DIGITAL RECONSTRUCTION OF DISTRICT SIX ARCHITECTURE FROM ARCHIVAL PHOTOGRAPHS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,

FACULTY OF SCIENCE

AT THE UNIVERSITY OF CAPE TOWN

IN FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Christopher R. J. de Kadt

May 2007

Supervised by

James Gain and Patrick Marais

Abstract

In this thesis we present a strategy for reconstructing instances of District Six Architecture from small sets of old, uncalibrated photographs that are located in the District Six Museum photographic archive.

Our reconstruction strategy comprises two major parts. First, we implement a geometry reconstruction framework, based on work by Debevec et al. [1996]. This is used to reconstruct the geometry of a building given as little input as a single photograph. The approach used in this framework requires the user to design a basic model representing the building at hand, using a set of geometric primitives, and then define correspondences between the edges of this model and the edges of the building that are visible in the photographs. This approach is effective, as constraints inherent in the geometry of architectural scenes are exploited through the use of these primitives.

The second component of the reconstruction strategy involves texturing the reconstructed models. To accomplish this, we use a combination of the original textures extracted from the photographs, and synthesized textures generated from samples of the original textures. For each face of the reconstructed model, the user is able to use either the original texture material, synthesized material, or a combination of both to create desirable results.

Finally, to illustrate the effectiveness of our reconstruction strategy, we consider three example cases of District Six architecture and their reconstructions. All three examples were reconstructed successfully, and using findings from these results, critical analyses of both aspects of our strategy are presented.

Acknowledgements

First and foremost, I would like to thank the NRF for the financial support provided to me throughout the duration of my project.

Second, I would like to thank the members of the CVC lab, who have all contributed to this work in some way. Specifically, I would like to thank Carl for his coffee and company during our many late night work sessions, and Bruce for his brilliant brain, capable of solving even the most bamboozling problems. To James and Patrick — thank you for your dedicated supervision. I could not have asked for more.

Third, I would like to thank the people who, although not directly involved in this project, helped make it possible. To my family, Julia, Alison and Ken, thank you for your continued support.

And to the DOTA crew — thanks for all the good times. Just remember — I 0wn j00.

Finally, I would like to extend a reverential thank you to the Flying Spaghetti Monster, for always guiding me down the right path with His Noodly Appendage.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

This thesis aims to address the problem of producing accurate, textured digital reconstructions of instances of District Six Architecture using the archive of photographs located at the District Six Museum in Cape Town, South Africa. This is particularly challenging, as the District Six area was demolished over thirty years ago, and no new data can be captured. As a result, solutions to the problems associated with using a limited supply of old, low quality material must also be investigated.

The following three sections provide a brief introduction to Digital Reconstruction, Heritage Reconstruction and District Six, after which the precise goals and aims of this project are presented.

1.1.1 Digital Reconstruction

Digital reconstruction of three-dimensional objects is a well studied topic in Computer Graphics. Its uses are extremely wide-ranging, and numerous techniques have been developed to aid in the reconstruction process.

The ability to produce digital reconstructions of real-world objects is extremely valuable in many fields, and imperative for the success of others. Major areas of use include architecture, engineering, medical imaging and heritage reconstruction.

In architecture and engineering, having a digital model of an object allows one to easily visualise and test any modifications one might want to make to that object. In medical imaging, techniques such as computed tomography, that are able to produce digital models of a patient's organs, may offer a physician insight that might otherwise only be available through surgery.

In the field of heritage reconstruction, building digital models of artifacts and architectural scenes that have been destroyed is often the only means by which they can be effectively visualised. Digital reconstructions are also used in this field to aid in the preservation of heritage artifacts that are in danger of being destroyed — although this cannot actually prevent damage to such artifacts, it does provide an excellent means of record keeping for historians or archaeologists.

Given the variety of situations in which digitally reconstructed models are required, the techniques developed to assist in their creation, also, fall into many different categories.

In situations where accurate data pertaining to the structure of the object exists, manual techniques, where the user controls almost every aspect of the reconstruction through the use of CAD tools, may be used. Where such data is unavailable, but the object still exists, three-dimensional scanning (or range scanning) methods are frequently applied. These include time-of-flight and triangulation laser scanners.

Finally, where only photographic data is available, or budget constraints prohibit the use of range scanners, photogrammetry-based methods may be employed. These typically involve using multiple images, coupled with stereopsis, to recover the geometric structure of the object at hand. Although point-based photogrammetric methods have been developed to solve general reconstruction problems, more specialised techniques, designed specifically to solve the problem of reconstructing architectural scenes, are available.

One such technique is described by Debevec et al. [1996]. Here, inherent constraints in architectural scenes, such as parallelism and orthogonality of edges (or faces) are used to reduce the reconstruction problem to a simpler form. This technique forms the basis for our work on District Six architecture reconstruction.

1.1.2 Heritage Reconstruction

In the field of heritage reconstruction, techniques are required to reconstruct or model both extant objects, as well as those that have already been destroyed. In conjunction with this, the subsequent uses of, and reasons for the reconstructions vary greatly, and as a result there exist many different sets of requirements that may need to be satisfied.

Reasons for reconstructing heritage sites include not only visualisation and preservation, as mentioned above, but also education, tourism and object interaction. Although most of these uses have similar requirements (e.g., high geometric detail, photo-realism and a high level of cost-efficiency), the order of importance of these requirements is dependent on the application at hand. For instance, if the reconstruction is to be used for a display in a museum, a high priority would be photo-realism. However, if the purpose is to assist in the rebuilding of a heritage site, then geometric detail becomes more important. Consequently, each situation is unique and presents its own difficulties, and there is currently no generic solution that is able to solve every problem.

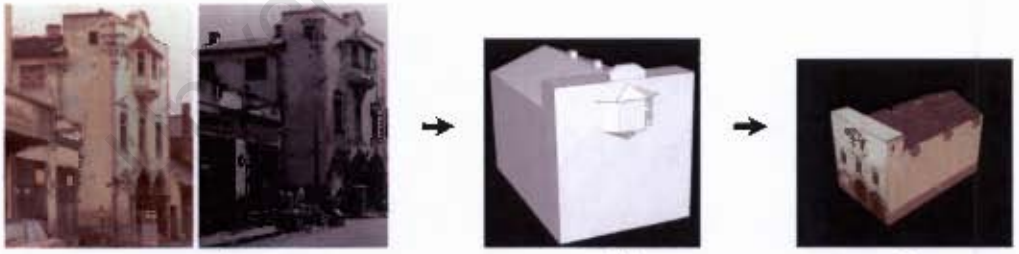
Although general techniques may be applied in the solutions to some of these more specific problems, they are often unable to handle the reconstructions in a satisfactory fashion, and situation-specific modifications must be made to acquire decent results. An example of this is the work by El-Hakim et al. [2004], which combines photogrammetric techniques and range scanning in order to reconstruct large, existing sites, with sufficient fine detail. A further example is discussed by Gardner et al. [2003], where linear light source reflectometry is used to capture the material properties of an artifact which may be used in a lighting-independent reconstruction.

1.1.3 District Six

District Six, located in Cape Town, South Africa, was one of very few mixed-race areas in the country during the Apartheid regime. However, in 1966, it was declared a white area under the Group areas Act of 1950. By 1982, the entire community had been uprooted, with sixty thousand people being forcibly removed and all of their homes destroyed.

Of what was once a diverse and vibrant community, all that now remains are a select few buildings (primarily churches) and a museum containing a number of artifacts and an archive of photographs that were taken before and during the demolition of the area. As a result, the photographs in this archive are all very old, and often of poor quality or in bad condition. Furthermore, little contextual information pertaining to the photographs is available.

1.2 Aims



(a) (b) (c)

Figure 1: The process of reconstructing a building. First, the user imports a number of photographs of the building (such as those in (a)), from which a model (illustrated in (b)) is reconstructed. This model is then textured to produce the final product, depicted in (c).

The aims of this project are best divided into two parts. The first being the development of a geometry reconstruction framework, and the second the implementation of a photo-based texturing scheme. Figure 1 illustrates the work flow from the reconstruction framework through to the photo-based texturing scheme.

1.2.1 Geometry Reconstruction Framework

The first goal of this project is to develop a framework that will allow a user to produce a three-dimensional model of an instance of District Six architecture from some given set of photographs, as shown in Figures 1(a) and 1(b). This framework should implement and, as necessary, modify the photogrammetric modelling technique described by Debevec [1996].

Ideally, the framework should be able to produce reconstructions in situations where there is only a single photograph available, as this is often the case when working with the District Six photographic archive. Furthermore, it is noted that a majority of the images in the District Six archive are uncalibrated (see Section2.1.3), and although the framework need not directly handle this problem, techniques to pre-determine the calibration parameters of the cameras should be investigated.

As output, the framework should provide a polygonal model of the reconstructed architectural structure that can be textured using information extracted from the original photographs.

1.2.2 Photo-based Texturing

The second aim of this project is to design and implement a texturing scheme that can be used to texture models acquired by the geometry reconstruction framework, as illustrated in Figure 1(c).

A primary requirement of the texturing scheme is that it use the original photographs to texture the model. Where only a single source photograph is available for a polygon, it should be projected correctly onto the model. Where multiple source images are available for a polygon, the most appropriate one should be selected. Finally, where no appropriate source images are available for a given polygon, a texture should be synthesised, based on the source material available to other polygons. Here, interactive techniques to achieve the most desirable results should be investigated.

A secondary requirement of the scheme is a high level of photo-realism. Ideally, renderings of the textured model should be of a near-photo quality. One consideration here, is that the quality of the output images will be dependent on the quality of the source photographs.

1.3 Thesis Structure

The remainder of this thesis is broken down into six chapters, as follows:

- Chapter 2 provides a detailed review of current digital reconstruction techniques, coupled with their applications in the field of heritage and architecture reconstruction. In addition to this, a brief survey of image-based texturing schemes and texture synthesis algorithms is presented. Finally, an analysis of the photographic material available from the District Six archive and a

discussion of the associated problems are also introduced.

- Chapter 3 provides an overview of the Geometry Reconstruction Framework. The details of the reconstruction algorithm used, as well as the underlying mathematics are discussed here. In addition to these, an overview of the User's view and functionality of the framework is presented.
- Chapter 4 discusses the underlying architecture of the Geometry Reconstruction Framework, and a number of the more crucial implementation aspects are covered in detail.
- Chapter 5 discusses, in detail, the photo-based texturing scheme that is used in the project, paying particular attention to how the problems associated with the lack of available material are handled.
- Chapter 6 presents reconstructions based on photographs from the District Six archive, along with a critical analysis of these results.
- Finally, Chapter 7 draws a brief conclusion to the thesis and presents some avenues for future work.

Chapter 2

Background

As mentioned in Chapter 1, constructing and rendering three-dimensional models of real-world objects is a crucial and widely used area of computer graphics. Many techniques have been developed to aid in the accurate reconstruction of such real-world objects and, in line with this, much effort has been invested in solving the problem of photo-realistic rendering of the resulting reconstructions.

One area of study which uses many of the reconstruction and texturing techniques is that of heritage reconstruction. Due to the variety of problems in this field, coupled with the divergent solutions that result, it provides an excellent proving ground for a multitude of these methods.

The remainder of this chapter presents background information on a number of digital reconstruction techniques, image-based rendering and texture synthesis schemes, and architecture and heritage reconstruction methods, as well as a discussion of the photographic material available in the District Six Museum archive.

2.1 Digital Reconstruction Techniques

Digital reconstruction methods can most easily be divided into three groups: CAD techniques, range scanning and photogrammetry. Each of these has advantages and disadvantages, as well as situations where they are effective or ineffective.

2.1.1 CAD-based reconstruction

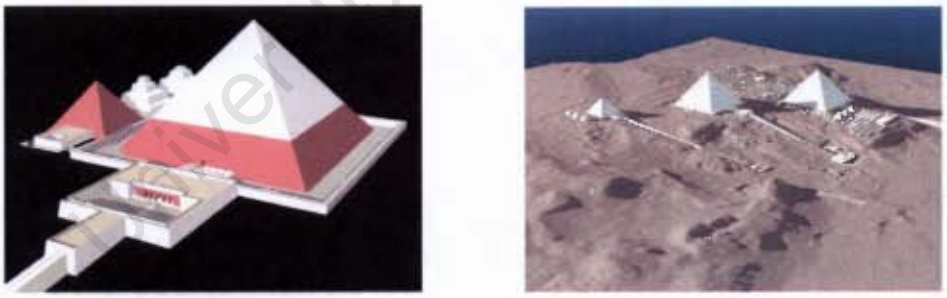
Broadly speaking, CAD (Computer Aided Design) involves using computer-based utilities to assist in the design of geometric structures. Today, CAD tools offer users wide-ranging functionality, from two-dimensional drafting applications through to three-dimensional solid and surface modellers.

The use of CAD tools is now ubiquitous across many fields including automotive engineering, construction, civil and building engineering and architecture. As a class of tools used to assist in the design and development of new objects, CAD is valuable. These computer-based systems provide massive advantages in terms of accuracy and speed over older methodologies where everything had to be hand drawn.

Despite being intended primarily for authoring new geometry, CAD tools may also be effective in the reconstruction of real-world objects. Where detailed, accurate information pertaining to the geometric structure of the object is available, it could be used to construct a three-dimensional model. Examples of where this approach may be used, include where blueprints for a building are accessible, or when accurate measurements of an object can be taken and used in the modelling process.

Due to the abundance of available CAD software, it is often possible to find the correct tool for most types of reconstruction. Furthermore, as there is an implicit high level of user interaction associated with these systems, they are generally very flexible and robust.

There are, however, a number of negative aspects associated with CAD-based reconstruction. The first problem is that it is very time consuming. For instance, the reconstruction of a reasonably complex building may take many days to complete. In addition, where it is impossible to obtain accurate data about the object at hand, this method becomes substantially less effective. Finally, in order to be at all effective, the user must have a high level of proficiency in the chosen CAD system and these systems are often extremely complex.



(a) (b)

Figure 2: Images from the Giza Plateau Mapping Project. (a) is the reconstructed model of a pyramid complex on the Giza Plateau, while (b) is a rendering of the plateau terrain, along with a number of the architectural scenes. Images used with the kind permission of the Giza Plateau Mapping Project, The Oriental Institute, University of Chicago.

An excellent example of the use of CAD-based reconstruction is the Giza Plateau Mapping Project at the University of Chicago [Lehner, 1992]. This endeavour, which has continued for many years, is dedicated to researching the geology and topography of the Giza plateau. Part of this project involved constructing a computer model of the Giza Plateau including the pyramids. To achieve this, a CAD-based reconstruction technique was used, and although the project was extremely time

consuming, the results, which can be seen in Figure 2, are quite compelling.

2.1.2 Range Scanning

A range scanner, or three-dimensional scanner, is a device used to collect geometric data about a real-world object, that can then be used in the construction of a digital model of that object.

Functionality

Typically, a range scanner is used to create a point cloud (a collection of points in space) of samples from the surface of the object at hand. As with a regular camera, a range scanner produces an image. However, instead of representing chromatic intensity, each point in the image stores the distance (or range) from the scanner to the object surface along the vector passing through that point on the image-plane. Using the position of the scanner as a reference point, one can then fully describe the three-dimensional position in space of each point on the range image, thus yielding the point cloud.

Technologies

According to Bernardini and Rushmeier [2002], the most commonly used class of range scanner is a triangulation scanner. In this technique, a light source, such as a laser, projects some pattern (for instance a single dot) onto the surface of the object that is to be scanned. A sensor, positioned some distance away from the source, is then used to detect the light reflected from the object.

The light source, sensor and the point where the light strikes the object surface now form a triangle. Through analysis of the location of the reflected light on the sensor's field of view, the angle at that corner can be determined. Combining this with the known distance between the camera and the source and angle at the source corner, yields the three-dimensional position of the visible point on the object surface.

One intrinsic limitation of this technique is that the point that is being scanned must be clearly visible from both the source and sensor positions. As a result, complex, self-occluding geometry may be difficult or impossible to scan. Beraldin et al. [1995] further suggests that triangulation systems are also susceptible to problems when the object's surface has discontinuities, but that these can be alleviated to some extent through the use of robust optical arrangements or multiple sensors. One final limitation of this method is that it is only really suitable for close-range scanning. Under the right circumstances, however, accuracy can be in the order of tens of micrometers.

A second class of range scanner is the time-of-flight laser scanner, which uses laser light to acquire data about the geometry of an object. These scanners determine the distance to a surface by timing

the round trip of a pulse of light. The scanner uses a laser to emit a pulse of light, and then measures the time elapsed before the light is detected by a sensor on the scanner. Given the speed of light, and the time taken for the round trip, it is simple to compute the distance to the surface.

The major problem associated with time-of-flight scanners is that the accuracy of the scan depends on the accuracy of the time measurement. As a result, these scanners are generally not suitable for close range scanning, but are excellent for scanning larger object, where a high degree of accuracy on the fine details is not requireds.

Registration

In most cases, it is insufficient to take just a single scan of an object, since from any given viewpoint, some amount of geometry is obscured. The solution to this problem is to take multiple scans of the object, and then bring the resulting point clouds into a common reference system from which a complete model can finally be constructed. This process of finding a common frame of reference is known as registration.

Some more sophisticated systems provide an initial registration by tracking the position and orientation of the scanner. In others, work is currently being undertaken to develop an automatic feature matching technique. In the most simple systems, user-intervention is required to provide this initial matching.

Regardless of what is used, the initial registration must be further refined through other techniques. One such technique, proposed by Besl and McKay [1992] involves using the ICP (Iterative Closest Point) Algorithm, which, given a reasonable starting point, can register two point clouds very quickly. Where more than two sets of points exist, the ICP algorithm has been extended to incrementally handle multiple point clouds. Examples of such techniques include work by Bergevin et al. [1996] and Benjemaa and Schmitt [1997].

In addition to merely using the cloud points for registration of views, methods of using textural information related to the scans in order to assist with registration have also been developed. Examples include the work of Roth [1999], where the textural information is used in the initial registration, and Gonzalez and Woods [2003], where the textures are used to assist the user in performing a manual registration.

Reconstruction

The final stage in the range scanning process is the reconstruction of the scanned surface from multiple, aligned scans (also known as scan integration). The task here is to unify all of the scans into a single, non-redundant surface.

Bernardini and Rushmeier [2002] provide an excellent literature survey of current techniques, in which they are split into four categories according to the size of problem being solved. These categories are Delaunay triangulation-based, surface-based, volumetric, and deformable surface methods. Each of these categories is suited to slightly different tasks.

Delaunay methods, for instance, offer a high degree of correctness, but are unsuitable for handling noise and are computationally exceptionally expensive. In contrast, volumetric methods are able to handle massive scenes, but poor choice of voxel size may either result in loss of fine detail (on the large side) or excessively large numbers of triangles (on the small side).

Analysis

Range scanning is a very powerful reconstruction technique. It is flexible, robust and affords the user the opportunity to scan arbitrary objects. Using the two technologies described above, it is possible to scan both very large objects (at lower levels of detail) and very small objects (with a high degree of accuracy). Work by Levoy et al. [2000], in which extremely detailed models of many of Michelangelo's statues are reconstructed, is an excellent showcase of the capacity of range scanning.

There are, however, also many drawbacks associated with this technique. The first of these is cost — good range scanning hardware is very expensive, and as a result can only be used in well funded projects. Second, scanning is a time consuming process, and also requires that the user have a high level of expertise in the field. A third problem is that this technique can only be used on extant objects. In our District Six scenario, it would be impossible to use. A final problem with range scanning is that it is an invasive technique. A large amount of equipment and many people are needed to effectively scan an object, and their presence is often unwanted and unwelcome at many sites.

2.1.3 Photogrammetry

In its broadest definition, photogrammetry is the science of taking reliable measurements from photographs. More specifically, it involves using the overlap of multiple photographs, combined with stereopsis (illustrated in Figure 4), to triangulate the positions of points in three-dimensional space, which can then be used in the reconstruction of real-world objects.

In typical photogrammetric schemes, it is first required that the intrinsic calibration parameters (e.g., focal length and principal point) of the cameras (or images) be determined. These parameters are used to determine the perspective projection that the camera approximates, as well as the extent to which it deviates from a true perspective projection.

Given a set of well calibrated photographs, the second stage is to determine the values of free variables representing the three-dimensional positions of a set of visible points, as well as the position and

orientation of each camera. This is done by first performing image registration to determine matching features over the set of images, and then using these correspondences to solve for the free variables.

Camera Calibration

In this context, given a photograph taken by a camera, calibration is the task of determining the intrinsic parameters of that camera. These parameters include the focal length, the principal point, the scale factor relating the two image axes, the skew between the two image axes and the distortions (radial or tangential) caused by the lens.

As most photogrammetric methods assume a true perspective projection, it is important to know the intrinsic parameters associated with each image with as high a degree of accuracy as possible. If one is unable to correct for distortions in the images, error will be introduced into the reconstruction. As it is such an important topic in computer vision, much time has been invested in developing techniques to solve the calibration problem [Abdel-Aziz and Karara, 1971; Zhang, 2000; van den Heuvel, 1999; Cipolla et al., 1999; Wilczkowiak et al., 2001].

One category of solutions involves using the camera in question to take photographs of a calibration target (illustrated in Figure 3), which are then used to solve for all of the intrinsic parameters. One of the first techniques used in this way is the Direct Linear Transformation (DLT), pioneered by Abdel-Aziz and Karara [1971]. In this method, numerous (typically eight or more), non-coplanar control points, with known real-world positions, must be photographed. Using these control points, up to sixteen parameters are solved for using the least-squares method. Eleven of these describe the relationship between the object-space and image-plane reference frames, while the last five account for various distortions.

Figure 3: A photograph of a typical camera calibration target.

Work by Heikkila and Silven [1997] uses the DLT along with non-linear parameter estimation to

both determine the calibration parameters and correct the distorted image coordinates. Here, compensations are made for both radial and tangential distortion.

A further example of such techniques is the work by Zhang [2000], which introduces another calibration-target based method, where a camera is required to observe a planar pattern from at least two positions. A closed-form solution is then used to find an initial estimate of the parameters, after which a non-linear refinement provides the final solution.

There are, however, numerous problems associated with this class of solution. The first, and most serious, being that one must be able to photograph the calibration target to be successful. Second, is that once the camera has been calibrated, none of the camera settings may change, as this modifies the intrinsic parameters. The final problem is that, although it is possible to use real-world objects (e.g., buildings) as the calibration target, the measurements of the object must be known exactly, and there must be enough visible control points.

More recently, methods have been developed to determine the calibration parameters of a camera from a single image, without the use of calibration targets [van den Heuvel, 1999; Cipolla et al., 1999; Wilczkowiak et al., 2001]. These are extremely useful for material such as the photographs in the District Six archive, where none of the calibration parameters for any of the photographs are known. This is, however, a very difficult problem to solve, and to date no solution exists that is effective in every scenario.

Despite this, some schemes do exist that make use of features present in many architectural scenes. van den Heuvel [1999] suggests that the intrinsic parameters of a camera can be determined from a single image, through the analysis of vanishing points of sets of parallel lines. In this system, straight lines are extracted using a line-growing algorithm and then grouped into sets of parallel lines. Thereafter, a two-stage algorithm is used to first determine the (radial) lens distortion using a least squares approximation, and then calculate the principal point and focal length using the adjusted observations from the first stage, along with constraints of perpendicularity and parallelism between the sets of lines.

In the same vein, Cipolla et al. [1999] present a method where analysis of vanishing points is used to determine the principal point and focal length for images of an architectural scene that are later used for a partial reconstruction of the scene (only visible areas are reconstructed). This method does not, however, account for radial distortion and assumes a known aspect ratio and orthogonality of the image-plane axes.

One final example of this category of calibration is work by Wilczkowiak et al. [2001], which uses parallelepipeds that are frequently found in architectural scenes. Here, they show, and make use of, the duality that exists between parallelepipeds and perspective cameras, in order to determine intrinsic parameters of the camera, including focal length, aspect ratio and principal point.

