By restricting the input images to be powers-of-2 in width and height, the generation and traversal of the levels of the pyramid was simplified.

By using these image pyramids, we were able to increase the performance of our system as well as help prevent the alignment system from getting trapped in local minima during the search phase.

## 2.3 Cylindrical Panoramas

The core of our system consists of a *cylindrical panorama mapping system* as described in Section 2 of Szeliski [SZE97]. This simplifies mosaic generation considerably since we only need to estimate translational motion from frame-to-frame (in cylindrical coordinates). In other words, we warp each input image to a cylinder by mapping world coordinates  $\mathbf{p} = (X, Y, Z)$  to 2D cylindrical screen coordinates  $(\theta, v)$  using:

$$\theta = \tan^{-1}(X / Z), \quad v = Y / \sqrt{X^2 + Z^2}$$

### 2.3.1 Focal Length Computation

Note however that in order to perform this warping in the first place, we need to know the focal length of the camera. This is achieved by applying the *8-parameter perspective transformation algorithm* described in Section 3 of the Szeliski paper to a pair of input images. As explained in Section 5 of the technical report [SHUM97], we compute the following equations. (Note that the paper has errors in the focal length computation and the correct formulas are listed in the technical report):

$$f_0' = \sqrt{\frac{m_5^2 - m_2^2}{m_0^2 + m_1^2 - m_5^2 - m_4^2}}, \quad \text{if } m_0^2 + m_1^2 \neq m_3^2 + m_4^2,$$

$$f_0'' = \sqrt{-\frac{m_2 m_5}{m_0 m_3 + m_1 m_4}}, \quad \text{if } m_0 m_3 \neq m_1 m_4,$$

$$f_1' = \sqrt{\frac{m_0^2 + m_3^2 - m_1^2 - m_4^2}{m_7^2 - m_6^2}}, \quad \text{if } m_7^2 \neq m_6^2,$$

and

$$f_1'' = \sqrt{-\frac{m_0 m_1 + m_3 m_4}{m_6 m_7}}, \quad \text{if } m_6 m_7 \neq 0$$

where:

$$M = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & m_8 \end{bmatrix}$$

is the warping matrix  $M$  obtained by the algorithm.

We then choose one of  $f_0'$  and  $f_0''$  ( $f_1'$  and  $f_1''$ ) by favouring the term which has the larger denominator, and call it  $f_0$  ( $f_1$ ). We finally compute the focal length  $f$  as:

$$f = \sqrt{f_0 f_1}$$

In order to obtain a good estimate of the focal length by means of this warping technique it is necessary that the pair of images chosen correspond to an angle difference that is somewhat significant but at the same time still contain a lot of overlap.

The user is asked to select a pair of images with such characteristics and provide a good initial alignment of these pictures, so that the warping won't fall in a local minimum but instead converge to the real alignment position. To indicate the initial alignment the user selects four corresponding points in a pair of images. From such a set of points the initial warping matrix  $A$  is computed, where the elements  $a_{ij}$  of  $A$  are obtained by solving the following system of equations:

$$\begin{cases} x_i = (a_{11}r_i + a_{12}c_i + a_{13}) / (a_{31}r_i + a_{32}c_i + 1) \\ y_i = (a_{21}r_i + a_{22}c_i + a_{23}) / (a_{31}r_i + a_{32}c_i + 1) \end{cases}$$

where  $(x_i, y_i)$  and  $(r_i, c_i)$ ;  $1 \leq i \leq 4$  are corresponding points in the two images.

### 2.3.2 Coarse-to-Fine Alignment

After the focal length is computed and each image is warped to the cylindrical coordinates, we perform the alignment between the transformed images. Such alignment corresponds to a simple translation. This procedure is achieved using a coarse-to-fine approach in which Laplacian pyramids for each image are used (as described earlier). In our experiments we obtained the best results and generally avoided local minima using as

our coarsest level images of size of 64 in the lowest dimension. We first perform an exhaustive (brute-force) search in the coarsest level of each pair of images  $I_k, I_{k+1}$  trying to find the translational alignment  $t$  that minimizes the intensity error:

$$\sum_{\vec{x}} (I_k(\vec{x}) - I_{k+1}(\vec{x} + t))^2 .$$

With  $t$  as our initial estimation of the translational displacement, we apply the Lucas-Kanade registration algorithm [LK81] in a coarse-to-fine hierarchy [BAHH92]. This algorithm is simply the 8-parameter warping algorithm modified to only estimate the 2 translational parameters in cylindrical images.

