

BROWN UNIVERSITY  
Department of Computer Science  
Master's Project  
CS-97-M11

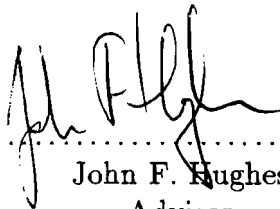
“Construction of an Image Mosaicing  
and  
Image-based Rendering System”  
by  
Chenghui Luo

**Construction of an Image Mosaicing  
and  
Image-based Rendering System**

by

Chenghui Luo

Submitted in partial fulfillment of the requirements for  
the Degree of Master of Science  
in the Department of Computer Science at Brown University



.....

John F. Hughes  
Advisor

May 13, 1997

## Acknowledgments

I would like to take this opportunity to thank all the people who provided help in this project. First I would like to thank my advisor John “Spike” Hughes for his guidance through this project. Without his encouragement, I would have been discouraged by those large number of images taken from a video camera. Mark Oribello helped to take images and digitize them; Loring Holden answered some programming questions; and Bing Chin provided helpful conversations – to all of them, I would like to say: “Thank you!”

I would also like to thank Bob Zeleznik for his guidance in a preliminary research project and the Computer Science Department and the graphics group for providing a happy and pleasant environment.

# Contents

<b>Acknowledgments</b>	<b>i</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Abstract . . . . .	1
1.2 Previous Work on Image-based Rendering . . . . .	1
<b>2 Description of Project</b>	<b>3</b>
2.1 Synthetic Environments . . . . .	3
2.2 Real World . . . . .	5
2.2.1 Image Acquisition . . . . .	5
2.2.2 Image Processing and Mosaicing . . . . .	5
2.2.3 View generating . . . . .	9
2.3 Conclusion and Future Work . . . . .	10

# Chapter 1

## Introduction

### 1.1 Abstract

Traditional 3D computer graphics systems take scenes consisting of geometric primitives as input. But the real world is too complex to be modeled geometrically. In this project, we use a set of images taken from real world and render them using a 3D graphics package. This approach of image-based rendering is a convincing way to build a virtual environment on a computer screen.

### 1.2 Previous Work on Image-based Rendering

Traditional 3D computer graphics systems take scenes consisting of geometric primitives, e.g., cubes, cones, cylinders, spheres, ..., as input. Such an environment is modeled by these primitives of different materials and a set of lights, and then the rendering system computes and outputs images.

With the increasing requirements of many applications, e.g., virtual reality, this traditional approach is not enough – the real world is not likely to be modeled by a finite set of relatively simple geometric primitives. Naturally, people turn to data

directly taken from the real world, e.g. images taken by video cameras. Images represent the real world accurately, and so it is possible to build a virtual environment from a set of images – provided that an image-based rendering system exists.

In recent years, several image-based rendering systems have appeared. A commercial product, QuickTime VR, for virtual environment navigation first appeared at Apple Computer [1]; and an image-based rendering system based on plenoptic modeling was built at UNC [6]. These two systems require some image registration work to obtain an environment map of each point in the environment, i.e., a panoramic projection of the whole space to a closed surface, e.g., sphere, cube, cylinder, which encloses this point. Some systems do not require environment maps, e.g., the light field rendering approach from Stanford University [5] and the Lumigraph approach from Microsoft Research [2], where a different representation method, light field or Lumigraph, is used.

From these systems, one sees that the basic framework of an image-based rendering system includes the following three necessary elements:

- An image acquisition system which captures images from real world.
- An image processing system which provides convenient representation of the real world.
- A view generating system which generates new views of an environment from a new direction or position.

# Chapter 2

## Description of Project

In this project, we explore and implement the basic but important elements of an image-based rendering system. Our goal is to build a basic and relatively simple system that can register images and generate views for the case that the view point is fixed in an environment so that the degrees of freedom are only those of rotations.

