

**POST-PROCESSED SHADOW DETERMINATION  
FOR  
COMPOSITION OF DEPTH IMAGES**

By

Russell W. Krywolt

B. Sc. (Computer Science) University of Lethbridge

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF  
THE REQUIREMENTS FOR THE DEGREE OF  
MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES  
COMPUTER SCIENCE

We accept this thesis as conforming  
to the required standard

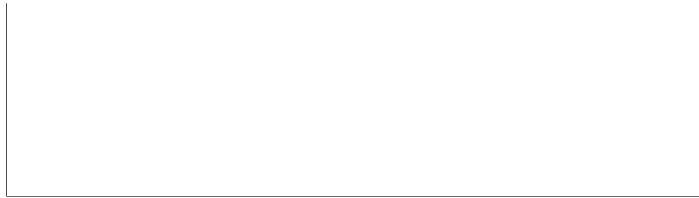
---

THE UNIVERSITY OF BRITISH COLUMBIA

October 1993

© Russell W. Krywolt, 1993

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.



Computer Science  
The University of British Columbia  
2075 Wesbrook Place  
Vancouver, Canada  
V6T 1Z1

Date:

OCTOBER 15, 1993

## **Abstract**

In the creation of computer generated pictures, many methods have been proposed to reduce the amount of time it takes to render an image or sequence of images. Efforts have been made to improve existing rendering algorithms and to create better ones, but the compositing method has received little attention. The compositing method breaks a scene up into parts which can be rendered separately which are then put together in such a way that the result is nearly indistinguishable from an image rendered from the complete scene. This can not only reduce the total time taken to generate an image sequence, but can also create composite images from others rendered by different methods. In 1984, Porter and Duff first introduced a well defined approach to computer image composition, and in 1985, Duff proposed an extension to the this method that employed depth images. Using depth images, visibility determination can be done by the computer, reducing the human effort required to produce the resultant image. The images produced are likely to be missing some elements that would be present if all objects were rendered in a single scene, such as shadows from objects in one image not being cast on objects from other images.

This thesis addresses the problem of modifying composited images such that the resultant picture contains shadows that would normally only be present in an image rendered from a scene containing all objects in both original pictures. A simple algorithm is developed to reconstruct the visible surface of objects in an image. The reconstruction is used to generate a shadow map for each image to be composited. The shadow map indicates where shadows would fall on one image were objects in the other image present when the original scene was rendered. The shadow map is then used to create shadows in

appropriate places on the original images, and these reshadowed images are composited to obtain the desired result. The application of these techniques to other areas is also investigated.

# Table of Contents

<b>Abstract</b>	<b>ii</b>
<b>List of Figures</b>	<b>vi</b>
<b>Acknowledgement</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background to Image Composition</b>	<b>8</b>
<b>3 Enhancements to simple depth composition</b>	<b>14</b>
3.1 Compositing with centre sampled pixels . . . . .	14
3.2 Generating antialiased images for composition . . . . .	18
3.3 Compositing pixels with equal depths . . . . .	22
<b>4 Visible surface reconstruction</b>	<b>26</b>
4.1 Simple surface reconstruction . . . . .	28
4.1.1 World coordinate retrieval . . . . .	28
4.1.2 Surface construction from world coordinates . . . . .	31
4.2 Image segmentation . . . . .	35
4.2.1 The ideal solution . . . . .	36
4.2.2 Possible approaches . . . . .	36
<b>5 The reshadowing process</b>	<b>40</b>
5.1 Creating the shadow map . . . . .	40

5.2	Filtering the shadow map . . . . .	42
5.3	Using the shadow map . . . . .	44
5.4	Reshadowing already shadowed images . . . . .	45
<b>6</b>	<b>Other applications and extensions</b>	<b>49</b>
6.1	CAR - Computer Augmented Reality . . . . .	49
6.2	What now? . . . . .	54
<b>7</b>	<b>Conclusions</b>	<b>56</b>
	<b>Bibliography</b>	<b>58</b>
<b>A</b>	<b>Depth Files</b>	<b>62</b>
A.1	Optik depth files . . . . .	62
A.2	Other uses of depth files . . . . .	64

## List of Figures

2.1	Table of 2D compositing operations from [port84] . . . . .	10
3.1	Division of pixel into four triangles and related depth values given by the dots at the corners . . . . .	15
3.2	Division of center sampled pixel into eight equal areas with associated depth values . . . . .	17
3.3	Sampling values within a single pixel . . . . .	19
3.4	A red cube composited with a gray sphere. The first image shows the overall appearance, the second shows a magnified section where colour sampling problems appears . . . . .	21
3.5	Pixels partially covering a square polygon with sampling at the lower left corner . . . . .	21
3.6	The red cube and gray sphere from <i>figure</i> 3.4, but with proper depth sampling used . . . . .	22
3.7	Interpenetrating squares of differing colours . . . . .	23
4.1	Reshadowing problems created by not segmenting images before performing surface reconstruction . . . . .	27
4.2	Viewing vectors for the scene . . . . .	29
4.3	Vectors for calculating world coordinates . . . . .	30