Feature Matching

In order to use stereopsis (illustrated in Figure 4) to gain information about the locations of points in space that are visible across the set of photographs, it is necessary to first locate these points, or features, in each of the images. In general photogrammetric reconstruction this is one of the most challenging phases. For complex surfaces, the problem of feature matching is often too time consuming — and frequently too difficult — to be performed manually. In addition, currently no automated technique exists that is able to solve this problem for arbitrarily complex surfaces.

However, one reasonably successful approach at general feature matching across a pair of images is introduced by Baumberg [2000]. This technique is able to detect some matching features corresponding to the same physical point on an object across images captured from arbitrary viewpoints (provided, of course, that there are features visible in both images).

This approach works first by detecting a set number of features of interest in each image. These features are detected in a scale-space, so that each feature has a good chance of being detected in both close-range and long-range images. For each of the features, affine invariants are calculated. These are characterisations of the feature that are less affected by local linear transformations such as rotation, skewing and stretching.

The matching process in this method involves comparing two vectors of these invariants to determine the similarity between any two interest points. As the process is geared towards use in structure-from-motion applications, the quantity of points is reduced to ensure that there are as few false-positives as possible. Although this does provide reliable data for estimating the camera poses, it may often be insufficient for the reconstruction of a complex surface.

Other matching methods involve using epipolar geometry to limit the search space for corresponding features in a pair of images. Epipolar geometry defines the relationship between a point on one image and a corresponding line on another. Using this relationship, the search space for a feature detected in one image is limited to a single line on another. For this to be effective, however, the orientations and positions of both cameras must first be established. Fortunately, this can be achieved by manually selecting a small set of corresponding features, after which, the rest can be automatically detected and matched.

Smit [1997] implements a technique which uses this idea in deep-level gold mines in South Africa. It is, however, noted that the objects being photographed must contain a reasonable level of contrast for the feature detection and matching to work effectively. In this case, the cameras are at a predetermined position and orientation, which does simplify the process. In order to find features of interest, a Sobel interest operator is run over an image, which locates such points by finding changes in the intensity gradient of the image.

When considering the reconstruction of more constrained objects — such as those commonly found in architectural scenes — the feature matching process becomes simpler. Where, for an arbitrarily

complex object a feature is a single point, for a building a feature may be an entire edge of the building. A further benefit of considering a more constrained set of objects is that manual feature selection becomes more viable, and often more sensible.

Debevec [1996], for instance, describes a method for the reconstruction of architectural scenes which requires the user to mark out corresponding features across the set of images that they are using. Here, the user is provided with a tool to mark out the visible edges of the buildings using lines. It is argued that user-based feature extraction and matching is suitable for this type of process because, as relatively few features need to be marked, the number of false-positives in the matching stage is reduced, and a very high degree of accuracy can be achieved.

Reconstruction

Given a set of calibrated photographs, with known corresponding features, the next stage in the photogrammetric process is to solve for the positions and orientations of the cameras, as well as the positions of the features. Intrinsic to all solutions to this problem is the property of stereopsis.

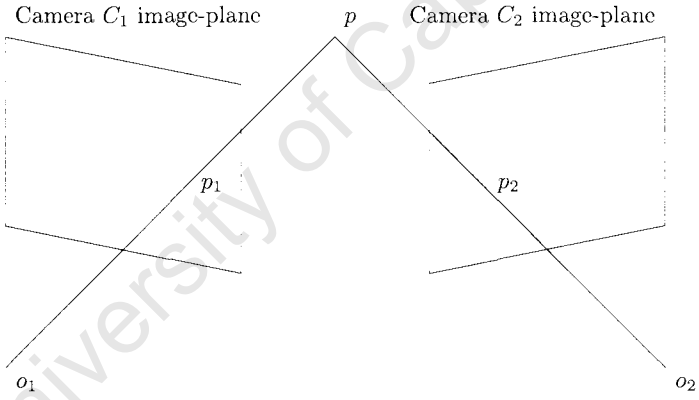


Figure 4: Stereopsis. This figure illustrates how the position in space of $\mathbf{p}$ can be determined by calculating the intersection of the line passing through o_1, the perspective centre of C_1, and p_1 and the line passing through o_2, the perspective centre of C_2, and p_2.

Consider Figure 4. Given the projection of a point, p_1, on the image-plane of a camera, C_1, the location in world coordinates of that point, say $\mathbf{p}$, can lie anywhere on the line p_1o_1 that runs from p_1 and passes through the perspective centre, o_1, of C_1. In other words, a single image of an object contains no depth information at all. If, however, there exists a point, p_2, on the image-plane of a second camera, C_2 (with perspective centre o_2), that corresponds (in world coordinates) to $\mathbf{p}$, then $\mathbf{p}$ must fall on the intersection of p_1o_1 and p_2o_2. Provided that the orientations and positions of $C1$ and $C2$ are known, it is trivial to compute the location of $\mathbf{p}$ (provided the lines intersect — real-world data may require an optimization). This can, of course, be extended to use more than two cameras, and when considering real-world situations this is desired as it generally gives a more accurate approximation of $\mathbf{p}$.

In the more general photogrammetric case, the parameters of the cameras are not known. This is problematic, as it introduces an extra six degrees of freedom, per camera (three for translation and three for orientation), to the system. As a result, in an unconstrained system it is impossible to solve for the position of a single point given multiple views. By finding many matching points of interest across the set of images, the system becomes implicitly constrained and solving for the free parameters of the cameras and points is possible. It is important to note that the number of parameters that must be solved for can be reduced by six (without loss of generality), by specifying the world coordinate system to be the reference frame of one camera.

With a large number of feature matches, the system does become sufficiently constrained for the parameters to be solved for, however, this is still a difficult task. Consider a situation with 100 features marked over three images, the result is over 300 variables that must be calculated. There are currently no solutions that are able to analytically solve for this number of variables, and as a result approximation techniques are most commonly used.

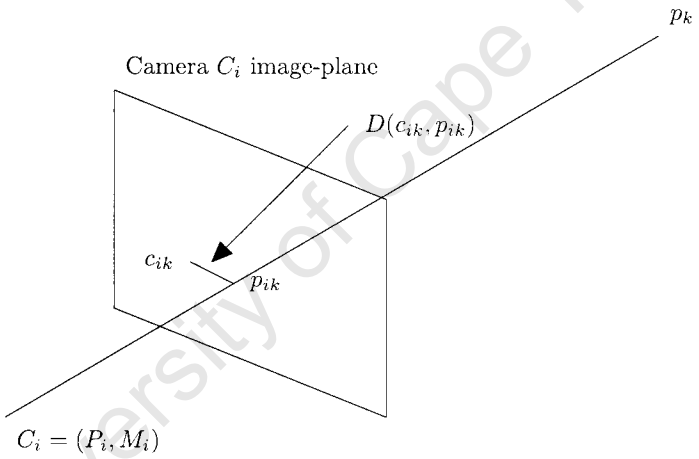


Figure 5: Camera projection. This figure illustrates how a point in space (p_k) is projected onto the image-plane of a camera (C_i), and how the error between the projected point and an observed point ($D(c_{ik}, p_{ik})$) can be measured.

Consider a camera, C_i, with a projection transformation matrix, P_i, and a camera transformation matrix, M_i, and a point in space, p_k (as illustrated in Figure 5). The coordinates of p_k, projected onto the image-plane of C_i, can be computed as $p_{ik} = P_i M_i p_k$. Further, let c_{ik} denote the observed position on the image-plane of C_i of the point in space p_k. One can then solve for the free parameters (M_i and p_k) by minimizing the objective function $\mathcal{O} = \sum_i \sum_k D(c_{ik}, p_{ik})^2$, where $D(a, b)$ is the Euclidean distance between a and b. It should be noted that this objective function is highly non-linear, and therefore suffers from the problem of local minima. This process of optimizing both the coordinates and orientations of the cameras and the coordinates of the features of interest is known, quite aptly, as a bundle adjustment.

This problem is presented in the above formulation for simplicity and clarity, but numerous alternative representations do exist, each with their own relative merits; Triggs et al. [2000] provide an

excellent survey of current bundle adjustment methods. Although in the above problem statement, each observation accounts for three parameters, it is possible to generalise the solution to optimize a parametrised model as opposed to a set of points (note that the parametrised model may, in fact, be a set of points). In this way, it is possible to greatly reduce the degrees of freedom if extra information is known about the object that is being reconstructed. In addition to this, there are numerous ways to model the error that must be minimised. Triggs et al. [2000] suggest non-linear least squares, robustified least squares, intensity-based methods and implicit models as good options for defining an error metric. These two factors afford the bundle adjustment solutions a high degree of flexibility.

Once a model and error metric have been decided upon, the parameters of the model must be optimized by minimizing the error function. There are currently many numerical optimization techniques available for this. Gill and Murray [1981], Nocedal and Wright [1999] and Fletcher [1987] provide in depth discussions and analyses of these methods. Some of the more commonly used methods are presented in Triggs et al. [2000] and these include techniques such as Newton's method, the Levenberg-Marquardt methods, the Gauss-Newton method and least squares optimization. Where it is necessary to minimize a function, subject to a set of constraints, Triggs et al. [2000] proposes using sequential quadratic programming.

[Shan et al., 2001] is one example of a bundle adjustment applied to photogrammetry, where, instead of their model parameters comprising a set of isolated points, they use a surface that is parametrised by relatively few variables. This is applied to the photogrammetric reconstruction of human faces, which are later textured from the original image. A comparison between using a parametrised surface and using a point-set model is drawn, and the success of the model-based parametrisation clearly illustrates the importance of structuring the problem so as to minimize the number of free parameters.

A further example is detailed in the work of Fitzgibbon and Zisserman [1998], where a structure from motion technique is implemented that uses a bundle adjustment to solve for the scene structure as well as the motion of a camera undergoing movement. Important here is that this technique is able to handle image sequences from an unknown camera undergoing unknown movement.

Analysis

When considering the reconstruction of real-world objects, current photogrammetry techniques provide a cost-effective alternative to range scanning. Although not as robust or flexible as many range scanning approaches, methods have been developed to handle many of the more common situations. Given the savings in both money and time that photogrammetry can afford, it is certainly a viable alternative to range scanning in many situations.

Even setting the cost issue aside, photogrammetry does have some decided advantages over range scanning. Although inferior in terms of quality of results (range scanning is still more accurate),

flexibility and robustness, photogrammetric techniques are more applicable in certain circumstances. Where range scanning often requires the presence of a lot of equipment and a significant amount of time, data for the photogrammetric techniques can be acquired in a few minutes with a small, hand-held camera. As a result photogrammetry is far less invasive and preferable in situations where the presence of a researcher might disrupt the local community.

In addition to this, when reconstructing objects that have been destroyed (as we will be doing with District Six architecture), range scanning techniques cannot be used. In this situation, again, photogrammetry becomes the most feasible option (provided, of course, that photographic records remain).

2.2 Photo-based Texturing and Rendering

Although it is possible to visualise a reconstructed model quite effectively by simply rendering it in a single colour with a reasonable lighting model, texturing can add a great amount of detail, thus improving the quality and realism of the rendering. Where actual photographs of the object are available, it is preferable to use these as the source of the textures. Provided all of the parameters of a photograph of the modelled object are known, it is possible to extract, for any point on the surface of the model, the texture coordinates for the photograph. This can be achieved through the use of projective texture mapping, a technique first introduced by Segal et al. [1992].

In situations where there are many photographs of the object, it may also be necessary to blend multiple views of a point on the surface of the object to give a final colour. Furthermore, depending on the viewpoint from which the model is being rendered, different weightings should be given to the input photographs (based on proximity to the synthetic viewpoint).

Debevec et al. [1998] describe an image-based texturing technique that implements a view-dependent texturing scheme able to handle multiple views of the reconstructed object. Only really effective when all the images of the object are taken under the same lighting conditions, this scheme blends the relevant images together, weighting them by how closely their view point matches the synthetic one. This approach also includes a hole filling strategy, where any polygons that are invisible from all cameras get assigned colour based on their neighbouring polygons. Although effective for minor holes, this is not ideal when, for instance, the texture of an entire face of a building is unknown.

One further example of photo-based texturing is the work of Oh et al. [2001]. Here, a single image is taken as input and then modified by the user to produce new views of the captured scene. Although a three-dimensional model is not explicitly computed, the scene can be rendered from novel viewpoints, provided they do not vary too greatly from the original. A number of tools (such as copying with re-projection that is distortion free) are provided to assist the user in texturing areas of the new view that were occluded in the original.

2.3 Texture Synthesis

Often it is desirable to texture a model even when there are no photographs available to extract textures from. In this case, the general approach is to generate a texture that has the required effect. The process of generating such textures is known as texture synthesis. Two major texture synthesis strategies exist, examples of which can be seen in Figure 6.

(a) (b)

Figure 6: Examples of synthesized textures. (a) contains a sample and a texture syntnesized from that sample using the scheme described by Efros and Leung [1999], while (b) is an example of a Perlin noise texture

Sample-based synthesis algorithms use a sample texture, from which a larger texture (such as that in Figure 6(a)) is generated. This category of synthesis is useful when generating realistic looking textures, as algorithms in this class often preserve local structure, resulting in a high degree of realism.

One example of such a system is described by Efros and Leung [1999]. This method grows a full image from a sample seed texture, by modelling the texture as a Markov random field, and adding a single pixel at a time, with a colour intensity determined by this probabilistic model. As a result this technique is well suited to solving the problem of filling in holes in textures. A further benefit is that it is able to handle a wide range of texture types.

A second sample-based synthesis procedure, detailed in Efros and Freeman [2001], and known as image quilting, works by joining together small blocks of the sample texture to form a larger texture. This system is very effective on semi-structured textures, where it is important to preserve whole features within the texture. One problem with this approach is that it is often possible to see repeating blocks within the texture, which can lessen the perceived realism. This is, however, only an issue with the more structured textures and is, in any event, difficult to entirely solve.

One further example of sample-based synthesis procedures is described by Wei and Levoy [2000]. This method works by taking a random noise field, and a sample texture patch, and then modifying the noise field so that it looks similar to the sample patch. As with the scheme proposed by Efros and Leung [1999], this technique models the texture as a Markov Random Field, and is able to

generate textures of similar quality. Key features of this procedure are the speed of the algorithm, and the guaranteed tileability of the generated textures.

The second class of texture synthesis algorithms involves the use of procedural methods to generate entirely new textures (an example of which is shown in Figure 6(b)). The underlying idea behind all methods in this class is having a procedure that takes, as input, a small set of simple parameters, and returns a complex output that is determined by the input parameters. Perhaps the most well known image synthesis technique in this category is Perlin noise [Perlin, 1985]. This method functions by generating a number of noise fields, each with different frequencies and amplitudes, and then adding them together to produce a final pattern. The result is a texture with a "random" high level structure (governed by the low frequency noise), that still has perturbations on a much smaller scale (produced by the high frequency noise).

This type of strategy is often used to generate entirely synthetic, yet still realistic looking textures. In situations where one has no access to sample material, one wants to generate textures that do not exist anywhere, or when memory is limited and numerous samples cannot be stored, these techniques can be very useful. Perlin noise has been successfully used in generating wood- and marble-like textures as well as water- or sky-like patterns [Perlin, 1985].

2.4 Heritage Reconstruction

As mentioned previously, Heritage Reconstruction is one of the major application areas for three-dimensional reconstruction techniques. The preservation of our cultural heritage has always been an important goal of society, and advances in computer graphics and modelling over recent years have provided many new avenues for realising this [Berndt and Carlos, 2000].

Organisations such as museums have a definite interest in digital reconstructions of heritage artifacts, as virtual exhibits augment physical displays in ways that make them far more attractive to visitors. Historians, archaeologists and artists are also able to make use of digital reconstructions, both in terms of preservation and when answering questions about objects that they study. For example, digital reconstructions offer researchers insight into questions about the creation and proportions of statues that they would otherwise not be able to answer [Berndt and Carlos, 2000].

In line with this demand, researchers have developed many techniques to facilitate the reconstruction of heritage sites and artifacts. Due to the divergent scenarios, however, no one technique is able to solve all of the problems. Where, for instance, smaller objects are being reconstructed, it is important to create highly detailed models. Where, alternatively, the subject of reconstruction is a larger scene, the problem of a high level of detail becomes eclipsed by that of modelling a massive scene. The material available for the reconstruction (Does the object still exist? Do we have access to it? Are there accurate measurements available?) and the requirements of the reconstruction (level of detail, texturing, lighting independence, etc.) augment the need for an array of different

reconstruction methods.

As a result, many methods of three-dimensional reconstruction have been applied in some way to the field of heritage reconstruction. As discussed in Section 2.1.1, Lehner [1992] uses CAD-based methods to assist in building a three-dimensional model of the Giza Plateau in Egypt. In this project, not only is the architecture of the plateau modelled, but also the terrain. The project makes use of maps, survey data and excavation reports to acquire sufficient information for the modelling process. The generation of the terrain model of the Giza Plateau was built using the *ARRIS* graphics program, after which the architectural complexes were modelled using *AutoCAD* software. In total, approximately nine months was required for the modelling process.

A further example, mentioned above, is the well known Digital Michelangelo project [Levoy et al., 2000]. The aim of this project (results of which can be seen in Figure 7) was to develop a framework for building digital models of real-world statues. The authors used laser range scanners to design accurate and highly detailed models of the selected statues. This approach involved spending many hours taking scans of the statues from different locations, after which point cloud registration algorithms were applied to generate a final model. The results of the project are highly-detailed, three-dimensional scans of numerous statues, with the largest scan having approximately two billion polygons. These models have since been used to render thousands of images of the statues in various synthetic conditions. Despite the excellent results, the scope of this method is limited in that it requires expensive equipment and can only be used where the objects still exist.

Figure 7: Two images of the head of Michelangelo's 'David'. The image on the left is a photograph, and the image on the right is a synthetic image, rendered from data acquired through range scanning performed as part of Stanford University's Digital Michelangelo Project. This image is used with permission from Professor Marc Levoy.

Another heritage reconstruction project is described by Gruen et al. [2002], and involved modelling two giant statues of Buddha that were originally situated in Bamiyan, Afghanistan before being destroyed by Taleban militia. Since the statues had been demolished, it was decided that a photogrammetry approach would be taken. Three different sets of photographs were used for the reconstruction - a set of photographs found on the Internet, a set of tourist images and a set of three metric images. A standard bundle adjustment was used to determine the parameters of the camera

and positions of the features of interest. Although results from this project are good, with the metric image set being used to create a point cloud of over 170,000 points, it is not an ideal approach for all situations. Where the structure being reconstructed is very regular, a point cloud is not an ideal representation, as the geometry can be modelled more accurately as a set of flat surfaces and regular curves, and a more constrained optimization technique should thus be used.

The above methods each make use of only a single reconstruction technique, however, in some cases a hybrid is employed for optimal results. One instance of this is detailed by El-Hakim et al. [2004], where both photogrammetry and range scanning are employed in their solutions. Their project is aimed at the reconstruction of human-made objects, where the structure is well constrained, and as a result uses a photogrammetry-based solution to determine basic shape. In a few select areas, where a high level of geometric detail is required, it is proposed that a range scanning technique should be used. This technique is extremely effective when used to reconstruct instances of architectural heritage, and was showcased through the modelling of the Abbey of Pomposa in Italy.

Finally, examples of heritage reconstruction also include work by Streilein and Nierderost [1998], Buehrer et al. [1999] and Beraldin et al. [2002]. The first describes the reconstruction of the Disentis monastery in Switzerland that was performed through the analysis of still video imagery, the second outlines the use of photogrammetry-based techniques to assist in the reconstruction of a rock-hewn church in Ethiopia and the third documents the reconstruction of a Byzantine crypt that was achieved through the use of laser scanning and high-resolution textural information.

2.5 Architecture Reconstruction Techniques

Although there is a degree of overlap between the reconstruction of heritage sites and that of architectural scenes, it is worth paying special attention to the latter, for two reasons. First, the focus of this project lies in the reconstruction of architectural scenes and second, the constraints inherent in most architecture have resulted in a number of interesting solutions.

One example of such an architectural reconstruction scheme is the work by Vosselman and Dijkman [2001], which describes a technique that constructs three-dimensional models of buildings using aerial point clouds and ground maps. Given a dense point cloud of an urban area, taken from a high altitude, this technique is able to reconstruct the buildings represented by that point cloud. To extract polygonal information from the point cloud, a three-dimensional variant of the well known Hough transform [Parker, 1996] is used. The parameters of the polygons are then determined using a least squares estimation, after which they are combined with the assistance of the ground plan to form a final model.

A number of photogrammetric methods have also been developed that are able exploit the constraints inherent in architectural scenes. Essentially, assumptions are made about the coplanarity of points and the parallelism or orthogonality of lines, that result in a far smaller number of free parameters.

One technique that makes use of such constraints is presented by van den Heuvel [2001], where a line-photogrammetric approach is used to model architectural objects from a single, uncalibrated photograph. This method first determines the intrinsic parameters of the camera, after which a least squares bundle adjustment is used to determine the model parameters. The results of this technique are illustrated through the partial reconstruction of a building from a single image of the Meydenbauer archives.

Two further photogrammetry-based techniques are described by Dick et al. [2000] and Dick et al. [2004]. The first technique automatically models architectural scenes from a small number of photographs, and is able to solve for the focal length of each image used. Again, this uses constraints inherent in architecture such as perpendicularity and verticality. Also introduced is a technique for evaluating the likelihood of a reconstructed model, which allows a rapid search through many such models to find the optimal one.

The second method processes multiple images of an architectural scene and, from those, produces a three-dimensional model of the scene, while identifying some architectural components. This system models a scene as a collection of walls, where each wall is a rectangle and contains a number of volumetric primitives that correspond to architectural features such as doors or windows. A probabilistic framework is then used to construct the model in terms of these primitives and extract identification information from the images.

One final technique, called photogrammetric modelling, is described by Debevec et al. [1996]. In this user-interactive method, a model that represents the scene is built using any number of parametrised primitives. Correspondences between the model and the images are then defined by the user, after which the parameters of both the models and all of the images used in the reconstruction are solved for. Through this modelling technique, constraints such as parallelism, orthogonality, coplanarity and symmetry are implicitly exploited. With such a highly constrained model, it is often possible to accurately reconstruct a building from only a single image. Furthermore, through the constraints afforded by the use of geometric primitives, a priori information can be used to accurately model geometry that is not visible in any of the photographs.

Given the robustness and flexibility of photogrammetric modelling, it was decided that our reconstruction framework would be based on it. Although there are newer methods, that provide a greater degree of automation, the speed of use was not a major concern. More important was the accurate reconstruction of the architecture and the ability to do this using the limited available material. It was also noted that a number of other techniques are also able to solve the problem of calibration. However, this problem can be — and often is — solved independently of the reconstruction, and should therefore not affect the decision of which reconstruction framework to use. Finally, the aim of the project was to design an application that could produce full three-dimensional models of buildings. Where photogrammetric modelling is able to do so through the use of symmetry and the geometric primitives, other methods often only produce partial reconstructions.

2.6 District Six Photographic Material

Located at the District Six Museum is an archive of photographs comprising many hundreds of pictures of District Six (some of which are illustrated in Figure 8), taken before or during the time when the area was destroyed. As a result, the photographs are all old, with even the most recent examples having been taken over thirty years ago. Furthermore, at the time, the images were not intended to act as an accurate photographic record of the architecture, and were thus not taken with this goal in mind.

As a result, when considering the reconstruction of architecture, there are many problems associated with this archive. These include:

- **Finding sets of matching photographs.** One of the more difficult problems is that of finding a set of photographs of a given building. Because there are so many images in the archive, and much of the architecture from the time looks so similar, the task of finding two photographs of any one building is both difficult and labour intensive. Furthermore, the photographs are not arranged in the archive according to which building they capture, but rather by other criteria such as who the photographer is, or when they were captured. This makes the already difficult task of locating sets of images even more laborious. Given this, it may, at times, be necessary to produce a reconstruction from only a single picture.