## ***2.4 Planar Panoramas***

Since we implemented the 8-parameter alignment algorithm, we were also able to generate planar panoramas of scenes as well. This requires the user to manually click on four corresponding, non-collinear points in each image. For a while we were hoping that the 8-parameter algorithm combined with a coarse-to-fine search strategy would be sufficient to align almost any set of input images, with a low risk of falling into local minim. Unfortunately this was not the case, even at the coarsest levels of the pyramid, since the number of free variables in the general perspective warp makes gradient descent convergence difficult. This is what motivated us to implement the simpler cylindrical panoramas, and then use the 8-parameter algorithm for focal length computations and manual planar panorama generation.

## ***2.5 Mosaic Stitching***

The output from both our cylindrical and planar alignment algorithms consists of the set of input images and an associated transformation matrix describing the warp required between each sequential pair of images. Therefore in order to build a final mosaic image, we need to stitch all the input images together and form a final composited image where overlapping input images are blended together.

The paper by Szeliski proposes traversing each pixel in the final mosaic image, and determining the contribution each input image makes at that specific pixel. Unfortunately for large mosaics built from a large set of input images, this is an extremely inefficient and time-consuming task. In our original implementation for example, an approximate 8000x512 mosaic built from an input sequence of over 100 images took over 20 minutes to stitch together (i.e. after the alignment phase).

With some small modifications, we instead perform the final stitching by determining each input image's bounding box in the final mosaic image by warping the image bounds by the associated transformation. This allows us to only traverse those bounding box pixels within the mosaic image. This small implementation improvement allowed that same 8000x512 mosaic to be generated in less than a minute.

In order to avoid sharp edges and aliasing effects in the final mosaic image, our stitching algorithm performs a bilinear filtering step when determining the colour contribution from an input image at each given mosaic pixel. The bilinear filtering algorithm simply samples 4 colours in a 2x2 grid at a given pixel location in the input image. The final contribution into the mosaic is computed as a weighted sum of these four neighbouring colours, based on the subpixel distance of the pixel location to each of the neighbours.

## ***2.6 Mosaic Display***

The final mosaic image is saved to disk in the file “mosaic.jpg”, and the resulting mosaic is also displayed on the screen as a planar image. The JPG file can then be imported into a special-purpose viewer that can warp the image onto a cylinder (see the results webpage for two example panoramas using a cylindrical viewer).

Note that since our rendering system uses OpenGL to display the final mosaic, we attempt to convert the mosaic to a power-of-2 texture internally so that we can render it using OpenGL texturing hardware. Unfortunately, there is a physical texture size limit in many OpenGL video cards, and sometimes the larger mosaics that are generated will not appear on the screen correctly (they will appear as a solid rectangular region). In these cases, the mosaic image can simply be loaded into a general purpose image viewer to visualize the final results.

## ***2.7 Memory Management***

At the beginning of the project we didn't take into consideration that the mosaic could be generated from image sequences consisting of small inter-frame motion, which meant that a large number of input images was going to be necessary. Therefore our original implementation simply loaded all the images into memory and constructed their pyramids during the start-up phase. This is extremely inefficient from some of the large input sequences consisting of 100-200 input frames. As a result, we have improved our memory management by only dealing with pairs of images at a time. This has resulted in a significant performance increase for large sequence processing, since the OS no longer has to continuously page large chunks of memory to the hard drive.

## **3 Discussion**

We didn't perform any feathering or similar technique before the blending due to lack of time. This fact can be seen in the mosaic image of the “outdoor” mosaic where well-defined square regions with different brightness are very noticeable. The shots of this outdoor sequence were taken with a camera for which the gain control parameters hadn't been locked. On the other hand, the mosaic generated by the sequence inside the classroom does not show these differences of brightness because the shots were taken with a locked camera.

Another feature that we would have liked to implement would have been the automatic gap closing technique. For the cylindrical panoramas, the gap closing would have

consisted of cropping the end of the mosaic by matching the last frame of the sequence with the first, followed by an affine warp to handle “pixel drift” that is commonly seen with a rotating camera (see webpage results).

As mentioned earlier, the 8-parameter algorithm that we have implemented is slow to converge. From our experiments we could see that it also easily gets stuck in local minima unless an extremely good initial estimate is given manually using non-collinear point correspondences. The coarse-to-fine strategy improves the algorithm considerably, but it still doesn’t allow us to generate automatic panoramas. This algorithm is good for facilitating focal length computations, by computing the projective warp between two images at relatively different angles (at least 20° apart).