4.4	Reconstructing four triangles in one pixel from corner sampled depth data: black circles are known depth data, they grey circle is the interpolated depth data . . . . .	32
4.5	Reconstruction of triangles for centre sampled depth data: black circles are known depth values, grey circles are interpolated depth values, arrows show which known values contribute to the interpolated values . . . . .	33
4.6	Edge problems when reconstructed objects slope sharply away from the camera. Original view is on the left, 90 degree rotation on the right. . . .	34
4.7	Edge problems for sharply sloped objects. Left shows standard reconstruction, right shows reconstruction where sampled depth values are assigned to all parts of the pixel. . . . .	35
4.8	Original image . . . . .	37
4.9	Associated depth map shown as greyscale . . . . .	38
5.1	A shadow map created by the modified raytracer . . . . .	42
5.2	A shadow map after filtering has been performed . . . . .	43
5.3	The result of applying a weighted filter to the shadow map . . . . .	44
5.4	Rendered and composite images of the same scene. The rendered image is on the bottom . . . . .	46



## **Acknowledgement**

I would like to thank my supervisor, Alain Fournier, for providing many interesting problems for me to work on, and for his understanding when I spent time on problems and procedures not related to my thesis. In spite of having many demands on his time, he was always able to assist me when necessary. My thanks to David Forsey, not only for being my second reader, but also for encouraging me to explore all of the aspects of computer graphics that I found interesting. I would especially like to acknowledge the work and effort put in by everyone in the Computer Science Department in order to create the pleasant and relaxed research atmosphere of which I have become so fond.

Russell Krywolt

October 1993.

# Chapter 1

## Introduction

The generation of computer created images has always been limited by the speed at which such pictures can be rendered. While improvements in hardware have reduced the time it takes to render an image by orders of magnitude, the scenes that are being created today are vastly more complex than those made even five years ago. As a result, the increased computing power is being used to its fullest, and the average rendering time for a typical scene today has not been reduced in proportion to the speed increase of computers, and has remained fairly constant for those images with a high complexity. Since the trend of creating more complex images as faster hardware becomes available is likely to continue into the foreseeable future, computer graphics researchers have turned to other methods to attempt to reduce rendering times. These efforts can be grouped into two general categories, the first being developing algorithms and strategies to reduce the computational complexity of rendering a scene or series of scenes, and the second splitting the rendering effort into parts, rendering only those portions of the scene that change with time, and then *compositing*, or assembling the parts of the images into a coherent whole. The second method is obviously better suited to creating series of images that are temporally related (an animation) rather than single pictures, while the first method can be applied to both still frames and animated sequences. It is for this reason that most of the effort put into decreasing overall rendering times has been directed towards reducing the number of calculations done by a computer.

Despite the drawbacks of the compositing method when applied to single images, animation rendering times may be greatly affected. Since in many animations there is a considerable amount of the image that remains static as time progresses, the computer need only render the unchanging portion (or *background*) of the animation once, storing it, rendering the parts of the scene that change from one frame to the next, then compositing the two portions together on a frame by frame basis to produce the animation. This method is best illustrated by looking at any of the Walt Disney classic animated films, such as *Snow White and the Seven Dwarves*, *Pinocchio*, and others. In these films, vast and elaborate backgrounds were painstakingly created. Trees were painted in such detail that veins on leaves could be clearly seen, and features of other background elements were similarly detailed. The animated characters were then created a frame at a time on clear celluloid, called an animation *cell*. Each cell was then placed over the background in the proper position, and photographed to create a single frame of film. Similarly detailed foreground cells could be placed over the background and character if needed, creating a multi-layered scene where the motion was restricted to few cells in different layers. This process allowed the animation to be done much faster than if all the objects in every frame were repainted each time, as well as letting the artists create much greater detail in the background than would have been otherwise possible.

Another use for compositing is to form a single image from several images rendered by different processes. General purpose modeling and rendering software packages are based upon established algorithms, and are thus not often useful in a research environment. Current research can produce innovative modeling or rendering methods that just cannot be done with commercial software. Most often these creations are of a singular nature, that is, they are small pieces of software only intended to illustrate new ways of doing things. Many times the results of this research will somehow be used by other research based software, or perhaps by commercial software. As an example of this,

the hierarchical B-spline surface modeler used in [fors88] has proven to be such a useful modeling tool that it is often used at Imager, the computer graphics laboratory at the University of British Columbia, to create objects that are then exported to a commercial animation system. This is done by approximating the B-spline surface by a triangular mesh. While this mesh representation is fairly accurate, it is not perfect, and meshes of even simple B-spline objects can be large, slowing down both the modeling and rendering of the commercial software. Depending on the use of the B-spline surfaces, in many cases it would be easier and more accurate to render the B-spline object without having to convert it to a triangular mesh, and then composite it into an image rendered from the scene created by the commercial package. This use of the compositing technique would allow the creation of complex images that would otherwise be difficult to make.