- **The time period over which the photographs were taken.** As the images were captured over a period of decades, they have a number of problems associated with them. First, the quality of photograph varies greatly. Using images of poor quality can only worsen the resulting reconstructions. Unfortunately, however, one often has no choice but to use those images. Second, if two photographs of a building were taken many years apart, changes may have occurred to the building in the intervening years. As some of the pictures were taken during the demolition of the area, this becomes a significant issue.

- **Insufficient meta-data.** A majority of the photographs in the archive are accompanied by little or no additional information. This includes a lack of information about subject matter, the camera used to take the picture and the date when the picture was taken. This only exacerbates the problems already associated with this archive.

- **Different chromatic schemes.** Most of the photographs were taken in greyscale. However, a number were also captured in colour or sepia. This presents problems when texturing the reconstructed buildings, as it is difficult to combine these schemes in a visually appealing way. Although converting all images to greyscale is one solution, it does result in the loss of data, and this is not necessarily ideal.

- **The digitization process.** All of the images in the archive were taken before the age of digital cameras. In order to use any image for the purpose of reconstruction, it must first be digitized. This is problematic as one cannot guarantee that noise or other artefacts will not

be introduced during this process. Minor rotations of the image while scanning, or slightly warped images will adversely affect the quality of the digital copy and this will increase any error in the reconstruction.

- **The problem of calibration.** It is not known with which camera and lens most of the photographs were taken and this immediately results in problems with calibration. This is exacerbated by the fact that many of the photographs have been cropped, which may make it difficult to approximate the principal point. Any additional error introduced by the digitization process will further complicate the calibration, making the results less reliable.

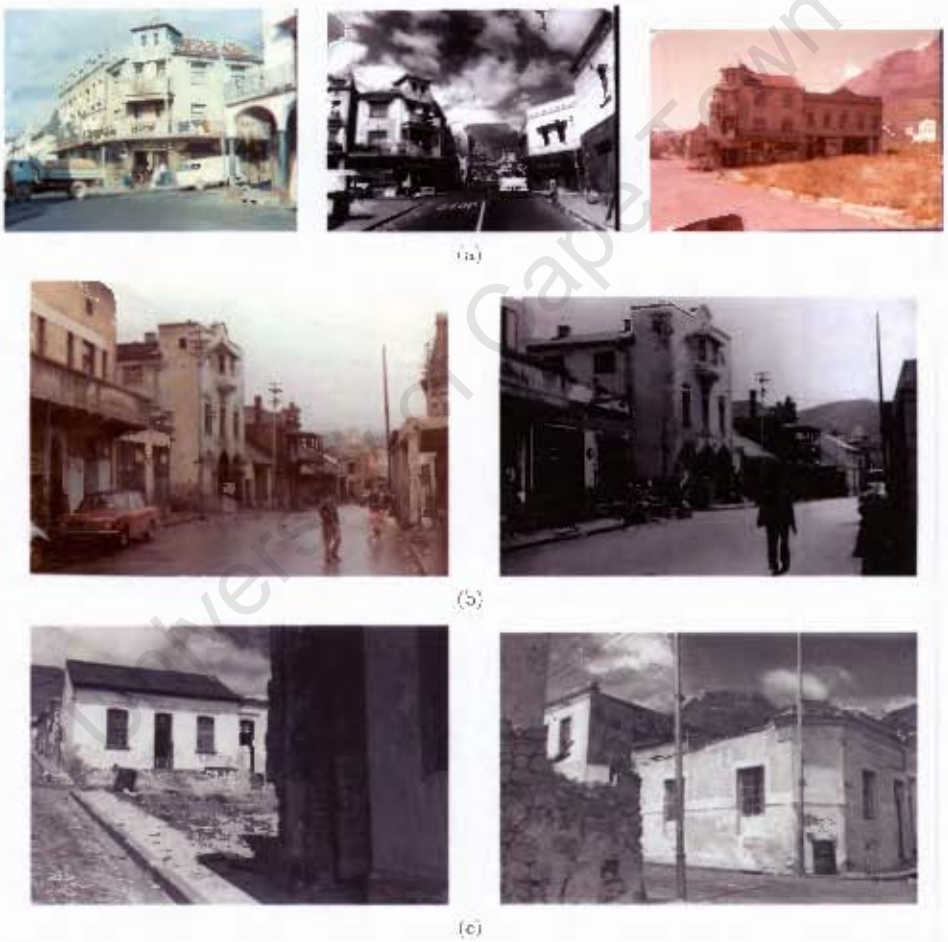


Figure 8: Some examples of the District Six photographic material. (a) illustrates three photographs of the Hanover Building. One of the photographs is in greyscale, while another was taken during the demolition of District Six. (b) contains two photographs of the British Bioscope — a building we go on to reconstruct. Again, one image is in greyscale, while the second is in colour. (c) illustrates two images of different District Six buildings. In each of these cases, we were unable to find additional, matching photographs.

Chapter 3

Reconstruction Overview

The Reconstruction Framework is the most important aspect of the District Six Virtual Reconstruction (DSVR) project. It is through the use of this component that it is possible to reconstruct models of District Six architectural scenes. The parameters of the corresponding camera views are also calculated, which allows for the extraction of textures at a later stage. The Reconstruction Framework implements our interpretation of the photogrammetric modelling technique described by Debevec et al. [1996].

The remainder of this chapter focuses first on the User's View, and then on the architecture of the framework. Topics discussed here include importing and calibrating photographs, modelling of scenes, drawing correspondences between the geometry and the photographs, and optimizing models.

3.1 The Model

Before continuing, it is necessary to describe, briefly, how any scene undergoing reconstruction is modelled. A scene — or model — comprises two principal components. The first is a collection of geometry and the second is a set of cameras. Both the geometry and the cameras are defined by a set of parameters and the model is, therefore, parametrised.

While the number of parameters introduced by the geometry is highly variable (depending on the constraints in place), each camera is modelled by six parameters. The first three determine the position in space of the camera, and the remaining three, its orientation. The field of view and image-plane size are static for each camera. As a result, every camera defines both a reference frame and a perspective projection.

Constraints can be defined both by restricting the freedom of the geometry and by defining relationships between the geometry and camera parameters. The parameters of the model can then

be optimized by minimizing the extent to which relationships defined between the geometry and camera parameters are violated.

3.2 The User's View

From the perspective of the user, the functionality of the framework can be partitioned into six actions. A brief description of each of these actions follows, and their use is illustrated by the accompanying flowchart in Figure 9. The underlying architecture behind each user interaction is then fully discussed in one of the subsequent sections. These actions are:

- **Calibrating and Importing Photographs.** Before any photograph can be used in the reconstruction framework, its calibration parameters (as discussed in Chapter 2) must be determined. Once these parameters have been calculated for all the photographs that will be used, they can then be imported into the system.
- **Geometric Primitive Design.** A tool has been created for the user to design any number of geometric primitives for use in the modelling process. It is important to note that once such a primitive has been created, it can be used in all subsequent reconstructions. As the primitive design forms an important part of the modelling process, it is discussed in detail under the section on modelling.
- **Modelling.** Using the set of pre-designed primitives, the user then builds a rough model of the architectural scene. It is not essential that the model have the same proportions as the actual scene, only that it is constructed from equivalent geometric primitives. It is while modelling that the user is able to add constraints to the system.
- **Defining Image-Model Correspondences.** Once at least a partial model has been designed, and a single photograph imported, the user is able to define image-model correspondences. These correspondences link user-marked edges (or observations) to edges in the model geometry, forming the basis for the objective function that is used to optimize the model.
- **Optimizing the model.** Again, once a partial model has been designed, and a single photograph imported, the user may instruct the system to optimize the model and camera parameters. As the non-linear optimization requires a longer time to run, the user can choose to use only the faster initial optimization for temporary results. It should be noted that if an under-constrained model is optimized, the results will be meaningless.
- **Texturing and Rendering.** Once satisfied with the model, the user may save it to file. Before the final, textured model can be displayed, the user needs to construct a synthetic texture for each face that does not appear in a photograph. Once this has been completed, the final, textured model can now be rendered. Although the texturing and rendering does not form part of the reconstruction framework, it is an essential part of the user's view and, as such, is introduced here. A full discussion is, however, deferred until Chapter 5.

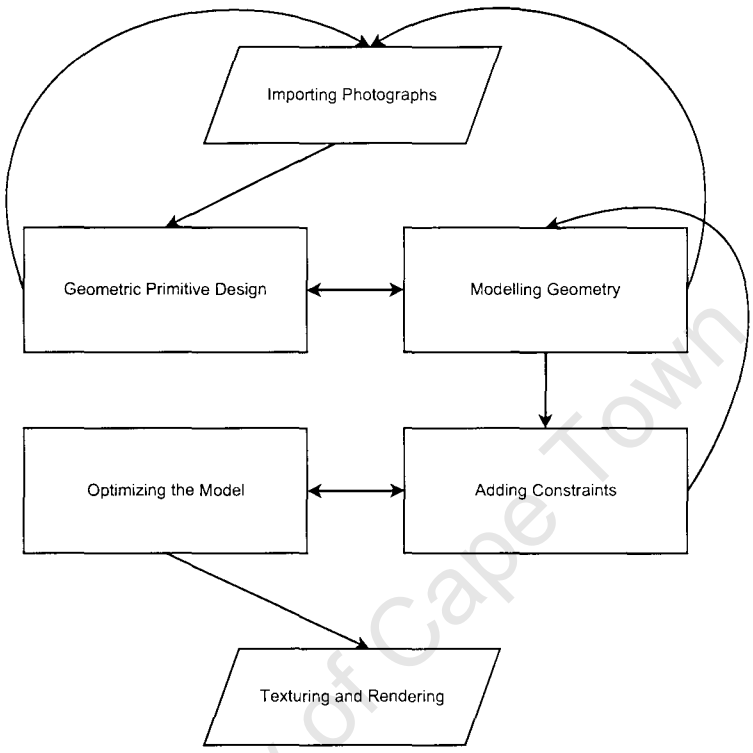


Figure 9: A flowchart describing the operations available to the user in the reconstruction framework.

Although most logically used in the above order, the first five of these actions can be performed several times, at any stage before the texturing and rendering is performed. This is important as it allows the user to add additional constraints depending on the success of the optimization, and also augment a semi-complete model with additional photographic material.

During the reconstruction phase, the user is presented with three windows. The first window is the Modelling Window, in which the current geometric model is displayed. Using this window the user can add primitives to the model, and select model edges for defining the image-model correspondences. The second window is the Primitive Construction Window, which assists the user in creating new geometric primitives. The final window is the Image Window, used for importing and displaying images, and for selecting image edges when defining the image-model correspondences.

3.3 Calibrating and Importing Photographs

As previously mentioned, before a photograph is used in the reconstruction framework, its calibration parameters must be determined by the user. Once this has been completed, the photograph may

be imported into the system, and used to define image-model correspondences as well as aid in the design of the model.

3.3.1 Photograph Calibration

As was made clear in Chapter 2, the available photographic material is generally of very poor quality, and entirely uncalibrated. In order to be effectively used in the reconstruction framework, it is necessary to determine the calibration parameters of the cameras used to take these photographs. Depending on the quality of the reconstruction the user hopes to achieve, different levels of calibration accuracy are required.

The absolute minimum requirements for any photograph are good estimations of the focal length and pixel size. The focal length is equivalent to the "zoom", or field of vision, of the camera. An incorrect focal length will result in distances along the z axis — observed as changes on the x or y axes — being calculated incorrectly, and this will result in an incorrect reconstruction. The pixel size is required as this is effectively the unit of measurement for the focal length, and is used in determining the field of view for the image. In general, however, as the reconstruction framework requires a field of view (rather than a focal length), an arbitrary pixel size can be assumed, and the focal length calculated relative to this.

Also of importance is an estimation of the principal point. This is necessary as all features on the image are marked out relative to this point. If the principal point is incorrect, so, too, are the positions of these features. Generally, the principal point can be assumed to be very near the centre of the image — and the reconstruction framework makes this assumption. However, in cases where extreme cropping has occurred it is important that the user compute the actual position of the principal point, and re-centre the photograph on a larger image, so that the principal point again lies in the centre.

The reconstruction framework also assumes that the aspect ratio between the pixel sizes along x and y axes for all images is 1 (i.e., all images have square pixels). For a majority of photographs this is the case, and the user need not worry. However, where this is not the case, the result will be incorrect scaling between measurements made on the x and y axes, which will adversely affect the reconstruction. The user should always re-scale such an image before importing it, to ensure that this aspect ratio is correct.

Finally, as the photographs were taken with older optical equipment, there is often a non-trivial amount of radial lens distortion. In the ideal case, this should be compensated for using some pre-processing techniques; however, if this is not possible the impact will not be as damaging as that of the other calibration problems, and will primarily be noticed in the form of minor reconstruction or texturing artefacts (as illustrated in Figure 30(a)).

Our Calibration Approach

For our example reconstructions, we elected to determine only the focal length of the cameras. This was possible, as when scanning in the original images, careful attention was paid to ensuring that a minimal amount of cropping occurred (leaving the principal point for each image in its centre). Furthermore, the results were sufficiently convincing that it was not deemed necessary to remove the radial distortion present in the images.

Computing an estimate of the focal length of an image becomes simple, provided at least one cuboid can be seen in the image. As discussed in Chapter 2, Wilczkowiak et al. [2001] show the duality between the intrinsic parameters of a camera and a parallelepiped. In fact, they argue that only when the calibration parameters of a camera are correct, will a parallelepiped in space match one that has been observed in the image.

Using this property, by repeatedly modelling (and optimizing) a single cuboid in an image, while varying the assumed focal length, a good estimate of the correct focal length can be obtained. Essentially, the focal length, for which the measured error is minimal after optimization of the model, is the most accurate estimate of the correct focal length.

3.3.2 Importing Photographs

Having determined the necessary calibration parameters for all of the photographs, the user is able to import them into the reconstruction framework using the Image Window. When importing an image, the user must enter the field of view along the y axis for the photograph, along with values for the translation and rotation (in Euler angles) of the camera. These need not be accurate, and can be changed at any time. We elect to input the field of view for the camera instead of the focal length, as this measurement is independent of pixel size; it simply describes the relationship between focal length and pixel size, and can be used later when normalizing the images (see Chapter 4).

It should be noted, that importing a photograph into the framework is equivalent to adding an additional "camera" to the model. As a result each photograph increases the number of free parameters in the model by six, but can also be used to define many constraints. In general, good results can be obtained with as few as two cameras, as illustrated in Figure 29 .

3.4 Modelling

In our system, the geometry of a scene is modelled as a collection of parametrised geometric primitives. In order for this to be successful, each primitive must correspond to an equivalent primitive in the real scene. Relationships are defined between primitives in order to reduce the number of parameters that must be optimized during the reconstruction. This section describes how the geometric

primitives are defined and how they are used in the modelling process, and provides a justification for using this approach.

3.4.1 Geometric Primitives

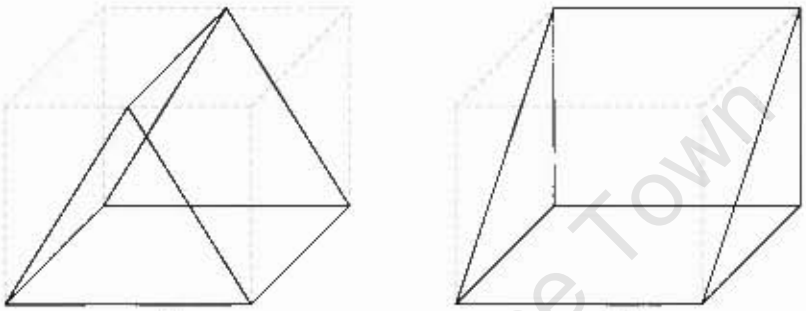


Figure 10: Sample geometric primitives. In 10(a), the two vertices that form the top bar of the prism can be computed as being the midpoint of the edge of the bounding box on which they lie. In 10(b), the vertices all fall on vertices of the bounding box.

In computer graphics, or constructive solid geometry, a geometric primitive is considered to be a simple, three-dimensional geometric shape [Foley et al., 1995]. Examples of these include cubes, cones, tetrahedrons and prisms. Following this definition, a parametrised geometric primitive is simply a geometric primitive that is defined by a set of parameters.

Consider a cuboid in space. It can be defined, relative to some frame of reference, by only nine parameters. Three describe its translation, three describe its orientation and the final three describe its dimensions (length, width and height). This provides one example of a parametrised geometric primitive. Although it is simple to parametrise a cuboid in this way, other primitives, are often described by different sets of parameters. Ideally, we do not want to have to consider each primitive individually, as this would be both time consuming and limiting, should a primitive that has not been defined be needed.

To avoid this problem, all primitives used in this framework are defined in terms of the same set of parameters. Consider again a cuboid in space. Further, suppose that this cuboid is interpreted as a bounding box. Any primitive that can be described in terms of the width, length and height of this bounding box is implicitly described in terms of the parameters of the cuboid. For a majority of primitives, defining the position of the vertices in terms of the extents of a bounding box is quite simple. Where the primitive becomes complex enough that this is no longer the case, it can (and probably should) be defined by two or more simpler primitives — each with their own bounding box.

In this framework, the user is presented with a tool that assists in defining arbitrary primitives in terms of the extents of a bounding box. A primitive is defined by first adding any number of vertices, and then joining these vertices to form polygonal faces. Each vertex is defined by the user as a function of one or more of the extents of the bounding box. In this way, as the shape of the bounding box changes, so does the shape of the primitive. When the primitive has been defined, it can be saved to an auxiliary file from which it can be retrieved for use in future reconstructions. The format of these files is simple, and advanced users may prefer to create new primitives by editing such a file directly. Some examples of these geometric primitives can be seen in Figures 10(a) and 10(b).

3.4.2 Geometry Model Structure

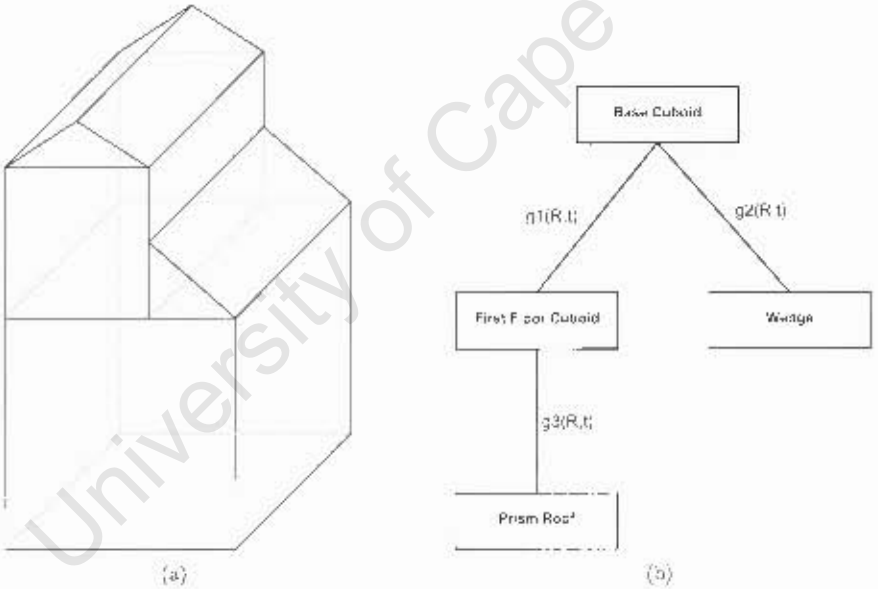


Figure 11: A typical model and scene graph. 11(a) illustrates how an architectural scene is built up of a number of geometric primitives. Here, two cuboids, one prism and a wedge are used. It is important to note that, in this case, the model can be constrained so that the prism at the top need only have one free parameter — its height. Similarly, the wedge is also parametrised by only its height. The base cuboid is parametrised by two parameters (width and height), as the third is set constant for scaling purposes. The first floor cuboid, again, is parametrised only by its height and width. 11(b) describes the hierarchical relationship structure of the model.

As mentioned earlier, the geometry of a scene is modelled using any number of previously defined primitives. In this fashion, even fairly complex architectural scenes can be represented with a high degree of accuracy. Since, in the most unconstrained case, each primitive would add an additional nine parameters to the system, it is possible to define constraints on any primitive, as well as

relationships between primitives that will reduce this number of parameters.

Six of the nine parameters for each primitive are used to describe that primitive's translation and orientation in space (represented by a translation vector, $\mathbf{t}$, and a rotation matrix, R). As a result, each primitive could be said to define its own frame of reference. In this modelling system, in order to reduce the number of free parameters, a hierarchical (tree-like) structure is used when describing the geometry. In this way, each primitive in the system defines its own frame of reference relative to that of its parent. Finally, the root primitive (which has no parent at all) defines the world coordinate system. This is a typical scene graph structure, widely used throughout computer graphics [Leler and Merry, 1996; Wernecke, 1994].

Consider a scene such as shown in 11(a).Let R_i be the rotation matrix, $\mathbf{t}_i$ the translation vector and $\mathbf{d}_i$ the dimension vector (representing width, height and depth) of the i^{th} primitive, P_i. Furthermore, let $g_i(R_i, \mathbf{t}_i)$ represent the rigid transformation from the reference frame of P_i to that of its parent. In this way the world coordinates of a vertex, $v_{mW}(\mathbf{d}_m)$, of the m^{th} primitive, P_m, can be computed as:

$$v_{mW}(\mathbf{d}_m) = g_1(R_1, \mathbf{t}_1) \cdots g_m(R_m, \mathbf{t}_m) v_m(d_m) \tag{1}$$

Following this same line of reasoning, the world orientation of a line segment, $l_m(\mathbf{d}_m)$, of P_m can be computed as:

$$l_{mW}(\mathbf{d}_m) = g_1(R_1, \mathbf{t}_1) \cdots g_m(R_m, \mathbf{t}_m) l_m(\mathbf{d}_m) \tag{2}$$

In most cases, the relationship between a primitive and its parent is, at least partially, known. It is very often the case that there is either no rotation at all, or a fixed rotation of 90 degrees around only one axis. Almost as frequently, the translations of a primitive are either zero or directly dependant on the dimensions of that primitive and its parent. In addition to this, some of the dimensions of one primitive can often be described in terms of those of a number of other primitives.

Being able to enforce constraints such as these is essential in reducing the number of free parameters that must be solved — an important feature, discussed fully in Section 3.4.4. In order to facilitate this, a number of options have been provided. By default, all parameters of all primitives are unconstrained. However, this may be altered in two ways. First, the user may fix the value of any parameter to a constant value. This option is most frequently used when defining the rotations between two primitives. The second option allows the user to set any scalar parameter of a primitive (i.e., elements in d_i or t_i) as a function of any other scalar parameters in the system. This feature allows the user to specify symmetry between primitives and can greatly assist in the reconstruction of invisible geometry.

3.4.3 Building a Model

Building a model in the reconstruction framework is a relatively simple procedure, and comprises three major steps. The first of these involves designing and importing geometric primitives, and has

already been discussed. The second and third steps are: adding primitives to the scene and setting constraints on the geometry, and are discussed in detail here.

Adding Primitives

When adding the first primitive to the scene, the user implicitly defines the world coordinate system. This is because any rotation or translation of this first block is equivalent to an inverse rotation or translation of the rest of the scene (including cameras). As a result, for the first block, no translation need be defined by the user. Similarly, in order to fix the scale of the scene, the user is required to specify the value for any one of the dimensions of the first primitive. If this were not the case, there would exist infinitely many solutions, each equally correct, and varying from the others only by a scale factor, and this would harm the robustness of the system,.

For all other primitives, the user must first select a parent primitive and then, once the new primitive has been chosen from those available, define a relationship between the new primitive and its parent. At this stage, the user may also specify initial values for any free parameters, as well as define initial constraints on the new primitive.