### 2.1 Synthetic Environments

We first show the basic idea of image-based rendering for the easier case of synthetic environments, i.e., environments modeled by geometric primitives. In this project, we use Open Inventor, a 3D graphics package, to model a 3D synthetic world. To acquire images, we set the `heightAngle` field of `SoPerspectiveCamera` to  $\frac{\pi}{2}$  and `aspectRatio` to 1 so that its width angle equals height angle and the film plane of the camera is a square. Then to obtain an environment map of the center of the camera, we take 6 pictures from 6 perpendicular directions, corresponding to the positive and negative directions of the  $X$ ,  $Y$  and  $Z$  axes, to obtain 6 images of the environment. Now we do not need to do any image processing on these images, since the 6 images cover all of the space precisely, so they are a perfect representation of the synthetic environment,

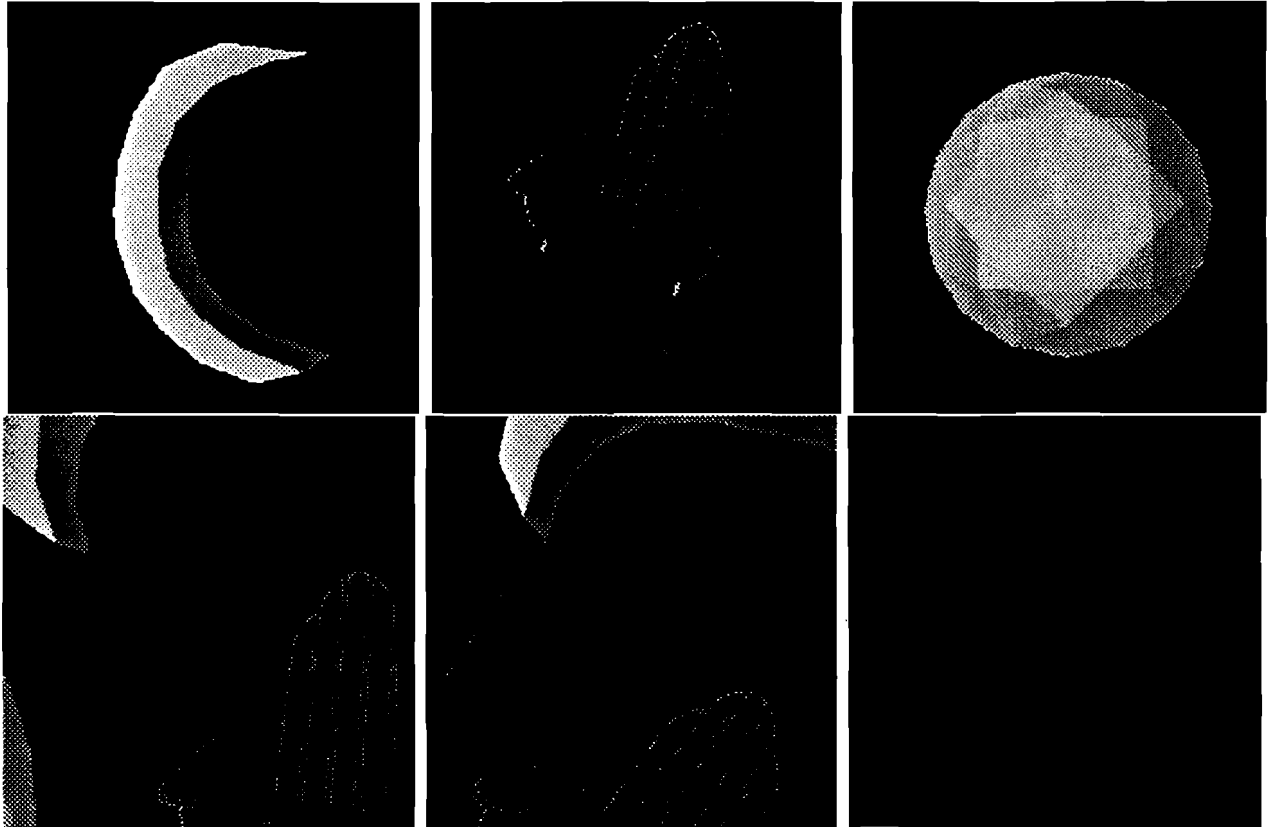


Figure 2.1: **Image-based rendering of a synthetic environment.** The upper row shows three images and the bottom left shows an image from image-based rendering. The bottom middle is an image with noises and the bottom right a modified cube.

with no holes or redundancies.

To generate new views from the a new direction, we just need to construct another synthetic environment, where the same camera that is used before is placed at the center of a cube and all the 6 images are texture mapped to the 6 faces of the cube. Then views from this camera in the cube correspond to views in the original synthetic environment. Notice that here we just use 6 images as inputs to recover the original synthetic environment. So the basic steps of image-based rendering are fairly simple for this case. An example is shown in Figure 2.1.