The first attempt to systematize the application of this method to computer graphics was reported by Porter and Duff in 1984 [port84]. They took the multi-layer approach of Disney and applied it to computer graphics, enabling animators to render those parts of an animation that could be considered to belong to the same layer, if they were cells, render them separately, and then merge them such that the each rendered frame (from the 'top' layer down) obscured the one below it, but only where pixels held interesting information <sup>1</sup>. In order to enable the computer to decide what information in a frame was considered important, Porter and Duff used what they termed the *alpha channel* or  $\alpha$ -channel, which is the computer analogue of the matte signal used in video compositing. The  $\alpha$ -channel can be thought of as a separate image of the same size as the original, but having a white pixel at the same coordinates as the image for a pixel that contained interesting information, and a black pixel where the image had no important colour information. The  $\alpha$ -channel value could also take on intermediate values between 1

---

<sup>1</sup> *Interesting information* in this case refers to any area of the frame the animator decided was important, whether it contained colour information or not

(white) and 0 (black) to indicate a transparency or partial coverage of a pixel in the original image. If a pixel with an intermediate  $\alpha$  value was composited over top of another pixel, the resultant image would contain a pixel that was a blend of both pixel colours, with the upper pixel contributing  $\alpha$  times its colour, the lower  $(1 - \alpha)$  times its colour. Porter and Duff also defined several other compositing operations not normally used in cell animation, such as all boolean combinations of images ( **and**, **or**, **not**, **xor**, etc) as well as other operators that combine the upper and lower layers in various ways. These operations, however powerful, are not always adequate. For example, if a moving character is supposed to be weaving in and out of various parts of the scene, many layers had to be constructed, and their operations specified exactly, in order to get the right look and feel for the final animation. This time consuming process would be simplified if all of the static portions of the animation could be rendered at once, rather than in layers, and have the moving parts of the scene composited into it, with the computer taking care of any visibility problems. This can be done using a modified version of the popular Z-buffer visibility determination algorithm, as illustrated by Duff in [duff85]. Duff's method requires that for each pixel in a rendered image, a corresponding depth value is stored. This depth value represents (in some form) the distance the pixel is from the eye plane. When compositing images, the depth information of corresponding pixels in each image is compared, and the pixel with the smallest depth is displayed in the resultant image. This allows parts of animations to be composited easily, with some increase in storage.

While this method provides an easy way to composite images without much user intervention, the images produced can lack some elements that would be present if the models of the two scenes were combined and then rendered. In many cases, if a scene was rendered all at once, shadows from some objects would fall on other objects, but if these shadowing objects were rendered separately from the rest of the scene and then

composited, those shadows would not appear in the resultant image. Likewise, reflections present in a rendered image may be absent in a composited image. This thesis will focus upon the former problem, presenting a solution that will allow the *reshadowing* of images to occur when compositing to give the composite image the same shadows as would appear in a rendered image of the same scene.

When considering the complete reshadowing problem, many aspects of it must be solved before a complete solution is achieved. Because the scope of this thesis is insufficient to offer a complete solution to each and every aspect of the general problem, some restrictions are placed on the images to be reshadowed and composited. When solutions to the various problems that necessitate the restrictions are found, the restrictions can be removed and the techniques described in the thesis can be directly applied to help solve the complete problem.

The difficulty of discerning what objects are present in a scene (detailed in *Chapter 4*) gives rise to the first restriction. If two images are to be reshadowed such that shadows from one scene are to fall on the other, but *not* vice versa, then the scene that is causing the shadows must be made up only of non-occluding objects, but the other may be of arbitrary complexity. If the reshadowing is to be mutual, then both images must contain only non-occluding objects<sup>2</sup>. The second restriction on images is that all must have the same viewing parameters (such as image dimensions, eye position, eye roll angle, and such). The last restriction requires that all lighting parameters be the same (that is, number of lights, position, colouration etc.).

Since the initial restriction placed on the images has two cases, the outline of the solution is slightly different. In the first case, where only one image is used to reshadow

---

<sup>2</sup>Consider the case where a ball moves above a plane such that the ball never occludes the plane, but casts a shadow. If the ball and plane were two separate images that were to be reshadowed, only the image of the plane need be reshadowed. Again, please see *Chapter 4* for a complete explanation of why this is necessary.

the other, it is first necessary to get some notion of the original three-dimensional shape of the objects in the restricted (non-occluding) scene. To do this, world coordinates for each visible pixel of every object are retrieved using the stored depth data and existing ray-tracing techniques. These world coordinates are used to create a simple triangulated description of the visible surface of each object. Once this surface description is created it is passed to a modified ray-tracing renderer. If there are no problems with the reconstruction, a shadow map is created by the ray-tracing software. This shadow map is an image with the same dimensions as those being reshadowed but with all pixels having a full white value. The world coordinates of each visible pixel in the other (arbitrarily complex) scene are retrieved by the same method as used in the non-occluding scene, but without reconstruction. From each of these coordinates, the ray-tracer sends a ray to each light in the scene (since the assumption is that light positions are known). If any ray intersects a triangle in the reconstruction of the non-occluding image, a black pixel is placed in the shadow map at the same location as the pixel (from the arbitrarily complex image) whose world coordinate was the origin of the ray. The shadow map is filtered to smooth sharp intensity transitions, and *overlaid* on the arbitrarily complex image. This overlaying causes pixels in the image to have their intensity reduced in proportion to the amount of black present in the corresponding pixel of the shadow map. This is the actual *reshadowing* step, and once performed, the two images are composited (using the modified depth composition method discussed in *Chapter 3*) to give the final result.

The second case is solved in a similar manner, but the process is repeated for both images. Thus, the first image is initially treated as the complex image (even though it is not), and reshadowed. The images then reverse roles, the second image is reshadowed, and the images are composited.