Setting Constraints

At any stage during the modelling process, the user may specify constraints on the geometry. This is done either through setting parameters of primitives to constant values, or by defining a parameter as a function of a number of other parameters. To perform these actions, the user selects the primitive of which they want to constrain a parameter, and then enters either a constant value or an expression in terms of other parameters into a dialogue box. Cyclical constraints (i.e., where parameter A references parameter B, and B references A) are not permitted in this model.

It is these relationships which result in a reduced number of free parameters in the model, and it is therefore preferable that as many as possible are added to the system.

3.4.4 Motivation

While this modelling strategy does have some limitations (see Section 7.2 for details), the motivation for using it is twofold. First, there are reasons why it is sensible for architectural scenes in general and, second, there are reasons why it is a logical choice specifically when considering the District Six material.

Architectural Scenes

Modelling an architectural scene using geometric primitives has a number of advantages over other schemes that use points, line segments, or polygons [Debevec et al., 1996], and these are summarised as follows:

- **The similarity to architectural structures.** Many architectural structures can be approximated very closely by a set of geometric primitives.
- **The constraints implicit in primitives.** The geometric primitives used in this system contain parallel and orthogonal lines. Since the primitives are polygonal, they also constrain sets of points to be coplanar. This is important as these constraints also exist in architectural scenes and should be maximally exploited.
- **More manageable modelling.** Using geometric primitives as opposed to individual polygons or line segments results in a far simpler modelling process, and this greatly reduces the modelling time and improves usability.
- **The simplicity of extracting surface and connectivity information.** Extracting both surface and polygon connectivity information from a model comprising a collection of geometric primitives is trivial, and this presents a decided advantage over systems which represent the geometry using points, lines and polygons as in these cases heuristic methods must often be used to solve this problem.
- **The reduced number of free parameters.** The most important advantage of modelling the geometry in terms of primitives and relationships is the massive reduction in the number of parameters that must be solved. If a line-modelling approach is used, each line in the model adds an additional 6 parameters. Every primitive contains at least six lines, but adds at most 9 parameters to the system. Where the relationships between primitives are well defined, even fewer parameters are needed — this can be seen in the example discussed in Section 3.7, where only 19 free parameters were used over the entire model. The result of this is that the system needs only a few correspondences to arrive at a robust solution.

District Six Scenario

Given the problems associated with material in the District Six photographic archive, this modelling technique offers a number of advantages specific to this type of situation:

- **The ability to reconstruct objects from only a single image.** Due to the constraints afforded by this primitive-based modelling system, it is often possible to reconstruct an architectural scene from only a single image. This is useful as it is often the case that only a single image of a building exists.

- **The ability to model hidden architecture.** Often, when working with examples of which only a few photographs exist, there is a fair amount of geometry that is hidden from all views. In line- or point-based techniques, it is extremely difficult to model this geometry correctly. However, using the geometric primitives and constraints of symmetry, additional knowledge that the user may have about the scene can be exploited to accurately define hidden geometry.
- **A high tolerance of image error.** Given that the geometric models built using this system are well constrained, a high degree of image error (as is often found in the photographs in the District Six archive), although still likely to introduce error into the model, is far less likely to distort it as badly as in less constrained systems.

3.5 Image Correspondences

In order to be able to measure the accuracy of the model (in terms of both the geometry and the cameras) correspondences between the images and the geometry must be defined. In this framework, line-based correspondences are used. In other words, edges of the real world object that are observed in the photographs are matched to edges of the model. When observed edges from different photographs are linked to the same model edge, the result is an implicit feature match. The task of defining these correspondences is separated into two actions; marking out edge features, and linking observed edges to geometry edges.

3.5.1 Marking Edge Features

The process of marking out the edge features for defining image-model correspondences is left as a user-interactive task. A tool is provided that allows the user to mark out edges of the object on each of the images. As it is not the end-points of the observed line segments that matter, but rather the gradient and position , the user is only required to mark out a short segment of each line. Longer segments do improve the accuracy of the observation, but where edges are obscured from vision by external features such as trees, or additional geometry, being able to use partial observations is extremely useful.

3.5.2 Adding Correspondences

Adding correspondences between user-defined image observations and model edges is a very straightforward procedure. A picking tool both for selecting observed edges and for selecting model edges is available. The user is simply required to select one of each, and then use the appropriate menu option to add a correspondence. Although a greater number of correspondences does result in a higher degree of accuracy and robustness, it also results in a slow-down in the optimization, and

often the change in accuracy is marginal. In accordance with this, the user should rather select fewer, more accurate observations than a large number of poor — or "short" — line segments.

3.5.3 Motivation

There are numerous benefits associated with using both line-based features and manual feature extraction. Using line-based features is a suitable strategy in this application, as it affords a much higher degree of accuracy than point-based features would. Trying to visually determine the exact pixel on which a vertex of the object lies is often very difficult and would result in the introduction of unnecessary error into the system. Visually, it is far easier to determine whether the position of a line segment is correct, and as a result this is a more stable approach. In addition, using vertices would require that they be visible in the images, whereas the line-based approach allows the user to mark out only small segments of the object edge.

Although a number of automated strategies do exist for detecting lines (such as those based on the well known Hough transform [Parker, 1996]), none is as good as the human visual system. In this system, only lines that form edges of the structure should be used, and lines introduced by shadows, features or changes in building material and texture can be very misleading to an automated system. Furthermore, as it requires additional knowledge to determine what makes a "good" feature, manual detection will generally achieve better results.

Requiring the user to define the image-model correspondences implicitly solves the problem of feature matching (edges in separate images matched to the same model edge are an implicit feature match). Although, again, there are automated techniques available for solving the matching problem (as discussed in Chapter 2), none is guaranteed to be error free. Furthermore, having to match the features both to the geometry and to other features makes this problem more difficult to solve.

Finally, with respect to the material available in the District Six photographic archive, the aforementioned problems are exacerbated. Where the image quality is poor both line detection and feature matching become more difficult, and in order to maintain a high degree of accuracy while performing these tasks, a user-interactive technique is most suitable.

3.6 Optimization

At any point during the modelling phase, provided there is some geometry, at least one image, and a sufficient number of correspondences, the user may elect to optimize the model. Essentially, this involves manipulating the free parameters of the model, until the geometry and cameras in the system conform to the observations. The optimization strategy, introduced by Taylor and Kriegman [1995], is performed in two stages. First, an initial estimate for all of the parameters is computed using a fast, simple optimization, after which a second, non-linear optimization is applied to the system to

compute the final values for each of the parameters. Both optimizations use a multivariate variant of Newton's method [Nocedal and Wright, 1999] to step iteratively closer to an optimal solution. Newton's method, the error metric used to measure the conformity of the model, and the underlying mathematics of both optimizations are discussed in detail in the following four sections.

3.6.1 Multivariate Newton's Method

Newton's method is an efficient, gradient-based, iterative algorithm for finding approximations of the roots of a real-based function. In this application, however, a variant of Newton's method suggested by Taylor and Kriegman [1995] is used, where, instead of calculating the roots, the intention is to find local (ideally global) minima of the function. In addition to this, a multivariate function is under consideration, and therefore a generalised Newton's Optimization method [Nocedal and Wright, 1999] is employed. Finally, since some of the free parameters are the rotation matrices of the cameras and primitives, in each iteration a local parametrisation of these rotation matrices, introduced by Taylor and Kriegman [1994], is performed to preserve the properties of these parameters. The details of this optimization strategy are presented below.

Multivariate Optimization

Consider a differentiable, real-valued function, $f(x)$. Given some arbitrary approximation of a root of $f(x)$, x_n, the Newton's method iterative step to reach a better approximation for that root is defined as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Given a reasonable initial estimate, x_0, this algorithm converges to a solution in very few steps.

In order to find a (local or global) minimum point of $f(x)$, we need only compute the corresponding root of $f'(x)$. Following this reasoning, given some arbitrary approximation of a minimum of $f(x)$, x_n^*, the Newton's method iterative step to reach a better approximation of that minimum is defined as

$$x_{n+1}^* = x_n^* - \frac{f'(x_n^*)}{f''(x_n^*)} \tag{3}$$

As with the root finding approximation, very few steps are required for convergence; however, when trying to minimize non-linear functions, it is essential to have a good initial estimate to prevent the solution converging to a local minimum.

Finally, consider a differentiable, real-valued function with m parameters, $f(\mathbf{x})$. Given an arbitrary approximation of any minimum point of $f(\mathbf{x})$, $\mathbf{x}_n$, we can generalise equation (3) to yield the Newton's method iterative step to reach a better approximation of that minimum, defined as

$$\mathbf{x}_{n+1} = \mathbf{x}_n - H^{-1}\nabla f(\mathbf{x}_n) \tag{4}$$

Where $\nabla f(\mathbf{x})$ is the vector of all first order partial derivatives of $f(\mathbf{x})$ and H (known as the Hessian matrix) is the m by m matrix of all second order partial derivatives of $f(\mathbf{x})$.

As calculating H^{-1} is a very slow process, equation (4) is often posed as a system of linear equations

$$H\mathbf{x}_{n+1} = H\mathbf{x}_n - \nabla f(\mathbf{x}_n)$$

to which a solution can be found very rapidly using a linear solver, such as that available in the *GNU Scientific Library* [GSLDocs, 2007].

Minimization on the Lie Group $SO(3)$

Optimizations over the set of rigid rotations, $SO(3) = \{R \in \mathbb{R}^{3\times3} : R^T R = I, det(R) = 1\}$ are problematic, as in most numerical optimization paradigms the free parameters are assumed to lie in a vector space isomorphic to $\mathbb{R}^n$. The Lie group, $SO(3)$ is not, however, isomorphic to $\mathbb{R}^3$ and as a result global parametrisations (such as Euler angles) will always exhibit anomalies at some points in the parameter space [Taylor and Kriegman, 1994].

The solution to this problem, proposed by Taylor and Kriegman [1994], is to use an atlas of local parametrisations instead of a single, global parametrisation. Essentially, for every iteration of the optimization, a mapping between between $SO(3)$ and $\mathbb{R}^3$ is found that is valid for some neighbourhood around the current rotation matrix, R_0.

3.6.2 The Error Metric

Before describing the initial and non-linear optimizations, it is necessary to introduce the error metric that is used to measure the disparity between the model and the observations. The optimization uses an error metric first introduced by Taylor and Kriegman [1995], which is discussed below.

Consider a straight line L_i of the model projected onto the image-plane of a camera C_j, that is described by translation vector $\mathbf{t}_j$ and rotation matrix R_j, as illustrated by Fig. 12(a). This model line can be defined by a point in space, $\mathbf{d}_i$ and a direction vector, $\mathbf{v}_i$ (that can be computed using Equations 1 and 2, respectively). The projection of this line onto the image-plane of C_j can be computed by finding the intersection of the image-plane with the plane defined by the centre of C_j and any two points on L_i. Assuming a unit focal length, $f = 1$ (with the image-plane located at $z = f = 1$), this line can be defined by $\mathbf{m} \cdot (x, y, 1)^T = 0$ where $\mathbf{m} = (m_x, m_y, m_z)^T$ is the normal to the plane defined by C_j and L_i, and can be calculated as

$$\mathbf{m} = R_j(\mathbf{v}_i \times (\mathbf{d}_i - \mathbf{t_j})) \tag{5}$$

Further, consider an observed line segment on the image-plane of C_j that is defined by the endpoints $\{(x_1, y_1), (x_2, y_2)\}$, and the projection onto that image-plane of L_i, as illustrated in Fig. 12(b). The

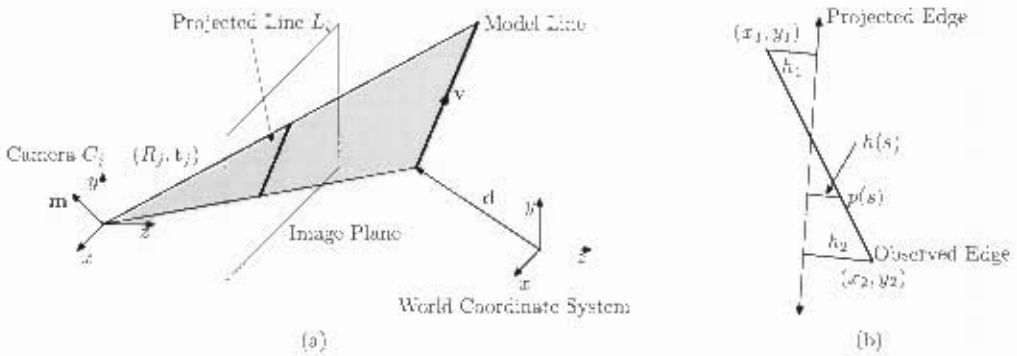


Figure 12: Projecting a model line onto the image-plane of a camera. 12(a) illustrates how a model line is projected onto the image-plane. Note $\mathbf{m}$, the normal to the plane defined by the model line and the centre of the camera. 12(b) illustrates the error between a projected line and its corresponding observation. Here, h_1 is the distance between (x_1, y_1) and the projected line and h_2 is the distance between (x_2, y_2) and the projected line.

observed edge of length l can be parametrised by a scalar variable, $s \in [0, l]$. Let $p(s)$ denote a point on the observed segment and let $h(s)$ be a function that determines the shortest distance from $p(s)$ to the projected line. Taylor and Kriegman [1995] show that the error between the projected model line L_i and observed segment on the image-plane of C_j can be computed as

$$Err_{ij} = \int_0^l h^2(s)ds = \frac{l}{3}(h_1^2 + h_1 h_2 + h_2^2) = \mathbf{m}^T(A^T B A)\mathbf{m} \tag{6}$$

with

$$A = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{pmatrix}$$

$$B = \frac{l}{3(m_x^2 - m_y^2)} \begin{pmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix}$$

It is important to note that this error metric gives each observed edge a weight proportional to its length, which is favourable since the longer edges implicitly have a higher degree of accuracy.

3.6.3 The Initial Estimate

As the non-linear optimization routine suffers from the problem of local minima, it is essential that before it is run, a good initial estimate is computed. Taylor and Kriegman [1995] proposed a solution whereby two initial procedures are used, of which the first estimates the camera rotations and the second estimates the camera translations and model parameters. These initial optimizations are performed separately to alleviate the problem of non-linearity in the error metric. An explanation of these procedures follows.

Taylor and Kriegman [1995] show that on further examination of Equation 5 two implicit constraints become apparent:

$$\mathbf{m}^T R_j \mathbf{v}_i = 0 \tag{7}$$

$$\mathbf{m}^T R_j (\mathbf{d}_i - \mathbf{t}_j) = 0 \tag{8}$$

The geometric interpretation of Equation 7 is that the vectors $\mathbf{m}$ and $\mathbf{v}'_i = R_j \mathbf{v_i}$, are orthogonal. When the parameters of the model are correct, this must always be true as $\mathbf{v}'_i$ lies on the plane to which $\mathbf{m}$ is normal. Similarly, the interpretation of Equation 8 is that, when the model parameters are correct, the vectors $\mathbf{m}$ and $\mathbf{d}'_i = R_j(\mathbf{d}_i - \mathbf{t}_j)$ must, also, be orthogonal.

On further examination, one notes that when given an observed edge, defined by the endpoints $\{(x_1, y_1), (x_2, y_2)\}$, and assuming a unit focal length, $\mathbf{m}'$, the observed normal to the plane defined by the camera centre and the observed edge is defined by:

$$\mathbf{m}' = \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} \tag{9}$$

By replacing $\mathbf{m}$ with $\mathbf{m}'$ in Equation 7, it becomes apparent that any model edges of known orientation act as constraints on R_j. As architectural scenes usually contain a number of horizontal or vertical edges, it is generally possible to determine an estimate for R_j independent of any other parameters in the model. Minimizing the objective function

$$\mathcal{O}_1 = \sum_i (\mathbf{m}'^T R_j \mathbf{v}_i)^2$$

over only the camera rotation matrix R_j limits the extent to which that matrix violates the constraint expressed by Equation 7.

Once the estimates of the camera rotation matrices have been computed, they can be treated as constants, and the constraint expressed by Equation 8 can be used to obtain estimates of the rest of the model parameters. Where P_i and Q_i are expressions for the vertices of model edge i, minimizing the objective function

$$\mathcal{O}_2 = \sum_j \sum_i (\mathbf{m}^T R_j (P_i - t_j))^2 + (\mathbf{m}^T R_j (Q_i - t_j))^2$$

over all parameters of the model apart from the camera rotation matrices, results in good initial estimates for those parameters.

Both of the initial estimation objective functions can be optimized efficiently using the Newton's method variant described above, and can be run at any stage by the user to quickly compute a reasonably accurate model. When the initial estimates of the parameters have been computed, the non-linear refinement can be used to calculate their final values.

3.6.4 The Non-linear Optimization

Where the two initial estimation procedures ($\mathcal{O}_1$ and $\mathcal{O}_2$) computed isolated sets of parameters for speed and simplicity, the non-linear objective function is optimized over the entire parameter set to ensure precision. In this way, any error is "spread" over all of the parameters, and the result is a substantially more accurate model. Recalling the error metric presented in Equation 6, the non-linear objective function can be defined as

$$\mathcal{O} = \sum_{i} \sum_{j} Err_{ij}$$

Again the optimization strategy presented above is used to minimize $\mathcal{O}$, and generally converges to a solution within ten iterations.

3.7 Real-world Example

(a) (b)

Figure 13: Two photographs of a large District Six residence. The relevant edges of these photographs have been marked out in red by the user.

To illustrate the functionality of the framework, consider the two images of a large District Six residence, shown in Figure 13. Using only these two photographs, a model of this house is accurately reconstructed, and then later textured. In these two photographs, the edges of the building have been marked out in red by the user, and these lines are later linked to the model in order to define the objective function (Section 3.5). Note that one image presents a very close-up view of only a portion of the building, while the other presents a view of the entire building, but from a great distance away.

The geometry of the building is modelled as a cuboid for the base, a prism for the roof, a cuboid

for the chimney, and a cuboid and a prism for each of the gables (Section 3.4). Figure 14 shows the scene model and the hierarchy of the primitives that comprise the model. The root node in the scene hierarchy is the base cuboid. Its children are the roof prism, the chimney and the two cuboids used in modelling the gables. Each of these cuboids has a further child — a prism, that completes the gables.

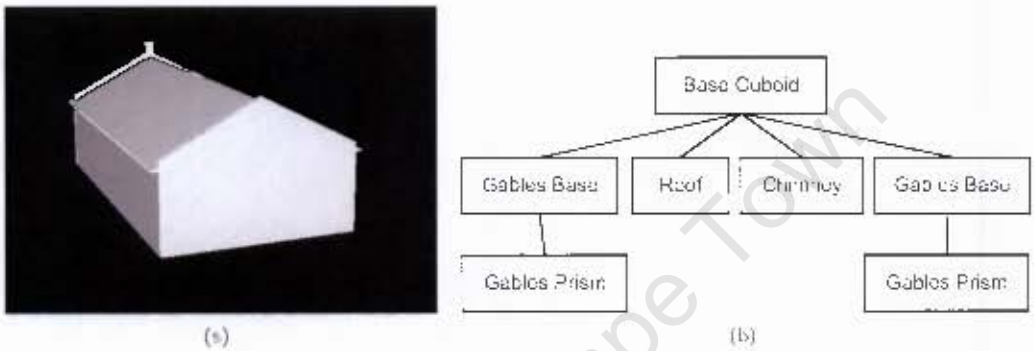


Figure 14: The model used in the reconstruction of the large District Six residence and its associated scene graph. 14(a) is the model that was used to reconstruct the building. 14(b) illustrates the primitive hierarchy of the primitives used in the model.

Figure 15: The two input photographs of the large residence with the reconstructed model projected onto them. The projected lines match the marked lines and building edges very accurately.

Including the two cameras (which contribute six parameters each), the total number of free parameters used to model the building is 19. Two are contributed by the base cuboid — for its depth and height — two by the roof prism — again for depth and height — and three by the gables — one for the height of the base cuboid, one for the height of the prism, and one for the width of the gables.

Since the two gables are identical, both share the same parameters.

The reconstruction of the model was successful, matching closely to the user observations. Figure 15 illustrates the model projected onto the original photographs, from the perspectives of the reconstructed cameras. The projected lines match up very closely to the building edges and the marked lines; however, in the second image, the base of the model appears to be lower than the base of the building. This problem is due to the fact that in the second photo it was impossible to mark out where the base of the building joined the ground, and thus this was not explicitly taken into account during the optimization. The general problem of foundations is discussed further in Chapter 6. Irrespective of this, the reconstruction was otherwise very accurate.

Chapter 4

Reconstruction Implementation

4.1 System Overview

The reconstruction framework was developed in the *C++* programming language, within an object oriented paradigm, and makes use of three third party libraries — *OpenGL*, *wxWidgets* and *The GNU Scientific Library* [GSLDocs, 2007; OpenGLDocs, 2007; wxDocs, 2007]. The structure of the framework comprises two major components — the *Scene Modeller* and the *Optimizer* — and a brief outline of each of their structures is depicted in Fig. 16.

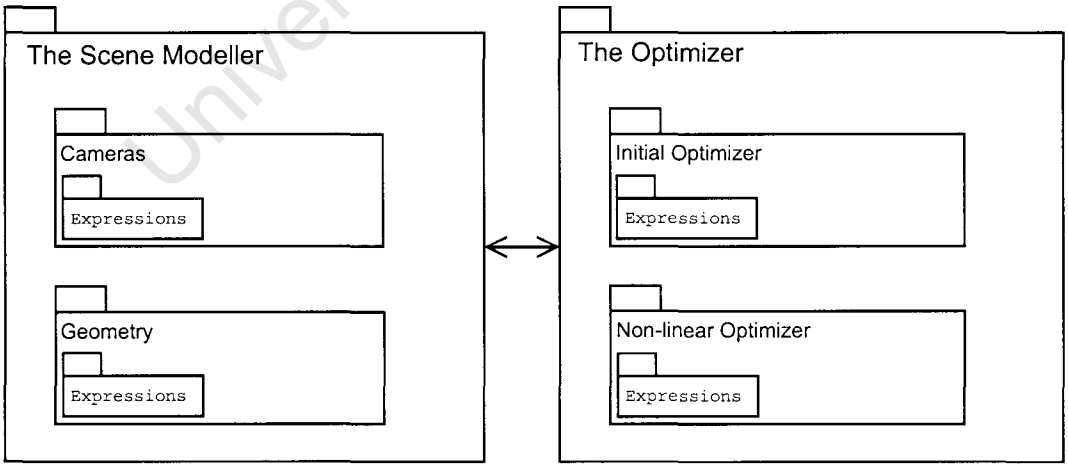


Figure 16: The reconstruction framework architecture. This diagram illustrates the two major components of the framework, along with their respective functionality. The Optimizer is used to fit the modelled scene to the observed image-edges.

The purpose of the *Scene Modeller*, as the name suggests, is to represent the model that the user

wishes to reconstruct. It is this component that stores all information pertaining to the cameras and geometry of the model, along with any constraints on the model that may exist.

The second major component of the framework, the *Optimizer*, interfaces with the *Scene Modeller* and is used to optimize the modelled geometry and cameras according to the observations that have been made. It is in this component that the optimization techniques discussed in Chapter 3 have been implemented.

A detailed discussion of both of these components, focusing on a number of the more important implementation aspects, appears in the following two sections.

4.2 The Scene Modeller

As noted above, the Scene Modeller represents the model that the user wishes to reconstruct. The most important points to consider here, are the representations of the cameras and geometry and the handling of user-imposed constraints on the model. Before these can be explored, however, it is crucial to introduce and discuss the concept of an Expression — an underlying component, ubiquitous throughout the framework's architecture.

4.2.1 Expressions

As explained in Chapter 3, the use of parametrised primitives — as well as parametrised cameras — is fundamental to the functionality of the framework. In this architecture, we consider all of the parameters of the primitives and cameras as Expressions.