As Figure 2.1 (bottom middle) shows, sometimes there are noisy pixels at edges of a cube, due to the z-buffer rendering algorithm of Open Inventor. To avoid this, it is necessary to give the cube a small exterior offset (bottom right of the above figure)



so that seamless alignment of views is achieved.

## 2.2 Real World

For the case of real world data, we use a video camera to acquire images and `GPImageCanvas` of GP, a 2D graphics package, to process images. To generate new views, we use `SoPerspectiveCamera` and `SoExaminerViewer` of Open Inventor.

### 2.2.1 Image Acquisition

To acquire images of the real world from a point, we set a video camera on a fixed tripod so that the only movement of the camera is rotation. We first rotate the camera horizontally through  $360^\circ$  and then raise and lower the view vector to capture images in other directions. In this project, images are taken from Spike's office.

After images are taken, we digitize them by using commercial software, such as QuickTime video system, to obtain frames that the computer can access.

### 2.2.2 Image Processing and Mosaicing

In our approach, we use an environment map to represent real world, so we need to mosaic images together, since images taken as above have common parts and we need to match them.

We use a well-known, basic observation that the common part of two images, taken from the same point in two different directions, can be projected from one to the other by a 3 by 3 matrix [4, 9, 3]:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & m_8 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix},$$



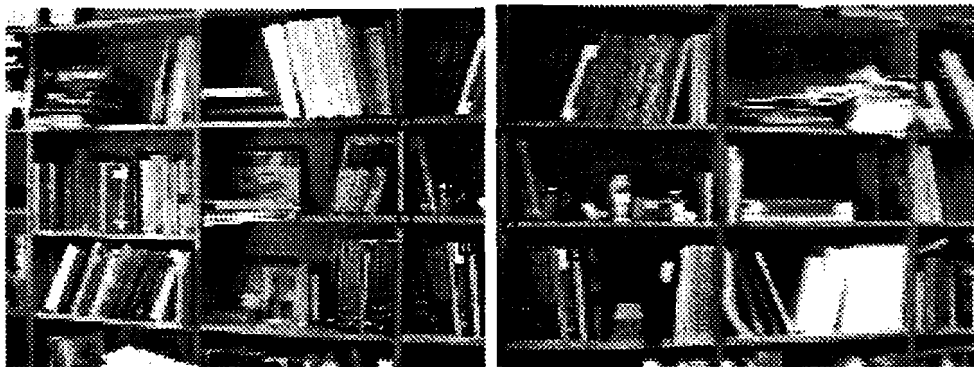


Figure 2.2: **Image mosaicing using manual correspondence.** The 8 black spots indicate correspondence of 4 pairs of pixels.

Another manual method is to use transparency, which is supported in Open Inventor, to find the matrix  $\mathbf{M}$ . We can first put two images at an initial position in 3D by texture mapping them to two rectangles, and then use a virtual track ball manipulator to rotate, or a handle box manipulator to translate, one of the two pictures until one sees them match well. Then we use a write action to save the transformation of that image. This method is not very accurate, but it is simple and intuitive. In practice, it is not bad. Another similar method is to use image features or image clues at the boundary of one image to match them, without using transparency. An example is shown in Figure 2.3.

The basic idea of automatic mosaicing, as described by Szeliski [8], is to find the fundamental matrix  $\mathbf{M}$  by minimizing an error function (sum of squared differences of intensities) between the two images:

$$E(\mathbf{m}) = \sum_i [I'(x'_i, y'_i) - I(x_i, y_i)]^2 = \sum_i e_i^2.$$

where  $i$  is taken over all pixels of the image to be mapped and  $I'(x'_i, y'_i)$  is the intensity of pixel  $(x'_i, y'_i)$  in the other image, which can be approximated by bilinear resampling.