Both methods of solution are discussed in detail in this thesis, and the results of their implementations are presented. The more difficult problems facing the solution of

the general case are discussed, but not implemented. Applications of this technique to the merging of real video and computer generated images are also discussed, along with related work currently being undertaken. Conclusions are made about the suitability of the developed methods to the general problem, and directions for future work are outlined. Finally, other applications of the depth compositing algorithms, along with other uses for depth images, are discussed and implemented. It should be noted that all of the steps mentioned previously are implemented as separate programs. While this makes the process more cumbersome than if everything was automated, it allows flexibility in the choice of other methods that may be implemented and tested with the system. Also, should any other uses be found for the parts of the reshadowing process, it enables easy exporting of the desired results without having to rewrite a large section of a big program.



## Chapter 2

### Background to Image Composition

The use of image composition techniques in computer graphics has evolved for a number of reasons. Composition provides a simple way to reduce the total time taken to produce an animation by allowing static parts of an animation to be rendered only once, and having the moving components rendered as needed, then put together to make the final scene. Using image composition, it is also possible to combine images rendered using different methods into a coherent whole. Since many tools that exist for specific purposes do not work in conjunction with complex general purpose renderers (or other tools), composition allows effects from all of these sources to be combined in an image or animation. Computerized image composition also mimics the sophisticated (and sometimes expensive) video composition tools used in video post-production, allowing computer artists and animators access to the effects without having to purchase separate equipment.

While image composition has been in use for a long time in the film and video industry, most people who wished to do computer compositing ended up writing specialized software to do specific tasks. In 1984, Thomas Porter and Tom Duff put forward an explicit method for compositing computer generated images ([port84]). This method relied upon the use of the  $\alpha$ -channel. The amount by which an object covers a pixel is determined by the  $\alpha$  value of that pixel. If pixel  $p$  in image  $a$  had an  $\alpha$  value of 1,  $p$  would be completely covered by this object. If  $p$  in  $a$  had an  $\alpha$  value of 0, then no objects would cover that pixel. Likewise, if the  $\alpha$  value of  $p$  were between 0 and 1, the pixel

would be partially covered by some object, and the ratio of covered to uncovered pixel area would be  $\alpha_a/(1 - \alpha_a)$ . When compositing a pixel from image  $b$  with the pixel  $p$  in  $a$ , Porter and Duff make the assumption that the object partially covering  $p$  in  $b$  will cover the object partially covering  $p$  in  $a$  by the ratio  $\alpha_b/(1 - \alpha_b)$ , since if the pixel coverages from  $a$  and  $b$  were simply added, it would be possible to get  $\alpha$  values of greater than one. The  $\alpha$  value of a pixel is also used when that pixel is fully covered by an object that is semi-transparent, with  $\alpha$  indicating the amount of the background colour that is blocked by the pixel, while  $(1 - \alpha)$  is equal to the amount of the background colour allowed through the pixel. While this dual interpretation of the  $\alpha$  value may not seem proper, Porter and Duff show that they are equivalent for compositing purposes. Emphasis is placed on the multiplication of the actual colour information by the  $\alpha$  information for each pixel before compositing so that the alpha value of the resulting composite pixel can be used directly to determine its colour. If this is not done, other, more complex manipulations would have to be performed at a later stage in the compositing process.

Using this information, Porter and Duff define operations to be performed when compositing pixels. These operations are summarized in the *figure 2.1* (for a corresponding pixel in image  $a$  and  $b$ , and colours  $rgb_a$  and  $rgb_b$  respectively).

From *figure 2.1*, the operation of  $x$  **over**  $y$  means that the output pixel will be covered as if  $x$  was placed in the foreground over  $y$ . Likewise,  $x$  **in**  $y$  means that portion of  $x$  that would be obscuring a portion of  $y$ , were they composited using the **over** operator. The operator  $x$  **out**  $y$  results in that part of  $x$  being shown that would *not* cover  $y$  if the **over** operator was used. Using  $x$  **atop**  $y$  is equivalent in coverage to using  $x$  **over**  $y$  and subtract the coverage given by  $x$  **out**  $y$ . Similarly  $x$  **xor**  $y$  is the same as taking  $x$  **over**  $y$ , then removing the portion covered by  $x$  **in**  $y$ . To get the colour of the output pixel, the  $rgb_a$  value is multiplied by  $F_a$  and added to the colour obtained by multiplying  $rgb_b$  by  $F_b$ . The final  $\alpha$  value is obtained by adding  $F_a$  to  $F_b$ .













operation	diagram	$F_A$	$F_B$
<i>clear</i>		0	0
<i>A</i>		1	0
<i>B</i>		0	1
<i>A over B</i>		1	$1-\alpha_A$
<i>B over A</i>		$1-\alpha_B$	1
<i>A in B</i>		$\alpha_B$	0
<i>B in A</i>		0	$\alpha_A$
<i>A out B</i>		$1-\alpha_B$	0
<i>B out A</i>		0	$1-\alpha_A$
<i>A atop B</i>		$\alpha_B$	$1-\alpha_A$
<i>B atop A</i>		$1-\alpha_B$	$\alpha_A$
<i>A xor B</i>		$1-\alpha_B$	$1-\alpha_A$

Figure 2.1: Table of 2D compositing operations from [port84]