In this context, an Expression can be a single scalar, a vector of dimension three, or a 3×3 matrix. Both the vector and matrix Expressions use arrays of scalar Expressions to represent their elements. Each Expression has a name associated with it, and where the Expression is a vector or matrix, the elements of that Expression can be referenced by appending a suffix to the Expression name. Although a matrix Expression has nine elements, since it is treated as a rotation matrix, it in fact only adds an additional three parameters to the system. For the purposes of the optimization, these three parameters can be accessed by appending a different suffix to the name of the Expression.

Every Expression is equal to either a numerical value or a combination of an arbitrary number of other Expressions. The operators available when defining these combinations are scalar addition, subtraction and multiplication, vector addition, subtraction and cross product, matrix addition, subtraction and multiplication, matrix-vector multiplication and finally, the equality operator. Expressions that are equal to a numerical value are set to be either constants or variables. A variable Expression generally corresponds to a free parameter of the model and is modified during the optimization, whereas the value of a constant Expression is never modified during this process.

The Expressions are implemented using a *Binary Tree* data structure, as illustrated in Figure 17. Each internal node of the tree is an operator that returns one of the three types of Expressions (scalar, vector or matrix), and takes, as operands, two valid Expressions (a matrix-vector multiplication operator, for instance, must take a matrix Expression as the left operand and a vector Expression as the right operand). Every leaf node in the tree is an Expression that is equal to a numerical (variable or constant) value. It is important to note that in this architecture, it is simple to combine two Expressions by modifying a leaf node of the one to point to the root (or even an internal) node of the other.

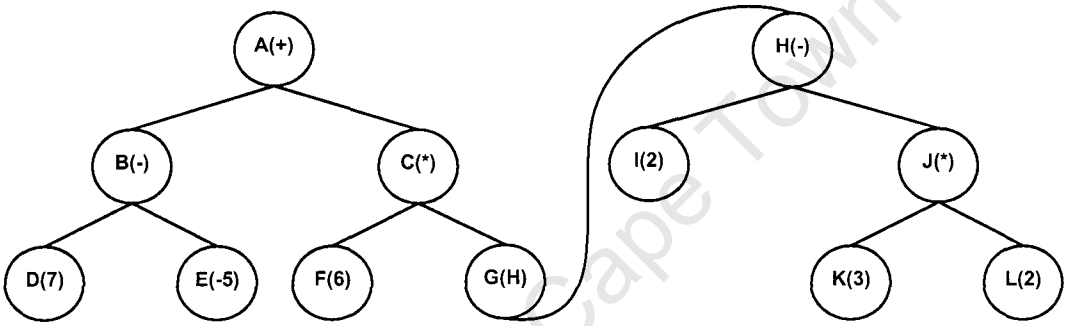


Figure 17: A typical Expression tree. Here, all nodes of the tree return scalar values. Note that the Expression tree with root node **H** forms a sub-tree of the Expression tree with root node **A** at node **G**. One correct order of node evaluation for these trees is in reverse alphabetical order (i.e., node **L** to node **A**).

Using this implementation, an Expression can be efficiently evaluated by recursing down its tree. Here, the value of each internal node is calculated only when both of its children have been evaluated. With this method, the final node calculated will always be the root node. An example of the order in which nodes of an Expression tree are evaluated can also be seen in Figure 17.

4.2.2 Cameras

The cameras in the framework represent photographs that are imported by the user. The position in space of each camera is defined by a matrix Expression and a vector Expression that represent its orientation and translation, respectively. Typically, these Expressions store a numerical value directly, and are not a combination of any other Expressions. In addition to this, each camera stores textural information from its respective photograph, as well as a set of observations.

Since the error metric described in Chapter 3 requires that all photographs have a unit focal length, a mapping between the actual image-space and the image-space with image-plane located at $z = 1$ must be determined. We call the process of calculating this scale factor *Image Normalization*.

Thereafter, user-defined observations may be added to the camera, and the positions of these are stored in the normalized image-space.

Image Normalization

When a user imports a photograph into the framework, they are required to specify a value (in degrees) for the field of view along the x axis, fov_x. Since the image has a known aspect ratio, $aspect = \frac{width}{height}$, and pixels are assumed to be square, it is possible to calculate a scale factor that transforms coordinates on the original image-plane to equivalent coordinates on the image-plane, located at $z = 1$.

To do this, it is first necessary to calculate the focal length of the original image. Assuming that each pixel has a length of 1, using the *Sin rule*, the focal length of the original image, $f_{original}$, can be calculated as

$$f_{original} = \frac{\sin(90 - \frac{fov_x}{2})\frac{width}{2}}{\sin(\frac{fov_x}{2})}$$

Having determined $f_{original}$, a point $\mathbf{p} = (x, y)$ on the original image-plane can be trivially transformed to an equivalent point $\mathbf{p}' = (x', y')$ on the new image-plane located at $z = 1$ with

$$\mathbf{p}' = \frac{1}{f_{original}}\mathbf{p}$$

This transformation is, of course, an invertible one-to-one mapping between the two image-planes and the inverse can thus be used to transform coordinates from the new image-plane back to the original one.

Adding Observations

When a user marks out a line segment on an image, the line is stored as a pair of end points, $\mathbf{s}$ and $\mathbf{e}$. These points are first transformed to be centred around the middle of the image, and then the transformation described above is applied, resulting in the final coordinates of the equivalent point on the image-plane at $z = 1$.

4.2.3 Geometry

As stated in Chapter 3, the geometry of a scene is modelled as a collection of parametrised geometric primitives. Internally, the geometry is stored as a scene graph (technically a scene tree, as it is always acyclic) of geometric primitives. Implementation aspects of both the geometric primitives and the scene graph are discussed below.

Parametrised Geometric Primitives

The geometric primitives are implemented with extensive use of the Expressions described above. Each primitive comprises nine parameters, which are represented by seven Expressions — three scalar Expressions for each of the translation and dimension vectors and one matrix Expression to describe the orientation of the primitive.

Scalar Expressions (as opposed to vector Expressions) are used to represent the translation and dimension vectors, as single elements of these vectors are often accessed independently of the others, and the scalar Expressions provide fewer layers of abstraction than vector Expressions would. As with the cameras, due to the local parametrisation in use (discussed in Chapter 3), the matrix Expression introduces only three additional parameters to the system.

Unlike with Expressions used to represent the cameras, those used in the geometric primitives may be — and often are — equal to a combination of other Expressions. Indeed, it is this feature that makes it possible to so effectively reduce the number of free parameters in the system. In addition to this, any of these Expressions may be set to a constant numerical value, and this too aids in the reduction of the number of free parameters.

The vertices (and hence edges and faces) of a primitive are defined in terms of the elements of the dimension vector of that primitive, d_x, d_y and d_z. Here, each vertex is represented by a vector Expression, of which each element can be set equal to either a combination of those elements or a numerical value. The edges of the primitive are defined by a set of vertex pairs, and the faces by a set of vertex lists.

As discussed in Chapter 3, the user is required to define a template for each class of primitive that is needed (for instance a cuboid or a tetrahedron). Each of these templates is stored on disk in a file that follows a specific format, and can be loaded into the framework to add an additional class of primitive for use during the modelling process. Although a utility exists to automatically write primitive templates to file, it is quite simple to create a template by constructing an appropriate template file directly.

Every template file must adhere to the following format.

- The first line of the file contains a string, which represents the name of the primitive template.
- On the second line of the file, is a single integer, V, that denotes the number of vertices.
- The following $3V$ lines represent the Expressions that define the x, y and z coordinates of each of the V vertices. Each of these lines must be an arithmetic expression, that may use the addition, multiplication, subtraction and bracket operators, and make reference to only the symbols `xM`, `yM` and `zM` (that refer to the x, y and z extents of the bounding box) or numeric values.
- The following line, again, contains only a single integer E, that specifies the number of edges

in the primitive

- The subsequent E lines each contain exactly two integers that specify the end vertices for each line.
- The next line contains a single integer F, the number of faces in the primitive.
- The final F lines each contain a list of integers that define the vertices of the faces (a clockwise winding convention is used).

On loading a template file, a primitive constructor object is created that parses the file and creates generic Expressions from each of the arithmetic expressions describing the vertices. Whenever the user creates a new primitive based on that template, the constructor object is used to instantiate the primitive and uses the generic Expressions and connectivity information from the template file to specify the vertices, edges and faces of the new primitive.

The Scene Graph

In order to construct a model, all of the primitives used to define the geometry are arranged into a hierarchical scene tree. This data structure is implemented as a standard tree, where each node has one parent and an arbitrary number of children, but also has an associated primitive.

Recall, from Chapter 3, that to compute the error metric and optimize the scene, it is necessary to calculate the positions of vertices, and directions of edges in world coordinates. This can be achieved for any vertex or direction vector by simply moving up the tree to the root node while building an Expression that, at each node, applies the relevant rotation and translation to the Expression received from the previous node. The resultant Expression will always, then, be in terms of the parameters of the model, and can be used directly in the optimization process.

This same technique of evaluating world coordinates of vertices is applied both when rendering the geometry during the modelling phase, and when writing the final polygonal model to disk after the final optimization.

4.2.4 Constraints

The ability to constrain aspects of the geometry is, again, implemented through the use of Expressions. Any of the Expressions of any of the geometric primitives may be either set to a constant value or set equal to some other Expression. Using the equality operator, one can ensure that two parameters always share the same value, and by creating a combination of two or more other Expressions, it is possible to ensure that any number of primitives always fit together exactly.

As the Expressions are combined by reference (through the use of *pointers*), changes made to the value of any single Expression will affect all other Expressions that are dependent on it, thus ensuring

consistency throughout the geometry model. During the optimization process, only the Expressions which are set equal to a numerical, variable value are considered to be free parameters.

4.2.5 Correspondences

The implementation of the image-model correspondences is fairly simple: a list of elements is stored, where each element contains a reference to both a camera and a primitive. The reference to the camera also specifies which user-defined observation of that camera is used, while the reference to the primitive specifies to which model edge the observation corresponds.

4.3 The Optimizer

The Optimizer is the component of the framework that handles all aspects of the optimization process. This section presents details on a number of the more interesting and important aspects of the implementation of this process.

This includes the implementation of the modified Newton's method, where special attention is paid to the techniques used when calculating derivatives and optimizing over the Lie group $SO(3)$, as well as the implementations of both the initial optimization and the non-linear optimization algorithms.

4.3.1 Newton's method

The variant of Newton's method, described in Chapter 3, is implemented by making use of Expressions and a system of linear equations solver available in the *GNU Scientific Library* (*GSL*) [GSLDocs, 2007]. The strategy used is described briefly below, followed by details on the two most important aspects of the process.

Given an objective function $\mathcal{O} = t_1 + t_2 + \cdots + t_m$ that must be minimized, where each term t_i is represented by an Expression that has been extracted from the model, the optimization algorithm runs as follows:

1. Examine the m Expressions and the model to determine the set of free parameters $\mathbf{p} = \{p_1, \ldots, p_n\}$ that must be optimized.
2. Calculate Expressions for $\nabla\mathcal{O}$, the set of all first order partial derivatives of $\mathcal{O}$, and evaluate these Expressions, storing the results in an array.
3. Calculate Expressions for H, the set of all second order partial derivatives of $\mathcal{O}$, and evaluate these Expressions, again storing the results in an array.

4. Solve the system of equations, $H\mathbf{p}_{new} = H\mathbf{p} - \nabla\mathcal{O}$. In this system, a *GSL* implementation of a fast linear system solver that makes use of Houscholder transformations is used to solve for $\mathbf{p}_{new}$.

5. Update $\mathbf{p}$ and, if the error has not changed for two consecutive iterations (to ensure convergence), terminate the optimization. If the error has decreased, go to step 2.

Two important considerations in the above algorithm are the method used to calculate derivatives of $\mathcal{O}$ (when computing the Hessian and gradient) and the way in which the local parametrisation of the rotation matrices is handled at each iteration.

Calculating Derivatives

It is clearly necessary to be able to calculate derivatives of Expressions. Given that the objective functions (and implicitly the Expressions) that must be differentiated vary with every reconstruction, it is consequently required that the derivatives of arbitrary Expressions can be calculated with respect to any of the free parameters.

One solution to this problem is to create a function that takes as input an Expression and a parameter, and returns a new Expression that is the derivative of the input Expression with respect to the input parameter. Using this function, calculating the gradient $\nabla\mathcal{O}$ of the objective function $\mathcal{O}$ would simply involve computing an Expression for each first order partial derivative of $\mathcal{O}$. Similarly, the Hessian could be determined by calculating, for each Expression $\nabla\mathcal{O}_i$ in $\nabla\mathcal{O}$, a vector of Expressions representing all first order partial derivatives of $\nabla\mathcal{O}_i$ (yielding a matrix of all second order partial derivatives of $\mathcal{O}$).

Given an Expression such as that depicted in Figure 18, it is possible to calculate its derivative by simply applying the basic rules of differentiation for polynomial functions (i.e., the product rule, the power rule and the chain rule). Where the Expression contains matrix or vector components, the generalised rules of differentiation (i.e., the matrix product rule, the cross product rule, etc.) can be applied.

These differentiation rules are applied to an Expression, by starting at the root node and recursively differentiating the children nodes. At each node, the derivative of its Expression is returned, and used when calculating that of its parent node. Depending on the operation of an internal node, different combinations of its child nodes and their derivatives are returned. For leaf nodes, however, only 1 or 0 is ever returned (depending on whether or not the node represents the parameter in question).

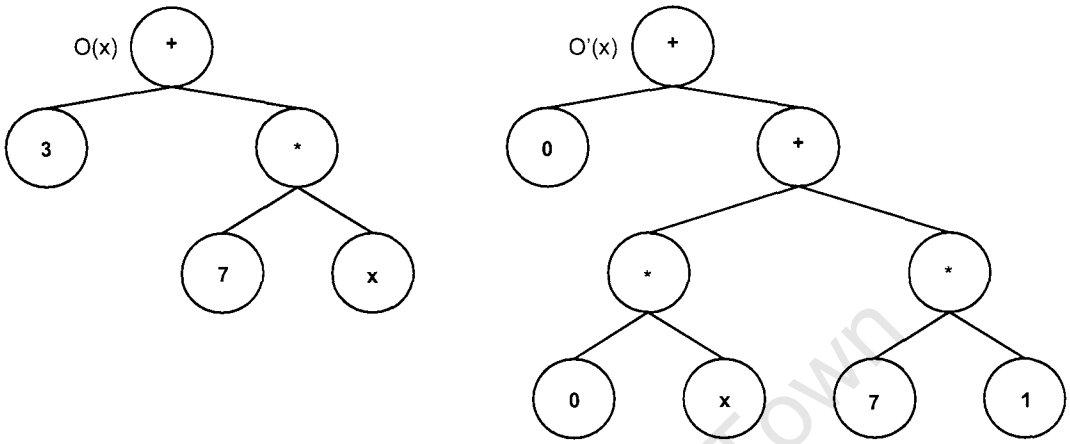


Figure 18: Differentiating an Expression tree. Consider a function $\mathbf{O}(\mathbf{x})$, that is described by the Expression tree on the left. The derivative of this function, with respect to $\mathbf{x}$ can be calculated by recursively differentiating each node in the tree, resulting in $\mathbf{O}'(\mathbf{x})$, represented by the Expression tree on the right. This case is a good example of the application of the product rule, which was applied to the right branch of the initial tree.

Optimization over $SO(3)$

Clearly, a number of the parameters of the model that must be optimized will be those defining the rotations of either cameras or primitives of the model. As discussed in Chapter 3, a local parametrisation of each rotation matrix to a set of only three parameters is used at each iteration of the optimization.

To implement this, each matrix Expression (that is a leaf node in an Expression tree), also has an additional three parameters with respect to which it can be differentiated. When the list of free parameters of the objective function $\mathcal{O}$ is determined, instead of adding nine parameters to that list, each matrix Expression only adds these additional three parameters.

When the derivative of one of these matrix Expressions with respect to one of its parameters is calculated, it is simply marked as a first (or second if this is the case) derivative matrix, and then evaluated appropriately only when the Expression containing the derivative matrix is evaluated.

The parametrisation of a matrix, R_0, is expressed as $R(\omega) = R_0 exp\{J(\omega)\}$, where $\omega = (\omega_x, \omega_y, \omega_z)^T$ are the new parameters, and has the property that $R(\omega_0 = (0, 0, 0)) = R_0$. To optimize an objective function, $\mathcal{O}$, with respect to the elements of ω, we need to calculate the first and second order derivatives of $R(\omega)$. However, since the parametrisation happens at every iteration, we only need to evaluate these derivatives at ω_0. Taylor and Kriegman [1994] show that the first and second order

derivatives of $R(\omega)$, evaluated at ω_0, can be calculated as follows

$$\frac{\partial}{\partial \omega_x} R(\omega) = \frac{\partial}{\partial \omega_x}(R_0 exp\{J(\omega)\}) = R_0 J(\hat{x})$$

$$\frac{\partial^2}{\partial \omega_x \partial \omega_y} R(\omega) = \frac{\partial^2}{\partial \omega_x \partial \omega_y}(R_0 exp\{J(\omega)\}) = R_0 \frac{1}{2}(J(\hat{x})J(\hat{y}) + J(\hat{y})J(\hat{x})$$

where $\hat{x} = (1, 0, 0)$ and $\hat{y} = (0, 1, 0)$. x or y can, of course, be substituted for z for the final partial derivatives.

These results are used in computing the Hessian and gradient of the objective function $\mathcal{O}$, and yield an update step, w_s. This update step is then applied, using a quaternion representation described by Taylor and Kriegman [1994], to provide a new rotation matrix as follows.

The initial rotation matrix, R_0 can be represented by a unit quaternion $\mathbf{q}_0$. The incremental step, $exp\{J(\omega_s)\}$ can be represented by a quaternion $\mathbf{q}_s = (\cos(\frac{\theta}{2}), (\frac{\sin(\frac{\theta}{2})}{\theta})\omega)$ with $\theta = \sqrt{\omega^T \omega}$. The resultant quaternion $\mathbf{q}_r$ can then be computed as $\mathbf{q}_r = \mathbf{q}_0 \mathbf{q}_s$ where standard quaternion multiplication is used. Converting $\mathbf{q}_r$ to a matrix yields the updated rotation.

4.3.2 Initial Parameter Estimation

In order to implement the initial optimization, all that is required is that Expressions for each of the two objective functions be defined. As each objective function is, in fact, a summation of a number of smaller terms, it is more convenient to use a number of smaller Expressions — each representing one of these terms.

First Initial Estimation

The objective function $\mathcal{O}_1$ for the first initial estimation step is used to optimize only the rotation matrices of the cameras. Each term t_i of this function is of the form $\mathbf{m}'^T R_j \mathbf{v}$. As the free parameters appear only in the term R_j, all other terms can be treated as constants, and as a result only first and second order derivatives of R_j need to be calculated at runtime.

Furthermore, as the cameras are independent of one another, each camera can be optimized by itself. The result of this is that it is possible to run one small optimization process for each camera rotation matrix, rather than a single global optimization over all of the cameras. This is efficient, as it avoids calculating derivatives that will always be equal to 0, and results in far fewer needless calculations being performed.

One final consideration in this step, is that only edges with a known orientation can be used. Where the orientation of the edge is dependent on the dimensions of the primitive (i.e., the slanted edges of a prism) or on the rotation of the primitive (where the rotation is a free parameter), it cannot be used. To achieve this, only horizontal and vertical edges, where all rotation matrices that relate the edge to world coordinates are constant, are used in this optimization step.

Second Initial Estimation

Where $\mathcal{O}_1$ was used to optimize only the rotation matrices of the cameras, $\mathcal{O}_2$ is minimized to solve for all of the remaining parameters of the model. As a result, for a term of the objective function t_i with the form $(\mathbf{m}^T R_j(P_i - t_j))^2 + (\mathbf{m}^T R_j(Q_i - t_j))^2$, all parameters except for R_j introduce free variables, and derivatives of all of these terms must be computed.

In addition to this, where the rotations of the cameras were independent of one another, one cannot guarantee such independence for the free variables of this objective function. As a result, Expressions must be generated for all model edge and image edge pairs, and the first and second order derivatives of each of these Expressions must contribute to the gradient and Hessian used in the simultaneous optimization of all of the parameters.

4.3.3 Non-linear Optimizations

The non-linear objective function $\mathcal{O}$ comprises a number of terms of the form $\mathbf{m}^T(A^T B A)\mathbf{m}$ where $\mathbf{m}$, A and B are defined as in Chapter 3. It transpires, as B is a function of the elements of $\mathbf{m}$, that all the free parameters are contained only in B and $\mathbf{m}$ and that to compute derivatives of the terms of $\mathcal{O}$, only $\mathbf{m}$ needs to be differentiated at runtime (the derivatives of the entire B term will always remain the same function of the derivatives of $\mathbf{m}$).

Using this reasoning, Expressions need only be built, differentiated and evaluated for the $\mathbf{m}$ in each term, and this can be done quite simply by analysing each correspondence that is used in the optimization. Using these, it is simple to evaluate both the gradient and Hessian of $\mathcal{O}$ used in the optimization process.

Chapter 5

Photo-based Texturing

As discussed in Chapter 2, due to the quality of the photographs in the archive, there are a number of problems associated with reconstructing District Six architectural scenes. Although some of these problems affect only the reconstruction of the scene geometry, a number of them are also relevant when considering the texturing of the reconstructed model, and, as a result, it was necessary to develop and implement strategies to overcome these problems.

The specific issues, and the effects that they have on the texturing process are reiterated below:

- **A limited number of available photographs.** For any given scene, it is very difficult to find more than two or three photographs. This is problematic for the texturing of the reconstructed model, since there is often no textural information available for substantial portions of the model. Furthermore, as none of these photographs were taken with the production of a digital model in mind, parts of the building are often obscured from view by other objects, thus further exacerbating the problem of insufficient textural information.
- **The long period of time over which the photographs were taken.** From a texturing perspective, this is problematic as certain schemes, such as that described by Debevec et al. [1996], assume that all photographs of the scene are taken under similar lighting conditions, and that they capture the same textural information. Where long time periods have passed between two images, there can be no guarantee that the lighting conditions have not changed, nor indeed that there have been no changes to the actual underlying texture of the building. Combining multiple textures when these two criteria are not guaranteed will generally yield poor results and is thus not suitable in most cases.
- **Different chromatic schemes.** Working with textural information extracted from photographs with differing chromatic schemes introduces problems similar to those associated with temporal variance within a set of photographs. As with changes in lighting conditions and underlying texture, different chromatic schemes also present problems when combining

multiple textures and this, again, means that conventional texturing schemes will not be effective for many cases.

A user-interactive strategy is presented, which overcomes these problems by using a combination of the original textures, synthesised textures and image manipulation tools. The remainder of this Chapter is in two parts: a section focusing on an overview of this technique — where the User's View and functionality of the system are introduced — and a section dedicated to the implementation details of the strategy.

5.1 Overview

Our texturing strategy involves first extracting all of the available textural information from the input photographs, and then using this information, guided by a user, to texture all the visible surfaces of the model.

Wherever possible, the original textures extracted from the photographs are used to texture the surfaces of the model. However, where these textures are incomplete, damaged or non-existent, the user may then use a texture synthesis approach to either complete, repair or entirely reconstruct the textures. In addition, the user may copy features (such as doors or windows) from the complete textures or import new textures (from external files), and use these to aid in the repairing and reconstruction of the textures.

It should be noted that in this texturing scheme only one texture is used per polygon. One consequence of this is that view-dependent texturing is not supported, but it would, in general, not be applicable in any case, due to the data source problems discussed above.

5.1.1 User's View

This texturing technique provides the user with a number of tools to assist in the repairing and construction of the model's textures. Initially, textures for each polygon in each primitive of the model are automatically extracted from all input photographs. Thereafter, for each visible polygon in the model, the user must decide which texture to use, and also which portions of that texture need to be repaired or synthesized.

Figure 19 presents a flow chart that illustrates the operations that are performed by the user when working with the texture for any given polygon, and is followed by a brief discussion of where and how these operations are applied.