To minimize this error function, we apply the Levenberg-Marquardt algorithm [7, 8]. Thus we need to compute an approximate Hessian matrix  $\mathbf{A}=(a_{jk})$  and a

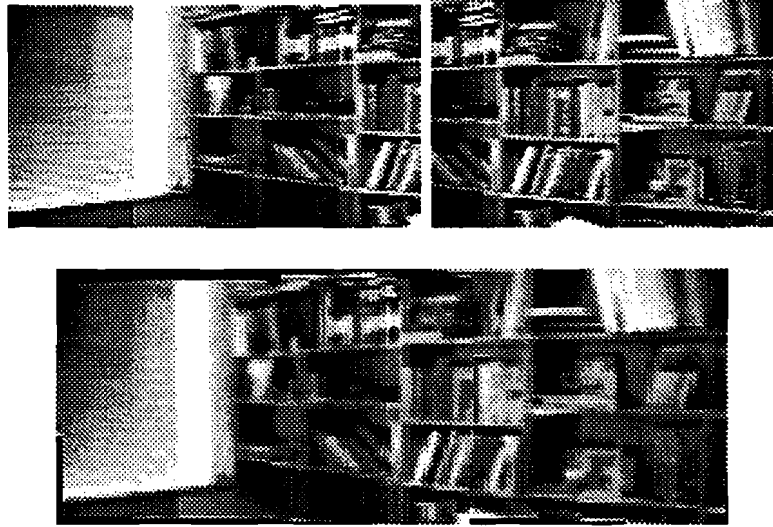


Figure 2.3: **Image mosaicing using image features.** The upper row shows two original images and the lower row the mosaiced image.

weighted gradient vector  $\mathbf{b} = (b_j)$  :

$$a_{jk} = \sum_i \frac{\partial e_i}{\partial m_j} \frac{\partial e_i}{\partial m_k}, \quad b_j = - \sum_i e_i \frac{\partial e_i}{\partial m_j},$$

where

$$\frac{\partial e_i}{\partial m_j} = \frac{\partial e_i}{\partial x'} \frac{\partial x'}{\partial m_j} + \frac{\partial e_i}{\partial y'} \frac{\partial y'}{\partial m_j},$$

and  $\frac{\partial e_i}{\partial x'} = \frac{\partial I'}{\partial x'}$  can be approximated by the difference of intensities of adjacent two pixels.

Let matrix  $\tilde{\mathbf{A}}_\lambda = (\tilde{a}_{jk})$  be as follows:

$$\tilde{a}_{jk} = \begin{cases} a_{jk} & \text{if } j \neq k \\ a_{jk}(1 + \lambda) & \text{if } j = k, \end{cases}$$

then the Levenberg-Marquardt algorithm is:

Given an initial guess for the set of fitted  $\mathbf{m}$ , compute  $E(\mathbf{m})$ .

Pick a modest value for  $\lambda$ , say  $\lambda = 0.001$ .

While ( $E(\mathbf{m}) > \text{a threshold}$  or iteration number  $< \text{some constant}$ ) {

Solve the linear system  $\tilde{\mathbf{A}}_\lambda \delta \mathbf{m} = \mathbf{b}$  for  $\delta \mathbf{m}$ .



Figure 2.4: User interface of image mosaicing.

```
Evaluate  $E(\mathbf{m} + \delta\mathbf{m})$ .  
If  $E(\mathbf{m} + \delta\mathbf{m}) \geq E(\mathbf{m})$ ,  
    increase  $\lambda$  by a substantial factor, say 10;  
Else decrease  $\lambda$  by a substantial factor,  
    and update the trial solution  $\mathbf{m} \leftarrow \mathbf{m} + \delta\mathbf{m}$ .  
}
```

This algorithm works well if two images have enough common parts, i.e., camera does not move too much. Otherwise, a global image registration must be done, but currently this is not further explored in this project.

Refer to Figure 2.4 for the user interface of this image mosaicing process.

### 2.2.3 View generating

Once images are well mosaiced, just as the synthetic case, we can texture map the mosaiced image to a cube or a cylinder. Then by placing a camera at the center of a cube or a cylinder and manipulating it, we can see new views from new directions. For images obtained by horizontal spinning, we use a cylinder. Figure 2.5 shows the mosaiced image of horizontal spinning of Spike's office and two shots from image-based rendering.