This approach is conceptually similar to the way the Disney multiplane camera works ([levo77]), where every image is treated as a layer in the compositing step. Thus to create an image where an object is interpenetrating another object, the images involved must be constructed in such a way that when composited, the result looks correct. The extra effort in designing the scenes for composition was both time consuming, and not easily done for some rendering software. To circumvent this extra work, Duff ([duff85]) put forward a way of compositing images so that the computer performs the object visibility calculations. The modified Z-buffer algorithm requires that some extra information be stored with the colour and alpha information of the image. For every pixel in the image, the depth value of that point (or the distance from the eye plane) is stored and used for visibility calculations. Given two images  $x$  and  $y$ , with depth values  $z_x$  and  $z_y$ , and colour and alpha values  $rgba_x$  and  $rgba_y$  at pixel location  $p$ , the pixel that has the smallest depth value is found by comparing  $z_x$  to  $z_y$ , and using the smaller value. The pixel with the smaller depth value then becomes the *foreground* pixel, (denoted by  $f$ ), and the other the *background* pixel (or pixel  $b$ ). If the depth values are equal, Duff chooses the pixel from the first input image as the foreground pixel, and the pixel from the second input image as the background pixel. The depth value of the output pixel is the depth value of the foreground pixel, while the colour of the output pixel is given by  $\beta(f \text{ over } b) + (1 - \beta)(b \text{ over } f)$ , where  $\beta$  is the fraction of the pixel covered by an object in  $f$ <sup>1</sup>. This method provides a very good way of creating one image from multiple input pictures with little effort. Some of the problems that show up in two-dimensional image composition, specifically those lighting effects not common to all input images, or those that the viewer would expect to see in a real scene but do not show up in a composited image (shadows, reflections, and other object-dependent light interactions), are still evident in the three-dimensional image compositing result.

---

<sup>1</sup>This is put in context of this thesis and explained more in *Chapter 3*

Much use has been made of the compositing method of [port84]. Almost every major production animation system in use today has a two dimensional image composition system capable of performing many useful operations on image sequences, such as cross dissolves, wipes, and so on, in addition to the basic compositing operations. Many of these animation package also include a three dimensional composition system based on [duff85], although most do not extend the method as done with two dimensional compositing. Some research has been done into other uses of the image composition technique, both in hardware and software.

Nakamae, Ishizaki, Nishita and Takita ([naka89]) proposed a way to render portions of an image separately and then use a three dimensional composite to correctly merge them into a single picture. Their hope was that by raytracing only those portions of the image where raytraced effects were desired and using scanline methods on the rest, that the rendering time for images would be reduced. While their method produced composite images nearly indistinguishable from singly rendered images of the same scene, the time component of their method, as was stated in their goal, was not analysed. Another rendering compositing method was presented by Jahami and Beigbeder in [jaha91]. Hardware implementation of compositing has been investigated for some time. Molnar, Eyles and Poulton ([moln92]) developed the PixelFlow system, a custom designed hardware architecture for image generation using image composition methods to combine various image elements produced by a network of hardware renderers. Mark Green and Chris Shaw ([shaw90]) described a parallel architecture for performing the three dimensional compositing in [duff85]. They designed a pipelined VLSI chip to implement the algorithms described by Duff. Earlier, Booth, Forsey and Paeth investigated hardware assistance and acceleration to the Z-buffer algorithm in [boot86]. Shaw and Green have also done other work on hardware composition, as described in [shaw89]. Other uses of compositing, and methods of doing it have been investigated by Nakamae, Harada,

Ishizaki, and Nishita in [naka86], as well as Foley and Kim ([fole87]). Compositing processes and systems have been looked at by Potmesil and Hoffert (FRAMES - [potm87]), and Nadas and Fournier (GRAPE - [nada87]). Variants of the Z-buffer algorithm have been developed by Salesin and Stolfi (the ZZ-buffer - [sale90]) and Haeberli and Akeley (the Accumulation buffer - [haeb90]).

While the various software and hardware methods of compositing have been useful, this thesis concentrates on the Duff method of [duff85]. The modifications made to the Duff three dimensional composition algorithm will be detailed in following chapters along with methods to use the depth data to assist in reshadowing composited images.

## **Chapter 3**

### **Enhancements to simple depth composition**

This chapter presents some modifications of the fairly robust method presented Duff in [duff85] that will improve the results. Three main areas of the Duff method can be examined. First, the output produced by a raytracer should be modified to calculate depth and colour at the centre of the pixel instead of at the lower left corner. Secondly, the generation of antialiased images and the related depth values are only discussed briefly in the Duff paper, but the methods to produce these should be more rigidly defined due to problems that arise when calculating depth for antialiased pixels. Also, the method given does not deal explicitly with compositing two pixels that have the same depth value. While this problem is not too serious when compositing single frames, it can result in some temporal aliasing problems when an animation is being produced.

#### **3.1 Compositing with centre sampled pixels**

In order to composite depth images, it is necessary to determine which pixels of each image should be displayed in the output image. This is done by comparing the depth value of corresponding pixels in each input image, and outputting the pixel with the smallest depth value. While this may seem sufficient, in practice the point sampled nature of the images will result in colour aliasing in the output image. These problems are overcome by averaging colour values of pixels in each input image based on the pixel depth values.