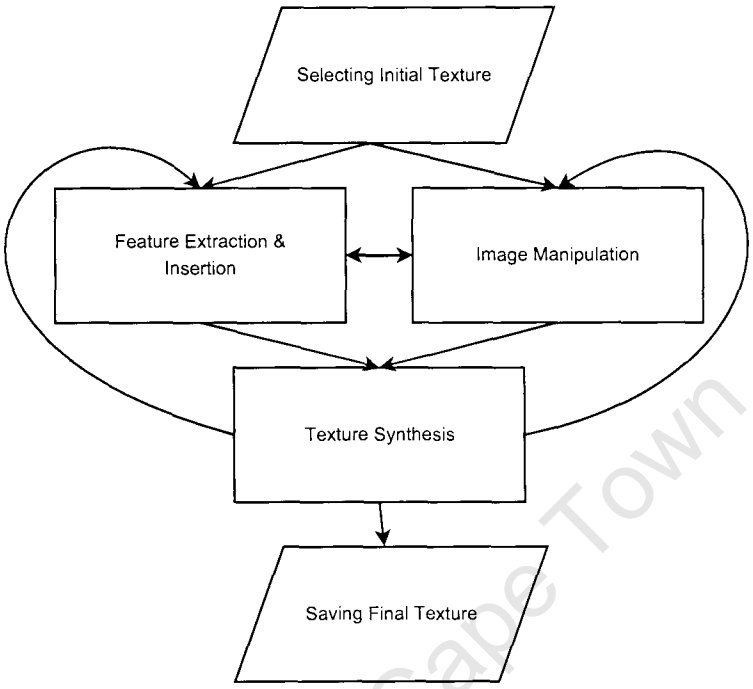


Figure 19: A flowchart describing the process of texturing a reconstructed model. The processes followed by the user while texturing a polygon of the model is illustrated by this flowchart. Although the texture synthesis can, in fact, be performed before feature extraction and insertion, it is not recommended as the synthesis process may then need to be repeated.

Throughout the texturing stage, the model is displayed in the rendering window, textured with the most current textures for each polygon. If dissatisfied with any of these polygons, the user may select its texture for editing by simply clicking on that polygon of the model.

When this is done, the user is presented with the set of all textures that have been extracted from the input photographs, along with the current texture for that polygon. From this set, the user is required to select a texture that will be used as the base texture for that polygon. Where there are numerous complete or incomplete textures for a polygon, the user may choose the one that is most suitable. When the polygon is not visible in any of the photographs, a blank texture is used as the starting point.

The next step generally involves the extraction and insertion of textural features. In this context, a feature is most typically the sub-texture for an object such as a door or window, but can also refer to samples of underlying textures such as paint or brickwork. Where the user is working with a complete texture, features are generally extracted for repairing or reconstructing other textures. Where the current texture is incomplete, pre-extracted features may be inserted for use in the synthesis process, or as additions to newly synthesised textures.

At this same stage, the user may also make use of the image manipulation functionality. Typically, this only involves erasing portions of the texture that are unusable, or converting colour textures to

greyscale, however it is also possible to export textures for manipulation by external utilities and then import the modified textures at a later stage.

When there is a sufficiently large texture sample (typically an area of 80 by 80 pixels is used), and if the texture is new or incomplete, the user may synthesize the remainder of the base texture using the synthesis utility provided. Although this can be performed before or after any of the image manipulation operations, it is a slow procedure and should only be performed once per polygon. After the base texture has been synthesized, the user may again insert any number of features to complete the texture.

At any point during this process, the user may update the texture for a polygon, and the model displayed in the rendering window will reflect any changes. Once satisfied, the user may save a final version of the textured model for rendering at a later stage.

5.1.2 Function Details

A number of functions — or operations — have been implemented that allow the user to perform the tasks discussed above. Precise details of these functions, coupled with a discussion of where they are used and what problems they are intended to solve, are presented below.

Texture Extraction

The first operation — although not one that the user instigates — is the extraction of textures from the input images. This is performed automatically as a pre-computation. The biggest problems here are correcting for the perspective projection of the camera, and determining which parts of the model are visible in the photograph (only self-occlusion by the geometry of the model is considered here). The implementation of this operation is fully discussed in Section 5.2.

Although not directly used to solve any particular problem, correct texture extraction is an essential and fundamental component of creating textures for the final model.

Feature Extraction and Insertion

In order to repair and complete damaged or incomplete textures, it is often necessary to add "features" to the base texture. Features such as doors, windows or plaster patterns add greatly to the perceived realism of a texture. In addition, where a texture is, at first, completely blank, one must be able to insert a sample region of some base texture for use in the synthesis process.

From any of the textures extracted from the original photographs, the user is able to select a rectangular region and save this to the feature clipboard. Once on the clipboard, a feature can then

be inserted into any texture used on the model. Should it be required, a feature can be rotated in increments of ninety degrees — this is useful for changing the orientation of ornamental features.

Where any further manipulation of a feature is required, it can be exported to an external image file for editing purposes and then imported at a later stage. As a result, it is possible to save features from one reconstruction for use in another. Since much of the architecture in the District Six scenario shares similar traits, this is a useful capability when constructing completely new textures.

Image Manipulation Tools

The two most frequently used image manipulation tools (aside from texture synthesis and feature insertion) are a "blackout" tool and a conversion to greyscale tool. The "blackout" tool allows the user to simply erase portions of a given texture. This is essential, as in many cases, parts of the scene are obscured from view by external objects such as trees, cars or people. Regions of a texture that have been erased are filled in at a later stage either by inserting features or by performing texture synthesis.

The tool that converts images to greyscale is used primarily when only a small number of the input images are in colour. It effectively provides the simplest solution to the problem of differing chromatic schemes — that is converting all images to greyscale. Clearly, when textures for a number of polygons of the model exist only in black and white, having other polygons with colour textures yields unsuitable results.

Although these two tools are the most frequently used, it will occasionally be the case that other image manipulation tools are required. For instance, where the user may wish to attempt greyscale to colour conversion or where the texture needs to be cleaned of noise. Instead of implementing an array of such well defined but infrequently used tools, export and import functionality has been added to allow the user to perform image processing operations with external applications. In this way, specialist software can be applied to any of the more difficult texture manipulation problems.

Texture Synthesis

The texture synthesis tool is the most fundamental aspect of this texturing approach. Since, in almost every case, there will be a number of polygons with no textural information, it is essential to be able to construct realistic looking textures for these surfaces. To do so, a texture synthesis algorithm, developed by Efros and Leung [1999], that can generate arbitrarily large, realistic textures, given only a small sample texture as input, is used.

To run the synthesis algorithm, it is required that a small sample patch of the desired texture is already in the image. For textures that are only damaged or incomplete, there is usually a sufficiently large sample already available. Where the texture at hand is to be entirely synthesized, however,

the user is required to use the feature insertion tool to provide such a sample. To synthesize the remainder of the texture, the user need only select a rectangular portion of the sample in order to proceed.

As mentioned earlier, one consideration here is that the texture synthesis algorithm is slow and does take a number of minutes to execute, and the running time is dependant on both the sample size and the size of the entire texture to be generated. As a result, the user should ensure that only one texture per polygon need ever be generated, and this should be taken into consideration when deciding on the order in which functions are performed.

5.2 Implementation Details

Before discussing specific details of the implementation of this texturing strategy, one should first consider the objectives, as these affect the implementation decisions that were made. Briefly, these are as follows:

- **Correct extraction of textural information from input photographs.** The primary goal is to extract relevant textures from the input photographs, while accurately correcting for perspective projection distortion caused by the viewing position of the camera. It should be noted that the quality of the extracted textures, at this stage, is of greater concern than the speed of extraction. Determining the visibility from each camera position is also required in order to ensure that the correct textural information is extracted for each polygon.
- **Maintaining the size ratio between the textures of polygons.** To simplify the problem of feature extraction and insertion, it is best to ensure that the ratio between the size of the textures of any two polygons is the same as the ratio between the actual size of the two polygons. If this is the case, then features extracted from one texture will generally not need to be scaled in order to fit into another.
- **Building a single texture for each polygon.** As discussed earlier, the output of the texturing stage should be a single texture for each polygon. Furthermore, all of these textures should be packed into a single, large texture map in order to simplify the rendering process. These two conditions affect the way the current textures for the polygons are stored during the texturing process.
- **Feature insertion and extraction, image manipulation and texture synthesis.** In order to implement the required functionality for our texturing strategy, certain considerations had to be made in deciding on the internal representation of the textures.

Keeping these objectives in mind, the remainder of this section covers, briefly, the architecture of the texturing system and, in detail, the implementation of the texture extraction process, feature insertion and extraction functionality, image manipulation tools and the texture synthesis algorithm.

5.2.1 System Architecture

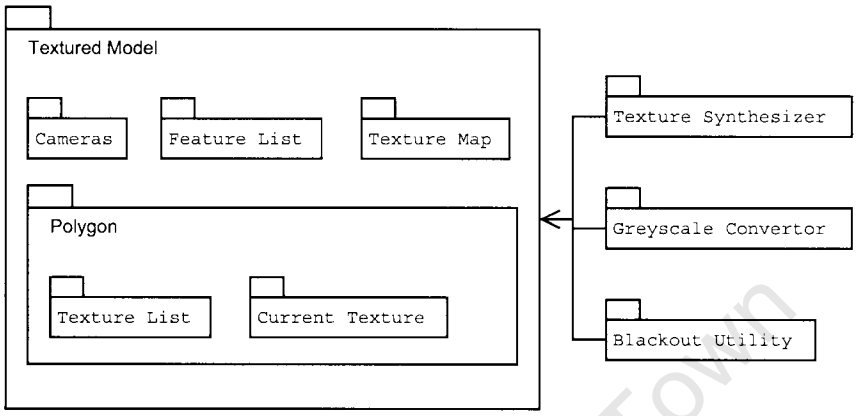


Figure 20: The architecture of the texturing system.

Consider the diagram in Figure 20 above: the core component of the texturing system is the *Textured Model*. This is initially used to store information about each of the input images, including image data and camera parameters, as well as all polygons in the model. The *Textured Model* is then used to extract textural information from each camera for each polygon, which is subsequently stored in the *Texture List*. For each polygon, the *Current Texture*, which represents the data currently being used to texture that polygon, is also maintained.

The feature extraction and insertion functionality is built into the *Textured Model*, and all of the extracted features are stored in the *Feature List*. All importing and exporting of these features, as well as any of the other textures, is performed by the *Textured Model*. Finally, the *Texture Map*, which stores textural information for every texture, and is used to texture the model is also located within the *Textured Model*.

The texture synthesis function, blackout function and greyscale conversion function are all implemented separately (in the *Texture Synthesizer*, *Greyscale Converter*, and *Blackout Utility*, respectively), and are applied to elements within the *Textured Model* when needed. Modifications are made directly to textures or features, but never to the *Texture Map*, which is only updated by copying, from a given polygon, the *Current Texture*.

Every image in the *Textured Model* is stored as an array of bytes, where every (non-overlapping) set of four bytes represents a pixel (using the RGBA format). This representation is used as each pixel in an image may have special properties (such as whether it needs to be synthesized) associated with it, which can be completely encoded in its alpha channel.

5.2.2 Texture Extraction

The extraction of textures from the original photographs is a combination of three processes. The first of these involves determining, for each polygon, a correct texture size (relative to the sizes of the textures of other polygons), as well as texture coordinates for each of the vertices of the polygon.

The second task involves efficiently packing one texture per polygon, into the large texture map, a process which may involve resizing all of the textures (by a constant factor to ensure that relative sizes are preserved). The final process is the actual sampling of the original photographs to determine the correct pixel values for all of the textures.

Computing Texture Size and Image Coordinates

When computing the texture size and image coordinates for the vertices of a polygon the two major concerns are: ensuring that the size of the resultant texture is proportional to the size of its polygon (thus ensuring that size ratios between textures are maintained), and that the texture is oriented correctly (i.e., horizontal edges of the polygon should appear horizontal in the texture). One should also note at this point, that even when a polygon is not rectangular, its texture is embedded inside a rectangular bounding box.

Since all of the polygons in the model are assumed to be planar, one simple solution to this problem involves calculating a basis for the plane in which the polygon lies, and using this basis to determine image-plane coordinates for each vertex of the polygon. In this context, a basis for such a plane is a pair of orthogonal unit vectors, $\mathbf{u} = \{u_x, u_y, u_z\}$ and $\mathbf{v} = \{v_x, v_y, v_z\}$, that lie in the plane of the polygon.

Using this basis, every point on the plane can now be uniquely defined as a linear combination of $\mathbf{u}$ and $\mathbf{v}$. Since $\mathbf{u}$ and $\mathbf{v}$ are orthogonal, given a vertex of the polygon $\mathbf{p} = a\mathbf{u} + b\mathbf{v}$, the image-plane coordinates of $\mathbf{p}$, p_x and p_y, can simply be defined as $p_x = s(a + q)$ and $p_y = s(b + r)$, where q and r are constants used to ensure that p_x and p_y are both greater than 0, and s is a constant scaling factor used to determine the size ratio between distances on the image-plane and distance in world coordinates.

Provided, for every polygon, that the unit size (used to define $\mathbf{u}$ and $\mathbf{v}$) is the same, this method will ensure that the size of all textures is proportional to the size of their polygons. Furthermore, by selecting appropriate vectors for $\mathbf{u}$ and $\mathbf{v}$, one can further ensure that the texture will be correctly oriented.

Consider Figure 21. In order to compute the two basis vectors, $\mathbf{u}$ and $\mathbf{v}$, one must first select two edges of the polygon, $\mathbf{u}'$ and $\mathbf{v}'$, that share a common vertex (which is treated as the origin for the basis). It is the selection of these two edges that determines the orientation of the texture. By choosing $\mathbf{u}'$ to be the lowest, most horizontal edge, and ensuring that $\mathbf{u}'$ is oriented clockwise, one

can be certain that the texture will be oriented correctly.

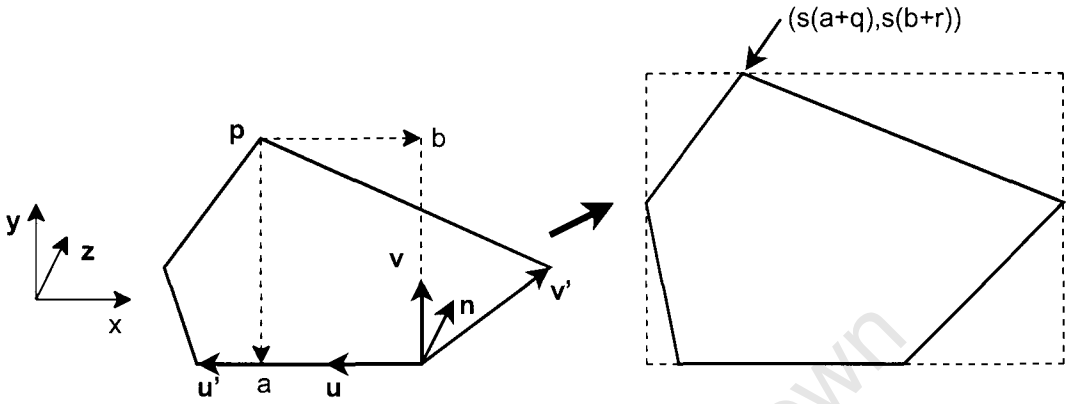


Figure 21: Illustration of how a basis for each polygon is determined

The next stage is to compute a unit normal to the polygon, $\mathbf{n} = |\mathbf{v}' \times \mathbf{u}'|$. From here, $\mathbf{u}$ is trivial to compute as $\mathbf{u} = |\mathbf{u}'|$. Given $\mathbf{n}$ and $\mathbf{u}$, the second vector of the basis can be computed as $\mathbf{v} = \mathbf{u} \times \mathbf{n}$. The coefficients of $\mathbf{u}$ and $\mathbf{v}$ for any vertex $\mathbf{p}$ can be simply computed as $a = \mathbf{p} \cdot \mathbf{u}$ and $b = \mathbf{p} \cdot \mathbf{v}$.

Determining the values for q and r is also simple, and involves finding the smallest value of a and b for any vertex of the polygon. All that then remains is to compute a value for s, that scales the texture to an appropriate size. Initially, we set the size of the longest edge of all of the textures to be 600 pixels, and thus s is computed as $s = \frac{600}{l}$ where l is the unscaled length of the longest edge. This value of s is, however, modified during the packing of the textures to ensure that all of the textures fit inside the texture map.

Packing Textures

Having computed texture sizes for the textures of every polygon, the next problem involves packing these textures into the texture map. However, there is no guarantee that all of the textures will indeed fit into the texture map at their current sizes. Furthermore, even if they do, finding the order or positions in which the textures must be packed is a difficult problem that can only be solved by an exhaustive search, which is extremely slow.

One solution to this problem, discussed by Sander et al. [2001], involves repeatedly attempting to pack the textures into the texture map using a "naive" algorithm, and, should they not fit, reducing the scale factor by a small margin. Although not optimal, this solution is fast and provides satisfactory results. Using this approach, all that remains to do, is discuss the naive packing algorithm.

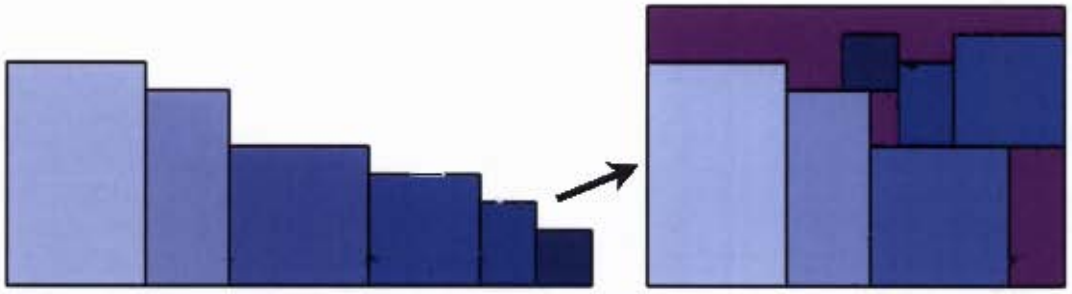


Figure 22: An illustration of how the textures are ordered by height, and then packed from left to right to left, and so on.

The algorithm used to pack textures into the texture map is illustrated in Figure 22. The first stage of this algorithm is to sort all of the textures by height, in descending order. From there, the textures are packed, in order, into the texture map, starting at the bottom left of the texture map and moving right. When no further texture can be added to the right of the previous one without exceeding the width of the texture map, a new row of textures is started, above the previous row, but this time proceeding from right to left.

In the second row (and similarly all others), the textures are seated as low as possible in the texture map without overlapping any on the previous row. This process of layering textures continues either until all of the textures have been packed, or until the top of any texture exceeds the maximum height of the texture map. If this happens, the algorithm has failed, and the texture sizes must be reduced. If not, the packing has succeeded and the entire process is complete.

Sampling Original Photographs

The final stage of extracting textures from the original photographs involves sampling the source photographs. This process is left until after the packing has been completed, as it is only at this point that the size of the texture that must be created is known.

The primary problem with sampling texture data from the input photographs is that the perspective of the camera must be corrected for. Figure 23 below illustrates that, as a result of the perspective projection of a camera, there is no linear mapping between texture coordinates and image coordinates, and thus that sampling linearly along the input image would result in badly distorted textures.

Since, however, the camera position and orientation for each of the input images is known, if one can establish the position, in world coordinates, of any given pixel on the texture, one can then project that position onto the original image to determine which of the input pixels should be sampled, thus correcting for the perspective projection of the camera.

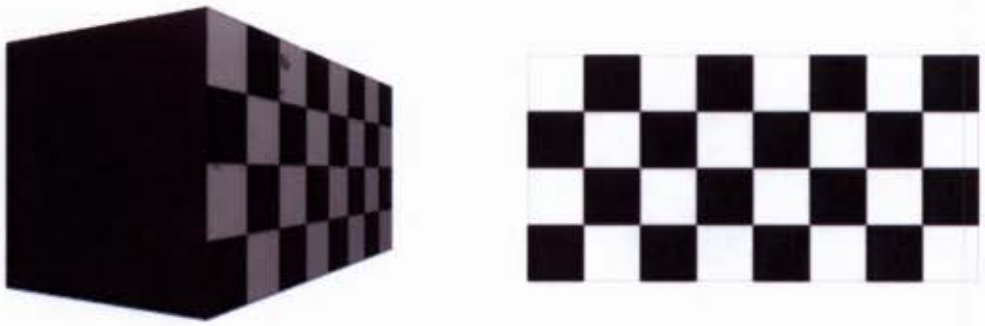


Figure 23: An illustration of how a perspective projection causes fore-shortening. Note how the squares towards the "far" end of the cuboid appear packed more closely together than those at the "near" end.

Recall that, when computing the texture coordinates for the vertices of a polygon, it was necessary to compute a pair of vectors, $\mathbf{u}$ and $\mathbf{v}$, that formed a basis for the plane in which the polygon lies. Using these vectors, the texture coordinates of an arbitrary vertex, $\mathbf{p}$ of the polygon, can be expressed as $x = s(a + a)$ and $y = s(r + b)$, where $a = \mathbf{p} \cdot \mathbf{u}$, $b = \mathbf{p} \cdot \mathbf{v}$ and $\mathbf{p} = a\mathbf{u} + b\mathbf{v}$.

From this, it becomes clear that the world coordinates (where the origin is the vertex that $\mathbf{u}$ and $\mathbf{v}$ share) of any texture coordinate, (x', y'), can be expressed in terms of the basis vectors as follows:

$$\mathbf{p}' = (\frac{x'}{s} - a)\mathbf{u} + (\frac{y'}{s} - b)\mathbf{v}$$

Given the world coordinates for $\mathbf{p}'$, all that needs to be done is to project it onto the image plane of the original photograph using the transformation matrix for the associated camera. In most cases, the resulting image-plane coordinates will not fall directly on any single pixel, and to solve this aliasing problem, a linearly weighted average of the four nearest pixels is used.

There is one final issue that must be solved before the texture sampling is complete, and that is the problem of visibility. Where one polygon is fully, or partially, obstructed by another, for the obstructed region, we do not want to extract any texture data. To solve this problem, we use a technique similar to that described in Debevec et al. [1998].

Each polygon is assigned a unique colour, and then for each input photograph, the entire model is rendered to an image, using the camera parameters of that photograph. Each pixel of this rendered image will now have a colour that indicates exactly which polygon is visible in that pixel. Overlaying this new image onto the original photograph, we are able extract only the relevant information for each polygon.

Since all of the textures are embedded in a bounding box, one additional, useful piece of information is whether a pixel within the bounding belongs to on the texture. To solve this problem, a similar technique to that of the visibility solution is used. Again, using the input photograph's camera parameters, just the single polygon is rendered to an image. This image is then used to determine

which pixels of the texture (even if not visible) belong to that polygon.

At this point it is important to discuss how the mask (the alpha value in the texture) is used. For the texture synthesis algorithm, described below, it is necessary to know which pixels need to be synthesized and which do not. Therefore, for all pixels of a texture, where its polygon is visible, the mask value is set to 0. Where it is obscured from view, but still belongs to the polygon, it is set to 1, and where it does not belong to the polygon, it is set to 2.

5.2.3 Texture Synthesis

As mentioned earlier, the texture synthesis approach introduced by Efros and Leung [1999] was used in our texturing strategy. The details of how this algorithm works, along with our implementation and some minor performance enhancements are presented below, and examples of textures synthesized using our implementation are exhibited in Figure 24.