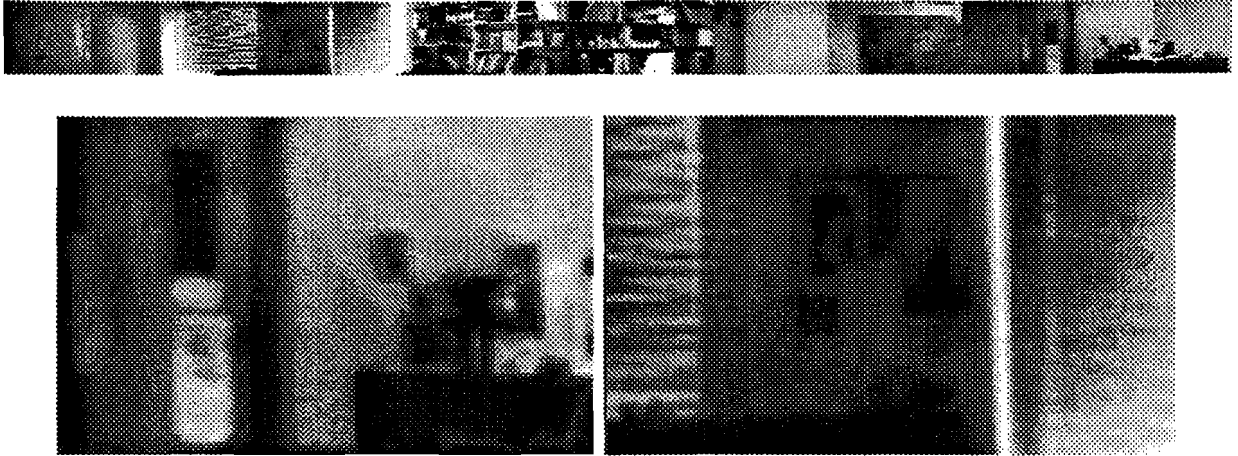


Figure 2.5: **Horizontal spinning of Spike's office.** The lower row shows two shots from image-based rendering.

## 2.3 Conclusion and Future Work

In this project, we investigated the basic framework of image-based rendering and implemented it by building a practical system. Image mosaicing is also applied to obtain an environment map. In practice, we find that the combination of a 2D and a 3D graphics package is very convenient and helpful: the 2D graphics package performs image mosaicing and processing, and the 3D graphics package performs image-based rendering. This is in real time on SGI workstations, thanks to SGI's special texture mapping hardware.

Currently this project has some known drawbacks. The biggest one is that sometimes the images are not well mosaiced due to the approximations in the algorithm and big intensity gradient between boundaries of adjacent images. Careful global registration should be a solution to fix this problem.

Another problem is to improve the speed of the automatic mosaic process. Currently we have used recursive increments to replace multiplications, but due to many times of bilinear resamplings and the complexity of the linear system, the speed is not fast enough. Typically, to mosaic two images of resolution of 100 by 100 takes

about 2 to 3 minutes.

One interesting question may be to simplify the computation of the matrix  $\mathbf{M}$ . We tried to decouple it into translations and rotations, but did not succeed. If this succeeds, the mosaic process will be faster.

Finally, a better device such as a pivot instead of just a tripod is more desirable for image acquisition through a video camera. The accuracy of image mosaicing depends on the accuracy of the original images.

# Bibliography

- [1] S. Chen, *QuickTime VR - An Image-Based Approach to Virtual Environment Navigation*, Computer Graphics (Proc. SIGGRAPH'95), pp. 29-38.
- [2] S. Gortler, R. Grzeszczuk, R. Szeliski, M. Cohen *The Lumigraph*, Computer Graphics (Proc. SIGGRAPH'96), pp. 43-54.
- [3] O. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint*, MIT Press, Cambridge, Mass., 1993.
- [4] P. Heckbert, *Fundamentals of Texture Mapping and Image Warping*, Masters Thesis, Dept. of EECS, UCB, Technical Report No. UCB/CSD 89/516, 1989.
- [5] M. Levoy, P. Hanrahan *Light Field Rendering*, Computer Graphics (Proc. SIGGRAPH'96), pp. 31-42.
- [6] L. McMillan, *Plenoptic Modeling: An Image-Based Rendering System*, Computer Graphics (Proc. SIGGRAPH'95), pp. 39-46.
- [7] W. Press et al., *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edition, Cambridge Univ. Press, Cambridge, England, 1992.
- [8] R. Szeliski, *Video Mosaics for Virtual Environments*, IEEE Computer Graphics and Applications, Mar. 1996, pp. 22-30.



- [9] L. Westover, *Footprint Evaluation for Volume Rendering*, Computer Graphics (Proc. SIGGRAPH'90).