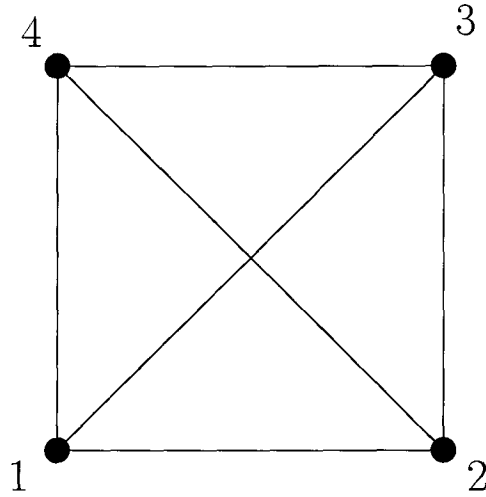


Figure 3.1: Division of pixel into four triangles and related depth values given by the dots at the corners

In [duff85] depth and colour values are assumed to have been sampled from the lower left corner of the pixel. While it is true that many renderers do this, an equal number sample from the centre of the pixel, and this creates a more complex situation than Duff presents. When the pixels in the image are sampled in the lower left corner, the values at every corner of every pixel are known <sup>1</sup>. This allows us to divide up the pixel into four triangles, each covering one-fourth of the pixel, and each having the depth value of the associated pixel corner (*figure 3.1*). By looking at the corners of the corresponding pixel in each composited image, it can be determined which pixel is nearer the camera, and thus which should be displayed in the final image. Given two images  $a$  and  $b$  and the pixel at location  $p$  in each image, if all four corners of  $p$  in image  $a$  have a smaller depth value than the corners of pixel  $p$  in image  $b$ , or all corners of  $p$  in  $b$  have a smaller depth value than all corners of  $p$  in image  $a$ , the output pixel of the composite image is taken directly from the appropriate input image. If, however, some corners of  $p$  in  $a$

---

<sup>1</sup>Except for the top and rightmost rows, which is why Duff stores an extra row of pixels off the top and right



are closer to the camera than in  $b$ , and some in  $b$  closer than in  $a$  (what Duff terms a *confused* pixel), the output pixel colour must reflect this mix, and so should be a blend of the pixel colours in  $a$  and  $b$ , proportionate to the amount of  $p$  that the pixels from each image cover. This method can be thought of as performing colour filtering using the depth data to determine the filter effects. For corner sampled images then, the filter used is not symmetric around the origin point.

Compositing with centre sampled pixels is done in much the same fashion, but some calculations must be modified, since for each pixel there is no knowledge about the depth at the corners. In keeping with the Duff method, consider two centre sampled images,  $a$  and  $b$ , and a pixel of interest at location  $p$  in each. Since  $p$  has eight neighbours, and the depth value at  $p$  is known only in the centre, the depth value of the neighboring pixels can be used as the depth at the appropriate place in  $p$ . If the pixels surrounding  $p$  are labeled from 1 to 8, as shown in *figure 3.2*, the depth values at the corners of  $p$  are the same as the depth values at pixels diagonally opposite  $p$  at the corner and the depth values at the sides of pixel  $p$  are the same as the pixels that share that side with  $p$ . With this interpretation, nine polygons of equal area can be constructed such that they cover  $p$  completely, each having the same depth value as the appropriate part of  $p$  (see 3.2). Using this, we can compute the portion of the pixel covered by  $a$ , by taking the ratio of polygons in  $p$  covered by  $a$  and dividing by the total number of polygons in the pixel, in this case, nine. If this amount is  $\beta$ , the output pixel will have  $\mathbf{rgb}\alpha = \beta(a \text{ over } b) + (1 - \beta)(b \text{ over } a)$ , with the depth value equal to the minimum depth value of  $a$  and  $b$ , where **over** is the operator used in [duff85]. As stated previously, this method can be thought of as a filtering operation using the depth data to determine the resultant colour information. The main difference between this and the previous filter is that this one is symmetrical around the origin. As may seem obvious, it would be possible to use the original Duff filter (or compositing operation) to composite centre sampled images, using the centre sampled points in place

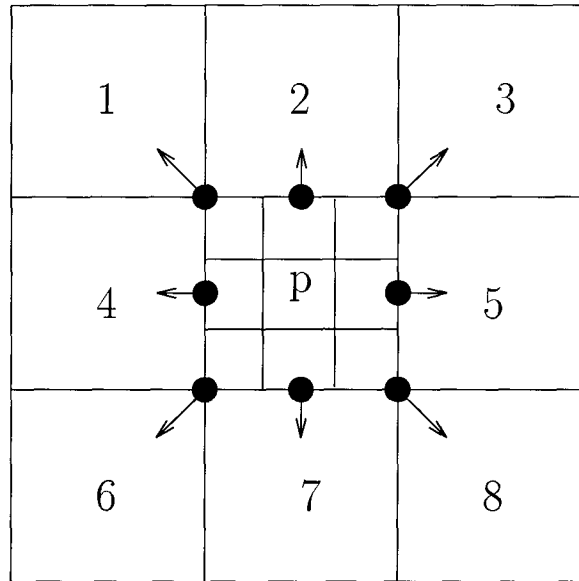


Figure 3.2: Division of center sampled pixel into eight equal areas with associated depth values

of the appropriate corner sample. Due to the methods used to produce antialiased colour information, this approach is not desirable, since elements present in all surrounding pixels may have already contributed to the colour of the pixel being composited. Filtering in the manner of the Duff paper takes only three surrounding pixels' information into consideration, while ignoring seven others that may contain information that should be reflected by the pixel being composited.