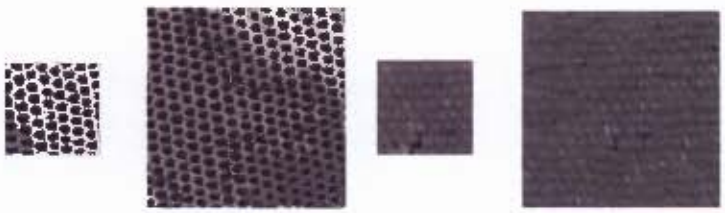
Essentially, this texture synthesis algorithm works by modelling a texture as a Markov Random Field [Efros and Leung, 1999]. What this means, is that the probability distribution for the value of a particular, unknown, pixel is assumed to be dependant only on the values of the set of other pixels within a certain neighbourhood of that pixel, rather than being dependant on the entire image. The result of this model is that globally, a certain amount of noise is permitted, while the local texture structure is generally still well preserved.

The algorithm starts by taking, as input, a sample texture and an image that must be completed. The incomplete image can either be entirely empty, or merely have a few holes that need to be filled. In our approach, the user is required to embed a sample texture into the image before synthesis, so the entirely empty case is never actually considered.

Now, for any empty pixel, that has at least one non-empty neighbour, a new value can be computed as follows: a window, centred around the empty pixel, of size w is taken. All empty pixels in this window are ignored, and only the rest are considered. For each pixel in the sample texture, a window of size w, centred around that pixel, is also taken.

An error for that pixel of the sample texture is then computed, by calculating the weighted square distance between the two windows, where the weight for each pixel is determined by its distance from the centre of the window — the further away it is, the lower its weight (determined by a two-dimensional Gaussian function). Only pixels that were not blank in the first window are used to calculate this error, and the final value is scaled appropriately.

In the original algorithm, a random pixel with a low error from the sample texture is used as the new pixel. However in this implementation, we find that better results are obtained by simply using the pixel from the sample texture with the lowest error value. This process, of filling in pixels one at a time, is repeated until the entire texture is complete.



(a) (b)

Figure 24: Synthesized textures. (a) and (b) are examples of textures synthesized by our implementation of the scheme introduced by Efros and Leung [1999]. (b) is especially pertinent as it is synthesized from a sample of the roof of the building in Figure 13.

5.2.4 Feature extraction and insertion

The feature extraction tool works by allowing the user to select a rectangular piece of any of the automatically extracted textures, by using the mouse to specify two opposing corners of the rectangle. When this is done, a new feature object is created, and the relevant texture data is copied to that feature object for later use.

The feature insertion tool allows the user to select any of the previously extracted features, and then copy it into the Current Texture for any polygon. When a feature is selected and the user elects to insert it into a texture, it is overlayed on top of the texture at the current mouse position (updated as the mouse is moved). When the user is satisfied with the position, the feature may be finalised, and its data is copied over the texture.

To ensure that the feature looks correct, linear blending between the feature and the original texture is performed over a three pixel wide border of the feature. We have experimentally determined that this width produces the most pleasing results. Finally, to prevent the texture synthesis algorithm from replacing features, the mask for all relevant pixels of the texture is set to 0.

5.2.5 Image Manipulation Tools

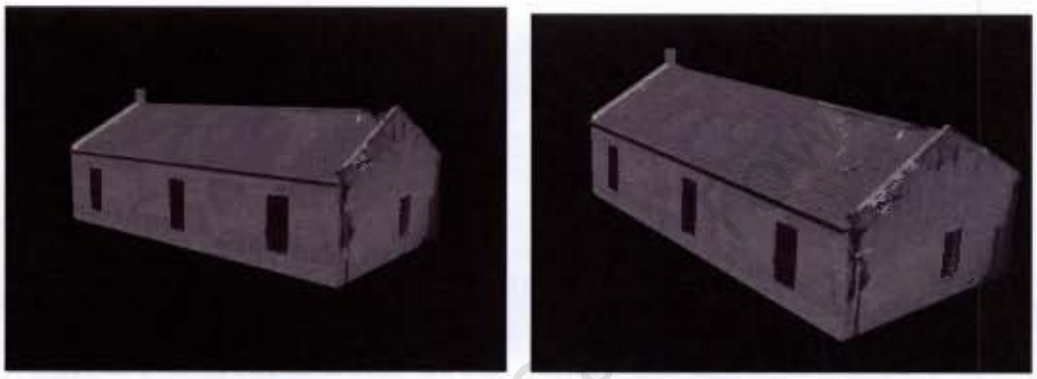
Both of the image manipulation tools are very simple to implement. The first of these — the blackout tool — simply resets the colour and the mask of a 10×10 area centred around the mouse cursor if the mouse is moved while the left mouse button is depressed. The colour is set to a medium grey, and the mask is set to 1 so that the region will always be correctly synthesized.

The greyscale conversion utility is also fairly simple. Each pixel of the texture is converted to its greyscale equivalent using the formula, described by Foley et al. [1995]

$$g = 0.35r + 0.59g + 0.06b$$

5.3 Real-world Example

Recall the reconstruction of the District Six residence, presented in Chapter 3. To illustrate the functionality of this texturing scheme, consider Figure 25 below, which contains two images of the fully textured model, after the texturing process has been applied.



(a) (b)

Figure 25: The final, textured, reconstructed model of the large residence.

The major problem with texturing this model, was that the original photographs (Figure 13) provided very poor texture information in the first, although at a high level of detail, only a little of the building was visible, while the second was taken from such a distance that only a little of the extracted textures were at all usable. In addition to this, a majority of the front wall and the roof was obscured by either a perimeter wall or trees, and this made using any of that texture material very difficult.

In order to solve the problems of the front wall and roof, the texture synthesis procedure, described above, was used. This worked exceptionally well on the roof, as the roof tile texture is very well structured, yielding authentic looking results. A texture was also synthesized for the front wall, but for this, samples were taken from the side wall as the lighting on the front wall was not ideal. The best synthesis result for the front wall texture was achieved by seeding the texture at a number of points, which creates less uniform results.

The front wall texture was completed by inserting a number of window features, that had been extracted from the original front wall texture, onto the synthesized texture. The back wall of the building, along with the far side wall and the other side of the roof were textured by copying the synthesized textures of the front wall, visible side wall and front-facing side of the roof, and then making minor changes to them. Finally the chimney was textured using information extracted from the second (distant) photograph.

The results of the application of this texturing strategy to the model are, considering the available input material, acceptable. The synthesized textures appear authentic, as do the features that have been inserted onto those textures.

Chapter 6

Results and Discussion

In order to analyse the success of the reconstruction and texturing techniques discussed in the previous chapters, it is necessary to critically consider a number of real-world situations. In addition to the example presented in Chapters 3 and 5, two further District Six reconstructions are examined in detail. The first of these is the British Bioscope, a large and complex building, reconstructed from two photographs, and the second is a small, simple, residential building, of which only a single image exists.

The discussion of the above real-world reconstruction examples focuses on the results that were achieved, looking primarily at how closely the reconstructed models match the input images and the success of the texturing technique, as well as any problems experienced during the reconstruction process.

Using these real-world cases and a computer generated synthetic case, the remainder of this chapter presents an analysis of each of the two major components of this reconstruction technique — the reconstruction framework and the texturing process — again focussing both on where they succeed and where they fail.

6.1 Real-world Examples

Three real-world examples are examined to provide test data for the analysis of the reconstruction system. Since it has already been discussed in previous chapters, only a brief overview of the first example is presented below. The other two cases are subsequently analysed in greater detail.

6.1.1 A large District Six residence

The first reconstruction, previously discussed in Chapters 3 and 5, is that of a large residence that existed prior to the demolition of the District Six region.

The input data for this reconstruction comprised two photographs of the residence, shown in Figure 13. The first photograph is captured from nearby, showing only part of the building, and the second from a significant distance away, showing the entire building, but at a very low resolution. The field of view for both cameras was computed to be approximately 45 degrees along the x axis.

The major problem with the reconstruction of this building, was the difficulty in marking out the edges where the building joined the ground in the first photograph, and the impossibility in doing so in the second. This is partially as a result of the ground being sloped, and partially as a result of a lack of landmarks on the photograph. Since it was not possible to mark out an edge at the base of the building on the second photograph, the corresponding edges are not aligned exactly as one would expect. Aside from this, however, the model, when projected onto the image-planes, matches the marked edges and visible geometry within the images with a high degree of conformity, as shown in Figure 15.

The texturing of this building was performed by using information from the close-up photograph almost exclusively. The second photograph was from such a distance that the textures extracted were almost entirely without detail. Referring back to Figure 25, the side wall of the house, visible in both images, was textured using the original information from the first image. The roof and front wall were badly obscured in both images, and thus new textures had to be synthesized from the available material. To complete the front wall texture, a window, that had been extracted from the first photograph, was inserted in a number of places. Finally, the chimney was textured using information obtained from the second image.

The texture synthesis procedure was particularly successful on the roof, as the texture had a very definite, repeating structure. Although still reasonably effective on the front walls, the results of the synthesis procedure on the paint texture were less convincing, as the paintwork texture sample was not particularly well structured, which results in a rather "noisy" effect. In order to achieve best results, the synthesis procedure was seeded at a number of points. Where the texture is not very well structured, this appears to produce more authentic looking output.

The final phase in the texturing of the obscured walls involved inserting a number of window features into the base texture. To do this, one copy of a window was extracted from the close-up input image, and then pasted into three places on the newly synthesized texture. The results of this are sufficiently convincing (as shown in Figure 25), and complete the texture generation process. For the remaining walls, (i.e., the back wall, and occluded side of the roof) the corresponding front-facing textures, with minor changes, were used.

6.1.2 The British Bioscope

The second sample reconstruction is of the British Bioscope that was located in District Six, prior to the re-zoning. This case is substantially more complex than the previous one for a number of reasons. First, the building has far greater geometric complexity than the previous example. This can be seen in the two input photographs of the building, shown in Figure 26. The building has a bay window, two chimneys and a piece of decorative geometry at the top of the façade.

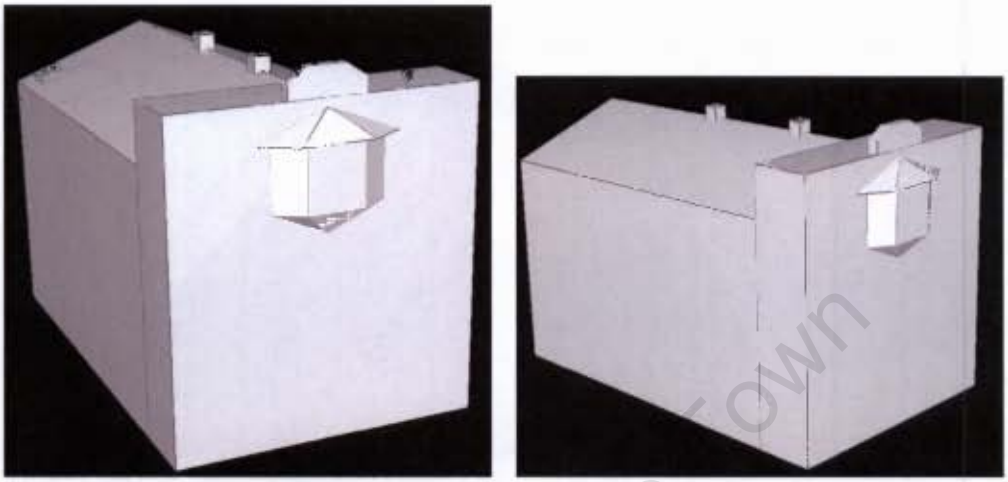
A second complication with this reconstruction, is that the two photographs have different chromatic schemes; the first is in colour, and the second in greyscale. To add to this, the second image is also substantially older than the first and of very poor quality. As a result, it was only useful in reconstructing the geometry, and not for texturing purposes.

A final complication with this example, is that both photographs are taken from a very similar vantage point, and both offer a poor view of the front wall. This is exacerbated by large portions of the building being obscured in both images by external geometry, including other buildings, a lamp post and a pillar box.

(a) (b)

Figure 26: The two photographs of the British Bioscope, used in its reconstruction. Note the poor viewing angle with respect to the front wall, and the obstructions evidenced in both of the photographs.

The geometry of the Bioscope was reconstructed successfully. The fields of view of the cameras used to take the two photographs were computed as being approximately 32 degrees and 36 degrees, respectively.



(a) (b)

Figure 27: The reconstructed model of the British Bioscope.

Consider the model in Figure 27. The building was modelled using two cuboids for the base structure (since two different heights were needed), a prism for the roof of the second cuboid, two cuboids for the two chimneys, three "bay window" primitives for the bay window, and two further primitives for the decorative architecture on the crest of the façade. This resulted in a total of 31 free model parameters, including those describing the two cameras.

Figure 28 illustrates the reconstructed model projected onto the original photographs. In general, the projected lines conformed very closely to the user observations as well as the visible geometry. The bay window and decorative geometry did, however, introduce a small amount of additional error into the system, and this is discussed fully in Section 6.2.1

As with the previous example, one problem experienced with this reconstruction was in determining where the building joined the ground. Again, the ground was sloped, but this time, it was possible to use a set of stairs visible in both photographs to determine where the front wall intersected the ground plane. It should be noted, that foundations built to ensure that a building is level are not considered for reconstruction, but only the actual building itself. Following this reasoning, in this example, we consider the bottom step to represent the end of the foundation and the beginning of the building.

(a) (b)

Figure 28: The two photographs of the British Bioscope, now with the reconstructed model projected onto them. The green lines are the edges of the model. Note how they conform closely to the original marked out lines (still partially visible in red).

The texturing process, the results of which are illustrated in Figure 29, involved synthesizing large portions of texture. Since most of the roof and side wall are entirely obscured, and smaller sections of the front wall are also hidden, extensive texture synthesis was required.

The side wall was textured by using multiple seeds that were taken from the visible portions. By doing this, a convincing "old paint" texture was achieved. To complete this texture, a window, which had been extracted from the visible portion of the side wall, was inserted at regular intervals. The second, hidden side wall used the same texture as the first, but with some minor modifications to give the appearance of uniqueness.

The roof was also textured using the texture synthesis procedure. The results of this are not, however, as good as the side wall, largely due to the very low quality of the sample. The visible portion of roof is seen from a grazing angle, and this results in a very low-detail texture, with a corresponding lack of detail in the synthesized texture. As with the side walls, the invisible face of the roof used the same texture as the visible face, again with some minor modifications.

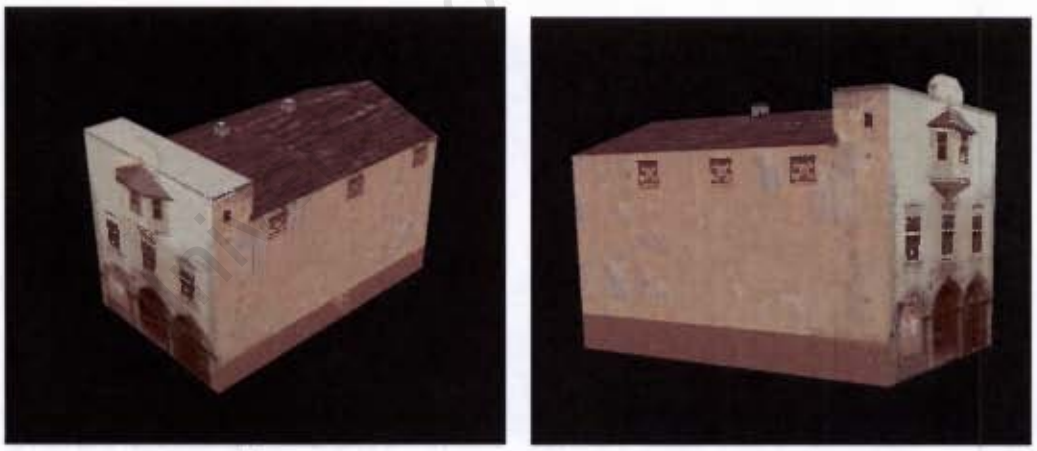
The front wall was the most complex face of the model to texture, as it contains a lot of detail, but is also obscured by a number of external features. Furthermore, the viewing angle of this wall is poor and as a result recessed features, such as windows, appear distorted.

The problem of texture detail being obscured was solved by simply erasing the invalid portions of the texture, and then using a combination of texture synthesis and feature insertion to repair those regions. Specifically, the left hand side of the front texture was obscured by a lamp post. This was removed, and replaced with a decorative brickwork coining feature, that had been extracted from the right hand side of the façade. The pillar box obscuring part of the lower half of the texture was also erased, and the missing region filled using texture synthesis and feature insertion.

The problem of distorted windows was solved by extracting a window, saving it to file, and repairing it using an external image editing application (by mirroring the left side onto the right). The window was then loaded as a new feature, and inserted over each of the distorted windows, greatly improving the visual appearance of the front texture.

The textures for the bay window were all in a usable condition, and only the invisible face had to be constructed. To achieve this, a base texture was synthesized, and then a window, extracted from the corresponding visible face of the bay window, was inserted. A similar strategy was used for constructing the missing faces of the roof and base of the bay window.

Finally, the remaining faces (chimneys and decorative roof-piece) used either original or synthesized textures. These faces are very small, and required no feature insertion.



(a) (b)

Figure 29: Two synthetic views of the reconstructed, textured model of the British Bioscope.

The overall result of this reconstruction is illustrated by Figure 29, and showcases the capacity of our reconstruction strategy.

6.1.3 A small residence from a single photograph

The final sample reconstruction is of a small residence, reconstructed from a single photograph. The building is architecturally simple, but reconstruction remains problematic as there is only one sample image (shown in Figure 30). Furthermore, the photograph, although yielding an excellent view of the front wall, presents very limited information pertaining to the side wall (and hence the depth) of the building.

Additional difficulties are caused by a degree of radial distortion evident in the image (a problem left as future work, and discussed in Chapter 7), and by the roof of the building which sags along the depth axis. These two characteristics result in additional error as, when marking out the edges of the building, it is impossible to draw a straight line on the image that matches exactly with what should be a linear feature.

(a) (b)

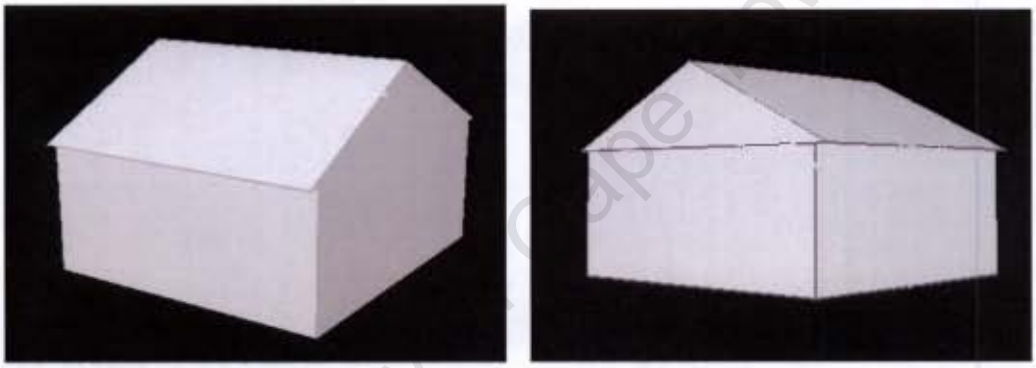
Figure 30: The single photograph used to reconstruct a small, District Six residence. (a) is the original input photograph, after the relevant edges have been marked out, while (b) is the photograph with the reconstructed model projected onto it from the perspective of the reconstructed camera. Note the error at the top right-hand corner of the roof.

The reconstruction of the geometry of the house was successful, and the field of view of the camera was established to be approximately 53 degrees. There were, however, two minor problems with the reconstruction. First, as illustrated in Figure 30(b), the top right-hand corner of roof, in the reconstructed model, fails to conform closely to the user observations in the image. This problem, discussed more fully in Section 6.2.1 appears to result from the sagging of the roof and the radial distortion present in the image mentioned above.

The second problem with the reconstruction is that the side wall appears to be longer than one

would initially guess. However, due to the very poor camera angle, it is impossible to determine visually how long that wall actually is, and the projected model does match up accurately with that portion of the image. Although this building, in common with many District Six houses, sits on a slope, this did not present any problems, as the border of the foundation is clearly visible in the photograph.

As illustrated in Figure 31, the building was modelled, very simply, as a cuboid for the base and a prism for the roof. The cuboid had two degrees of freedom (width and height — depth was fixed for scaling purposes), and the prism added a further two to the system (for its depth and height). Including the camera parameters, the entire model comprised only 10 free parameters.



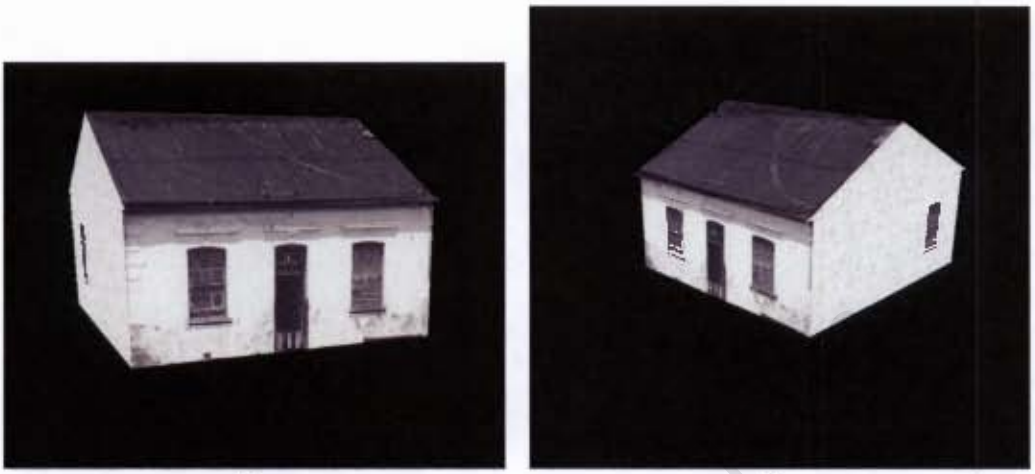
(a) (b)

Figure 31: Two views of the model of the house that was reconstructed from only a single photograph.

The texturing of the house was relatively simple, and two synthetic views of the textured model can be seen in Figure 32. For the front wall, the original texture from the photograph was of high fidelity and could be used directly. Similarly, the visible side of the roof was not obscured by any geometry, and the texture for this was automatically extracted without any difficulty (the viewing angle was, however, poor, so the quality of the texture did suffer). The hidden side of the roof was then textured using the same texture as the visible side.

Although the one side wall is entirely visible in the photograph, the angle is so poor that the extracted texture was unusable. To solve this problem, a new texture was synthesized from a paint sample taken off the front wall, after which a window, again extracted from the front wall, was inserted at the correct position in the texture. The synthesized texture was generated by inserting a number of seed textures into the image, resulting in a realistic paint texture. This same procedure was then followed to produce textures for the hidden side and back walls of the building.

This example represents a worst case scenario for reconstruction. However, the results remain



(a) (b)

Figure 32: Two synthetic views of the reconstructed, textured model of the small residence.

acceptable. Ideally one would want an additional photograph to ensure that the depth of the building is perceived correctly, but that was not possible in this case. The results of the texturing process were satisfactory, especially considering that some of the textures had to be entirely synthesized.

6.2 Analysis of results

In order to analyse the success of the reconstruction strategy, the accuracy of the geometry reconstruction, and quality of the texturing strategy must be independently assessed. To aid in this (in addition to the real world examples already presented) a synthetic reconstruction was undertaken. The remainder of this section consists of discussions on the geometry reconstruction and texturing aspects of this method of reconstruction.

6.2.1 Geometry Reconstruction

Overview

When considering the accuracy of the reconstruction of geometry from images, it is essential to understand that this technique simply fits a constrained model to a set of user defined observations. Indeed, at the geometry reconstruction phase, the only purpose of the photograph is to guide the user in marking out the observations.

As a result of this, any inaccuracies caused by lens distortion, poor image quality, or user input error will result in some degree of model error. Furthermore, where geometry cannot be exactly modelled by a set of primitive, the primitives then only represent an approximation of that geometry, and additional error may thus be introduced.