We also implemented the 3-parameter (rotation) algorithm described in Section 3.2 of the paper, and we ran a few experiments with it. It seems to work at least as well as the 8-parameter algorithm, as was expected. The major advantage with the 3-parameter and the 8-parameter algorithms is the ability to generate mosaics on surfaces other than simply spheres or cylinders. Unfortunately, due to time constraints, we only had time to experiment with the cylindrical and planar manifolds.

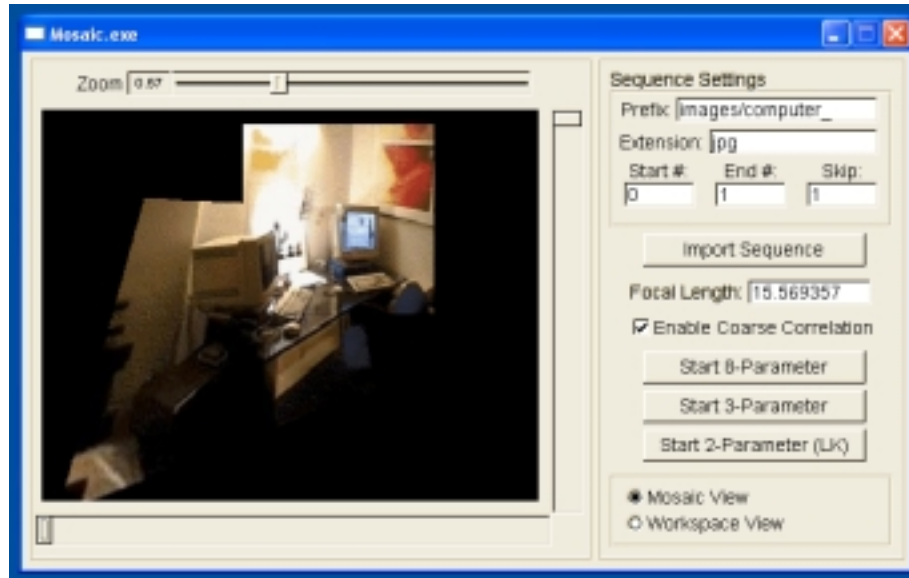
## 4 Software Usage

The input images follow a simple naming convention in order to simplify the loading of large sequences. The first part of each frame is a descriptive prefix, followed by a four-digit sequence number. The file name is then appended with a file extension describing the image type. For our experiments we exclusively used JPG files, but the PIL image library also allows for loading of other file types such as TIFF, PNG, GIF, and BMP. In other words, filenames consist of the following:

**<sequence prefix><nnnn>.<ext>**

For example, “images/room0027.jpg” would represent the 27<sup>th</sup> JPG image in the “room” sequence, located in the “images” subdirectory. Note that the subdirectory name is considered to be part of the prefix.

The functionality of the software is contained within a FLUID-based graphical user interface, which facilitates the loading of the above mentioned image sequences (see Figure 1). Additionally, the use of a GUI allows us to provide a point-and-click interface for selecting the four corresponding points when attempting to estimate an initial 8-parameter warp (for focal length computations).



**Figure 1 – A screenshot of the FLUID-based user interface**

## 5 Conclusions

Overall we are pleased with the results obtained in our cylindrical and planar mosaics. It would have been nice to have the 3-parameter or 8-parameter algorithm working as automatically as the Lucas-Kanade translational algorithm, but unfortunately it didn't turn out that way. After reading the Szeliski and Shum papers, we believed that the algorithms would work in a completely automatic manner, but that doesn't seem to be the case. For the longest time we were having no luck with the 3-parameter and 8-parameter algorithms which was causing us to fall behind on some of the more interesting mosaicing features, so we finally decided to implement the cylindrical mosaic approach and everything worked quite nicely. Maybe with a little more tweaking we might be able to get the other two algorithms working more automatically, but for now we are happy with the automatic cylindrical mosaic results.

## 6 References

- [BAHH92] J. R. Bergen, P. Anandan, K. J. Hanna, and R. Hingorani. Hierarchical modelbased motion estimation. In *Second European Conference on Computer Vision (ECCV'92)*, May 1992. Springer-Verlag. pp. 237-252.
- [LK81] B. D. Lucas and T. Kanade, An iterative image registration technique with an application in stereo vision, *Proc. Image Understanding Workshop*, pp. 121-130, 1981.
- [SHUM97] H. -Y. Shum, R. Szeliski, Panoramic Image Mosaics, *Microsoft Research – Technical Report MSR-TR-97-23*, 1997
- [SZE97] Szeliski and H.-Y. Shum, Creating Full View Panoramic Image Mosaics and Environment Maps, *Proc. ACM SIGGRAPH*, 1997.