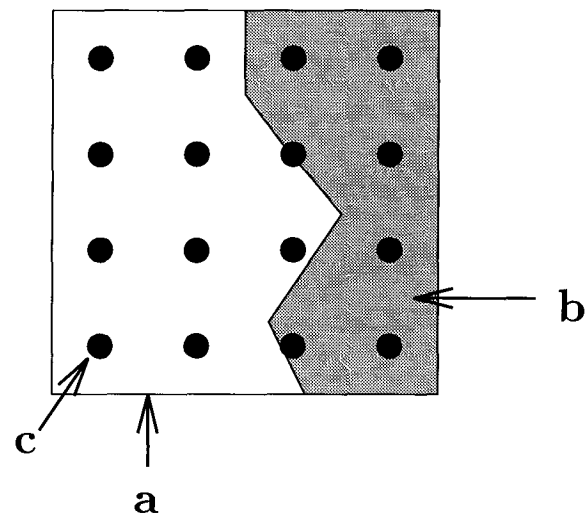
The original algorithm provides a simple means to prevent colour aliasing in composited corner sampled depth images, and is equally effective in dealing with centre sampled images with the modifications presented in this section. It should be noted that in either case, the filtering is independent of the order which images are composited, thus depth compositing  $a$  over  $b$  is equivalent to depth compositing  $b$  over  $a$ .

### 3.2 Generating antialiased images for composition

Differing methods of generating antialiased images can produce odd effects when images are composited together. In [duff85], no constraints are given for antialiasing methods that will produce consistent results when images are composited. This section looks into the cause of the problems and presents a simple way of eliminating them.

Antialiasing provides a way of masking, or even eliminating, problems in images and image sequences caused by the discrete nature of computer generated pictures, and many papers have dealt with methods for doing this ([crow77], [crow81], and many more, with a good summary of antialiasing methods given in [fole90]). When dealing with single images, the most apparent effects are jagged edges on colour or object boundaries. These *jaggies*, as they are often referred to, result from the space and colour quantizing of the computer's display hardware. For example, when rendering a polygon, if part of a pixel in the image is covered by an edge of the polygon and part is not, there is no way of dividing up the pixel so that one part displays the colour of the polygon, while the other displays the background colour. To overcome this, most antialiasing methods use an over-sampling approach. When rendering a scene, many samples of colour are taken within the area of a pixel. The colours obtained by each sample are then combined to give the final colour of the pixel (call this  $p$ ) in the image. Thus, as in the previous example, if the polygon colour (call this  $P$ ) is found in  $m$  samples, and the background colour ( $B$ ) is found in  $n$  samples, the colour of the pixel in the image would be  $(Pm + Bn)/(m + n)$ . This pixel is also given an  $\alpha$  value representing the ratio of  $m$  to the total number of samples (*figure 3.3*), to be used for compositing purposes.

Antialiased pixels solve one problem, but bring about another when during compositing ([fium83] and [carp84]). Considering the previous examples, it is not easy to see what the depth value for the pixel  $p$  should be. Since the multiple sampling method



- a) pixel
- b) polygon crossing part of pixel
- c) sample point

Figure 3.3: Sampling values within a single pixel

of antialiasing works so well on colour values, it might be tempting to use it to try to resolve the depth value problem. Taking the average depth value of all the samples is dangerous, however, since the average value may not be at all representative of the colour value and its association to other pixels. If the depth of the polygon is  $D$  and the depth of the background  $I$  (consider this the largest value possible, representing the lack of any depth information - see *Appendix A*), the antialiased depth of the image pixel would be  $(Dm + In)/(m + n)$ , as in the colour antialiasing, falling in between the polygon of interest and the background. Now consider that another image is composited such that the depth value of the second image at  $p$  would place it in behind the polygon with depth  $D$ . Because of the averaging done on the depth from the first image,  $p$  from the second image would have less depth than  $p$  from the first image. The pixel colour displayed in the resultant image will be the colour of  $p$  from the second image, using the compositing algorithm. When looking at the resultant image, jaggies at the boundary of  $p$  will be noticeable, since  $p$  carried antialiased colour information from the polygon in the first image that was lost upon compositing because the depth value at  $p$  did not accurately reflect the relationship of  $p$  to the polygon. An illustration of this is given in *figure 3.4*.

To prevent the above situation, the depth of  $p$  must be taken to be the least depth of all the samples. If the depth value used is always a specific sample from the  $m + n$  samples taken, more problems can occur. If two pixels  $p$  and  $q$  are on opposite sides of a square polygon as in *figure 3.5*, and the sample that is always chosen is taken at the lower left corner of the pixel, the depths of  $p$  and  $q$  will be different even though one would expect them to be the same. The solution to this problem is to take the minimum value from all the samples in a pixel. This ensures that the colour and depth data are related to the polygon which the antialiased pixel partially covers. Sampling problems, while not eliminated, are reduced since the method will return a consistent result for pixels that are partially covered by objects.