Although the objective function, described in Chapter 3, measures how the model deviates from the user's observations in a mean-squares fashion, we use a slightly different error metric when considering how closely the final model matches the observations. Rather than using mean-squared or root-mean-squared error, which is difficult to intuitively interpret, we use the area between the marked edge and the projected edge, normalised by the length of the marked edge, as our error metric. Effectively, this measure describes the distance (on average), in pixels, between the marked line and the projected model edge.

One example of where error is introduced into the system through a combination of poor image quality and geometric approximations is the roof of the small residence discussed in Section 6.1.3. In this case, the model edges are, on average, 1.14 pixels out, with a standard deviation of 1.23 pixels. However, much of this error can be attributed to a single edge at the top right of the roof of the building, which deviates from its observation by 4.32 pixels on average. This problem appears to stem from severe sagging of the roof (which causes it to be misaligned with the model) and a degree of radial distortion in the image. Discounting this edge, the model edges are, on average, 0.86 pixels out, with a standard deviation of 0.53 pixels.

In the case of the British Bioscope (depicted in Figure 28), the model generally conforms very closely to the user observations. The edges are, on average, only 0.71 pixels out, with a standard deviation of 0.46 pixels. Although this reconstruction has a low degree of error, there are still problems with the bay window and decorative geometry above the façade. For these edges, the average error rises to 0.96, with a standard deviation of 0.43. In this case, the additional error is clearly a result of the complex, curved geometry in the scene, as the model only approximates the actual structure.

Considering these problems, the three reconstructions presented above have an acceptable level of error. The quality of photographs is generally very poor, and it is often difficult to see exactly where the edges of the building are, which makes defining observations a difficult task. In addition to this, there is always some degree of lens distortion in the image, and as a result edges that are straight are often very slightly curved in the photograph, which further complicates the task of marking out observations.

In order to illustrate that in the ideal case (where there is no lens distortion, the camera focal length is known, and the edges are very clearly visible) the reconstruction framework will compute the parameters correctly, consider the synthetic case shown in Figure 33. This model was reconstructed from the two synthetic images shown, and all parameters are correct to within 1% of their original value. In this "ideal" case, the projected model edges conform, on average, to within 0.11 pixels of the user observations (with a standard deviation of 0.09 pixels).

(a) (b) (c)

Figure 33: The input images and output model of the synthetic test case. (a) and (b) are the computer rendered images of our synthetic case, while (c) is the recovered model. Note that in this case, the recovered model parameters conforms to the original to within 1%

Finally, the geometry reconstruction technique is quite time consuming, and modelling a reasonably complex building will generally take in the region of two hours. The optimizations run in a reasonable time, with the non-linear optimization taking less than a minute per iteration in even our most complex cases (generally between five and ten iterations are required for convergence). One issue to note here, is that the time taken by this optimization scales linearly with the number of user-defined observations, and quadratically with the number of free variables affected by each observation. In our examples, up to 50 observations were made, however, in an exceptionally complex case, with hundreds of observations, the running time of the algorithm would be substantially slower.

Problems

Although the geometry reconstruction technique used in this system was, on the whole, successful, there are a number of problems associated with it:

- **Highly detailed geometry.** Where there is very complex geometry that cannot be modelled by geometric primitives (such as decorative architecture), or where the resolution of the image does not permit the user to accurately distinguish subtleties within the geometry of the building, approximate models must be used. Although after texturing, the final results are often visually acceptable, this does cause a loss in geometric detail.
- **Curved surfaces.** Currently, this system is unable to handle curves and surfaces of revolution. Therefore, any curved surface must be approximated by polyhedral primitives, which, again, results in a loss of geometric detail.
- **Recessed surfaces.** Using this system it is difficult to model walls with recessed areas (at

windows, for example). Usually, such walls are modelled as flat surfaces, and the textures create the necessary effect, however, when a photo offers a very poor view of a wall, this does complicate the texturing process.

- **Uneven ground.** A very common issue in using this system, is that of uneven ground. Many buildings do not lie on a flat surface, and at times determining where the foundation ends, and the building begins, can be a difficult task. Generally, as illustrated by the real world examples above, it is possible to find features that allow one to solve this problem.
- **Low quality input images.** Where the input images are of a low quality, it becomes more difficult to achieve accurate reconstructions. This is because the quality of the reconstruction depends almost entirely on the quality of the observations.

6.2.2 Image-based Texturing

Overview

Determining the success of the texturing approach presented in this thesis is difficult, as what constitutes a "good" or "realistic" texture is highly subjective. Furthermore, it is also impossible to compare the textured model directly to the original photographs, as so much of each texture is regenerated, and large portions of the buildings are obscured from view in the photographs. This implicitly undermines the validity of any quantitative analysis.

However, a number of observations can be made about the components of this texturing approach. The synthesis procedure is effective; damaged textures were repaired, and new ones created, yielding results that appeared closely matched to the original sample textures. This was most successful for structured textures, such as the roof tiles in Figure 25, but also yielded satisfactory results for paint textures as exhibited in Figure 29. In general, this technique worked best if the new texture was seeded at a number of different points, and with a larger sample texture to work from.

The extraction and insertion of features were also largely successful, and allowed the user to modify the synthesized textures and ensure that they conformed closely to how the original surface of the building would have looked. Figures 25 and 29 illustrate clearly how inserted features, such as windows, add greatly to the quality of the final texture. In Figure 29, the window features are extracted, modified, and then reinserted to solve the indentation problem.

In order to best illustrate the success of our texturing scheme, consider Figure 34. The two images in Figures 34(a) and 34(b) are renderings of the British Bioscope using only the textures that were extracted from the input images. In Figure 34(a), none of the damaged texture material has been removed, while in Figure 34(b), all damaged material has been erased. Both of these renderings are clearly not in a usable state, and are, in contrast to those in Figure 34(c), of very poor quality.

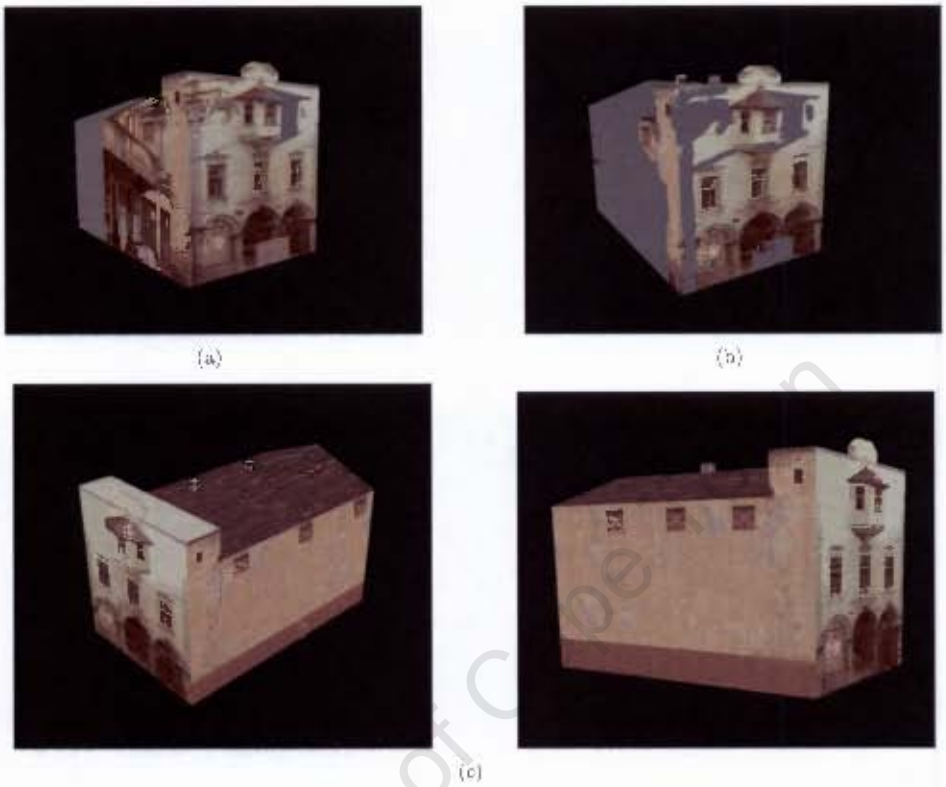


Figure 34: The Bioscope textured only with original material. (a) is a synthetic view of the British Bioscope, using all extracted texture material, while (b) is a view of the same, but with damaged material removed. To contrast this with the results of our texturing scheme, consider the two images in (c).

Problems

As with the geometry reconstruction, some problems were experienced with the texturing strategy:

- **Synthesis speed.** Possibly the biggest problem with this strategy, is that the texture synthesis procedure is slow. The running time of this procedure scales linearly with the size of the empty texture, the size of the sample texture and the size of the neighbourhood window (i.e., cubically in all three). Synthesizing a large texture from a large sample can take up to half an hour.
- **Synthesis errors.** Occasionally, the texture synthesis procedure will produce poor textures. This happens when the structure of the texture is lost, or the synthesis procedure starts repeating texels on a regular basis — a full discussion of these problems can be found in Efros and Leung [1999]. These problems were generally prevented, however, by inserting multiple seeds into the new texture.
- **Poor input textures.** Where the input texture was very poor, it was difficult to produce

acceptable results, as the low quality of such textures is propagated through the synthesis procedure.

6.2.3 Final Analysis

In general, for the problem at hand, both aspects of the architecture reconstruction strategy presented here have been successfully applied. Using the geometry reconstruction framework, a user is able to reconstruct, with an acceptable degree of accuracy, models of buildings, from as few as one photograph. Following on from this, by using our texturing strategy presented above, perceptually realistic textures, based on the input images can be constructed and applied to the reconstructed model.

One consideration, clearly evident in both aspects of the reconstruction strategy, is that the quality of the results is highly dependant on the quality of the input material. Where the original photographs are of a poor quality, it becomes harder to produce accurate models with high quality textures.

Chapter 7

Conclusion and Future Work

Having presented results in the previous chapter, it is now possible to draw a conclusion to this thesis, and discuss any future research opportunities that it may have created.

7.1 Summary of Work

In this thesis we set out to achieve two major goals. The first was to develop a geometry reconstruction framework, capable of reconstructing three-dimensional models of instances of District Six architecture from sets of photographs. The second was to design and implement a texturing scheme that would use the original photographs to texture the reconstructed model.

Ideally, the geometry reconstruction framework should be able to reconstruct buildings using small sets of uncalibrated images from the District Six Museum photographic archive, where in the worst case scenario, such a set contains only a single image. Although the reconstruction framework need not handle the camera calibration issue directly, methods to overcome this problem are required. As output, the reconstruction framework should produce a polygonal model of the building in question, which could subsequently be textured using the photo-based texturing scheme.

The texturing scheme should be capable of texturing a reconstructed model by using the original photographs as the source for texture material whenever possible, and otherwise synthesizing new texture material, based on small sample textures from the original photographs. User-interactive techniques, to facilitate the selection of the source material used in textures, as well as the texture synthesis, should be investigated.

To achieve the first goal, a reconstruction framework based on work by Debevec [1996] was implemented. This framework is able to reconstruct, with a high degree of accuracy, instances of District Six architecture, using small sets of uncalibrated photographs. Although unable to handle

certain more complex architectural structures, such as decorative geometry and curved surfaces, the framework is generally able to successfully reconstruct the geometry of the given buildings. The framework outputs models of reconstructed buildings to a file that is subsequently imported by the texturing utility.

When considering the problem of calibration, it was found that the focal length of a camera can be estimated, with a reasonable degree of accuracy, by simply identifying a cuboid in the image and attempting to reconstruct it. When the focal length of the camera is correct, the cuboid can be reconstructed with very little overall error.

The problem of texturing the model is solved by extracting, for each polygon of the model, all relevant texture information from the input photographs, and then relying on user input to determine what texture information to use for each polygon. The texturing utility allows the user to either apply the extracted texture information directly, or generate new textures through a texture synthesis algorithm described by Efros and Leung [1999]. Newly synthesized textures can be completed by inserting features previously extracted from the original photographic material.

This texturing approach is generally successful, and produces convincing, realistic textures. However, the quality of the output from this scheme is largely dependant on the quality of the input photographs, and when only poor textural information is available, this is reflected in the final results.

To illustrate the use and success of the utilities that had been developed, three real-world samples of District Six architecture were reconstructed from small sets of photographs, and one synthetic case was reconstructed using computer rendered images. The real-world cases comprised one large District Six residence, with a set of two greyscale photographs, the British Bioscope, with a set of one colour and one greyscale photographs, and one small residence, with only a single, greyscale photograph.

The three real-world tests showcase the capabilities of the reconstruction technique presented in this thesis, but also illustrate some of the difficulties associated with reconstructing buildings from photographs of poor quality. These examples show that the geometry reconstruction framework is able to produce good results even when only one input photograph is used, and that the texturing scheme is able to handle a wide variety of situations while still producing acceptable results. The synthetic case was then used to verify the correctness of the geometry reconstruction framework, by showing that, if provided with "perfect" input, it produces "perfect" results.

Finally, a critical analysis of both aspects of the reconstruction technique was presented, focusing on where the techniques were successful as well as which sorts of scenarios presented problems. Although generally very good, the geometry reconstruction framework has difficulties handling some complex geometry, and the quality of results decreases with the quality of the input photographs. It was further established that the texturing scheme, although able to handle most situations effectively, is, similarly, adversely affected by images of poor quality.

7.2 Future Work

Although our architecture reconstruction technique is able to successfully handle a wide array of situations, there are a number of areas in which it fails, as well as many ways in which it could be improved. All of the future work opportunities that have become apparent fall into either the geometry reconstruction or texturing categories.

7.2.1 Geometry Reconstruction

Aside from the problems associated with poor image quality, the difficulties encountered when using the geometry reconstruction framework presented in this thesis, generally involve complexity in the architectural geometry. Primarily, these are:

- Highly detailed geometry.
- Curves and surfaces of revolution.
- Surfaces with recessed features.

Clearly, modifying the geometry reconstruction framework to handle these situations is the most obvious step forward. Although handling detailed geometry, given the generally poor quality of the input data, is an exceptionally difficult problem, managing curved surfaces and surfaces with recessed features is certainly possible. Indeed, Debevec et al. [1996] asserts that curved surfaces can be parametrised in a similar fashion to the geometric primitives used in this system, and this should certainly be implemented at a later stage.

A further opportunity for potential future work is the user interface to the geometry reconstruction framework. Currently, the reconstruction of a reasonably complex scene can take a number of hours to perform. While automation of the reconstruction would be problematic (due to the quality and variety of input data), it would be worth investigating methods of speeding up the modelling process through an efficient user interface.

In this work, we determine only the focal length for each image used. Investigating and implementing more comprehensive camera calibration techniques is one final avenue for future work.

7.2.2 Photo-based Texturing

As with the geometry reconstruction framework, one of the major problems associated with the texturing scheme presented in this thesis is the poor quality of input data. Aside from this, however, the biggest flaw lies in the speed of the texture synthesis procedure.

Clearly, one path worth exploring further is improving the efficiency of the texture synthesis procedure used in the texturing scheme. This could be approached either by optimizing the performance of the current synthesis approach, or through the implementation and use of other, faster synthesis procedures, such as that described by Wei and Levoy [2000].

A final improvement to texturing would be the better use of available textures. Currently, where two photographs have different chromatic schemes, unless both are converted to greyscale, only one of the photographs will be used for extracting textural information. Techniques, such as that described by Welsh et al. [2002] are able to convert greyscale images to colour, and investigating the use of such techniques in texturing could be beneficial to the overall architecture reconstruction technique.

References

Y. I. Abdel-Aziz and H. M. Karara. Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. In *Symposium on Close-Range Photogrammetry*, pages 1–18, 1971.

A. Baumberg. Reliable feature matching across widely separated views. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 774–781, 2000.

R. Benjemaa and F. Schmitt. Fast global registration of 3d sampled surfaces using a multi-z-buffer technique. In *Proceedings of the International Conference on Recent Advances in 3D Digital Imaging and Modelling*, pages 113–120, May 1997.

J.-A. Beraldin, S. F. El-Hakim, and F. Blais. Performance evaluation of three active vision systems built at the national research council of canada. In *Optical 3-D Measurement Techniques III*, pages 352–361, 1995.

J.-A. Beraldin, S.F. El-Hakim, G. Godin, V. Valzano, A. Bandiera, and D. Latouche. Virtualizing a byzantine crypt by combining high-resolution textures with laser scanner 3d data. In *International Conference on Virtual Systems and MultiMedia*, pages 3 – 14, September 2002.

R. Bergevin, M. Soucy, H. Gagnon, and D. Laurendeau. Towards a general multiview registration technique. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):540–547, 1996.

Fausto Bernardini and Holly Rushmeier. The 3d model acquisition pipeline. *COMPUTER GRAPHICS forum*, 21(2):149–172, 2002.

E. Berndt and J. Carlos. Cultural heritage in the mature era of computer graphics. *Computer Graphics and Applications*, 20(1):36–37, 2000.

Paul J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

Thomas Buehrer, Zhang Li, Armin Gruen, Clive Fraser, and Heinz Ruther. Photogrammetric reconstruction and 3d visualisation of bet giorgis, a rock-hewn church in ethiopia, 1999.

R. Cipolla, T. Drummond, and D. Robertson. Camera calibration from vanishing points in images of architectural scenes. *British Machine Vision Conference*, 4(2):127 – 139, March 1999.

S. Cornou, M. Dhome, and P. Sayd. Architectural reconstruction with multiple views and geometric constraints. *The British Machine Vision Association*. 2003.

Paul Debevec, Yizhou Yu, and George Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *9th Eurographics Rendering Workshop*, 1998.

Paul E. Debevec. *Modeling and Rendering Architecture from Photographs*. PhD thesis, University of California at Berkeley, Computer Science Division, Berkeley CA, 1996.

Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH 96*, pages 11–20. ACM Press, 1996.

Anthony Dick, Phil Torr, and Roberto Cipolla. Automatic 3d modelling of architecture. In *British Machine Vision Conference*, pages 372–381. British Machine Vision Association, 2000.

Anthony Dick, Phil Torr, and Roberto Cipolla. Modelling and interpretation of architecture from several images. *International Journal of Computer Vision*, 60(2):111–134, 2004.

Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH 2001*, pages 341–346. ACM Press, August 2001.

Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, Corfu, Greece, September 1999. IEEE Computer Society.

Sabry F. El-Hakim, J.-Angelo Beraldin, Michel Picard, and Antonio Vettore. Effective 3d modeling of heritage sites. In *4th International Conference on 3-D Digital Imaging and Modeling*, pages 302–309, 2003.

Sabry F. El-Hakim, J.-Angelo Beraldin, Michel Picard, and Guy Godin. Detailed 3d reconstruction of large-scale heritage sites with integrated techniques. *IEEE Computer Graphics and Applications*, 24(3):21–29, 2004.

Andrew W. Fitzgibbon and Andrew Zisserman. Automatic camera recovery for closed or open image sequences. In *5th European Conference on Computer Vision*, volume 1, pages 311–326. Springer-Verlag, 1998.

R. Fletcher. *Practical Methods of Optimization*. John Wiley, 1987.

James Foley, Andries van Dam, Steven Feiner, and John Hughs. *Computer Graphics: Principles and Practice*. Addison-Wesley Professional, 1995.

A. Gardner, C. Tchou, T. Hawkins, and P. Debevec. Linear light source reflectometry. In *SIGGRAPH 2003*, pages 749–758. ACM Press, 2003.

P. Gill and W. Murray. *Practical Optimization*. Academic Press, 1981.

R. C. Gonzalez and R. E. Woods. *Digital Image Processing Using Matlab*. Prentice Hall, 2003.

Armin Gruen, Fabio Remondino, and Li Zhang. Image-based reconstruction of the great buddha of bamiyan, afghanistan. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXIV(5/W12), 2002.

GSLDocs. Gnu scientific library, 2007. URL `www.gnu.org/software/gsl/ visited:14/05/2007`.

J. Heikkila and O. Silven. A four-step camera calibration procedure with implicit image correction. In *Computer Vision and Pattern Recognition*, pages 1106–1112. IEEE Computer Society, 1997.

Mark Lehner. The giza plateau mapping project, 1992. URL `http://oi.uchicago.edu/research/projects/giz/ visited:14/05/2007`.

Wim Leler and Jim Merry. *3D with HOOPS*. Addison-Wesley, 1996.

Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In Kurt Akeley, editor, *SIGGRAPH 2000*, pages 131–144. ACM Press, 2000.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, 1999.

Byong Mok Oh, Max Chen, Julie Dorsey, and Fredo Durand. Image-based modeling and photo editing. In *SIGGRAPH 2001*, pages 433–442. ACM Press, 2001.

OpenGLDocs. The opengl api, 2007. URL `www.opengl.org visited:14/05/2007`.

J. R. Parker. *Algorithms for Image Processing and Computer Vision*. Wiley, 1996.

Ken Perlin. An image synthesizer. In *SIGGRAPH 85*, pages 287–296. ACM Press, 1985.

Michelle Pillers. Mead renaissance of the 90's. *Cadence Magazine*, 3, 1998.

G. Roth. Registering two overlapping range images. In *Proceedings of the 2nd International Conference on 3D Digital Imaging and Modeling*, pages 191–200, May 1999.

Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 409–416. ACM Press, 2001.

Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH 92*, pages 249 - 252. ACM Press, 1992.

Ying Shan, Zicheng Liu, and Zhengyou Zhang. Model-based bundle adjustment with application to face modeling. In *IEEE International Conference on Computer Vision*, volume 2, pages 644–651. IEEE Computer Society, 2001.

Julian Loyd Smit. *Three Dimensional Measurements of Textured Surfaces using Digital Photogrammetric Techniques*. PhD thesis, University of Cape Town, 1997.

Andre Streilein and Markus Nierderost. Reconstruction of the disentis monastery from high resolution still video imagery with object oriented measurement routines, 1998.

Camillo J. Taylor and David J. Kriegman. Minimization on the lie group so(3) and related manifolds, 1994.

Camillo J. Taylor and David J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(11):1021–1032, 1995.

Bill Triggs, Philip McLauchlan, Richard Hartley, and Andrew Fitzgibbon. Bundle adjustment – A modern synthesis. In W. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag, 2000.

F. van den Heuvel. Estimation of interior orientation parameters from constraints on line measurements in a single image. *International Archives of Photogrammetry and Remote Sensing*, 32 (5/W11):81–88, 1999.

Frank A. van den Heuvel. Object reconstruction from a single architectural image taken with an uncalibrated camera. *Photogrammetrie Fernerkundung Geoinformation*, pages 247 – 260, 2001.

Jana Visnovcova, Armin Gruen, and Li Zhang. Image-based object reconstruction and visualization for inventory of cultural heritage. In *Electronic Imaging and the Visual Arts*, volume 26, pages 118–122, 2001a.

Jana Visnovcova, Li Zhang, and Armin Gruen. Generating a 3d model of a bayon tower using non-metric imagery. *Asian Journal of Geoinformatics*, 2(1), 2001b.

Goerge Vosselman and Sander Dijkman. 3d building model reconstruction from point clouds and ground plans. *International Archives of Photogrammetry and Remote Sensing*, XXXIV(3/W4): 37–43, 2001.

Li-Yi Wei and Marc Levoy. Fast texture synthesis using three-structured vector quantization. In *SIGGRAPH 2000*, pages 479 – 488. ACM Press, 2000.

Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. Transferring color to greyscale images. *ACM Transactions on Graphics*, 21(3):277–280, 2002.

Josie Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor*. Addison-Wesley, 1994.

Marta Wilczkowiak, Edmond Boyer, and Peter Sturm. Camera calibration and 3d reconstruction from single images using parallelepipeds. In *Proceedings of the 8th International Conference on Computer Vision, Vancouver, Canada*, volume 1, pages 142–148. IEEE Computer Society Press, Jul 2001.

wxDocs. The wxwidgets api, 2007. URL `www.wxwidgets.org visited:14/05/2007`.

Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence, VOL 22, NO. 11*, 22(11):1330 – 1334, 2000.