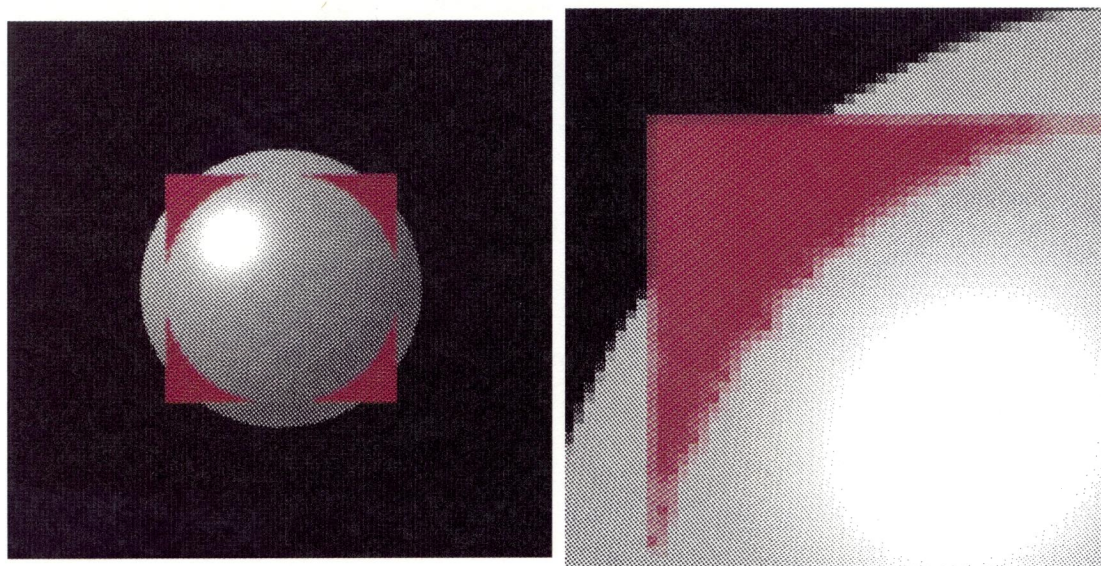


Figure 3.4: A red cube composited with a gray sphere. The first image shows the overall appearance, the second shows a magnified section where colour sampling problems appears

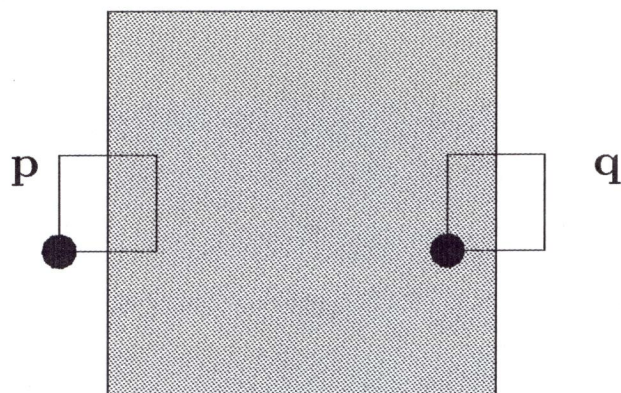


Figure 3.5: Pixels partially covering a square polygon with sampling at the lower left corner



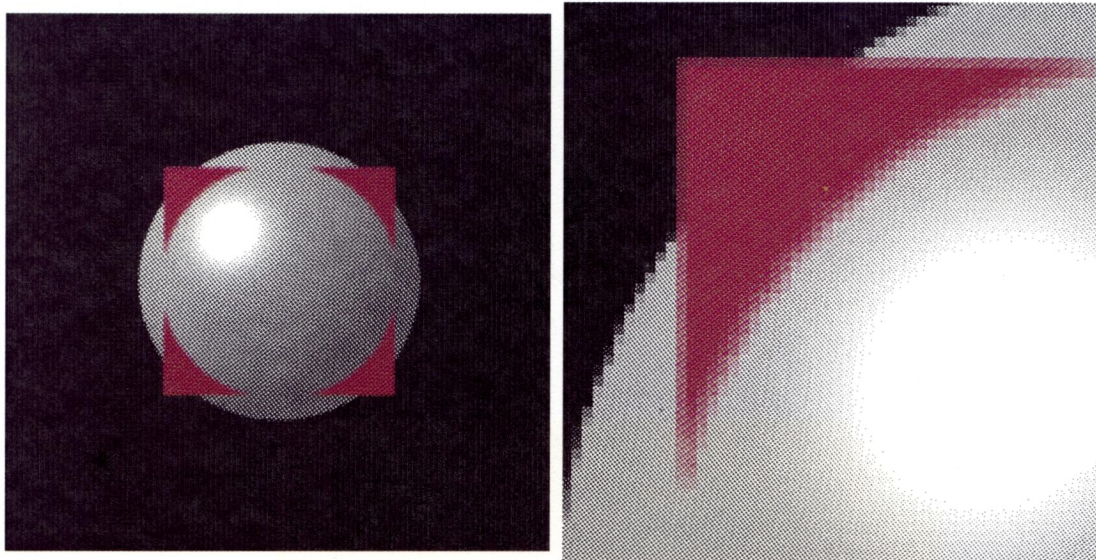


Figure 3.6: The red cube and gray sphere from *figure 3.4*, but with proper depth sampling used

### 3.3 Compositing pixels with equal depths

One problem that arises when compositing depth images is that corresponding pixels may have equivalent depths <sup>2</sup>. This occurs when two objects in the images to be composited interpenetrate in the final image. As an example, consider image *a*, a red square, and image *b*, a blue square, arranged such that when the images are composited, the resultant image has the blue square and the red square intersecting at a forty-five degree angle (shown in *figure 3.7*). If a pixel *p* has the same depth in both images *a* and *b*, then it lies on the surface of the red square at the point where it intersects the blue square. The colour of point *p* in the composite needs to be determined.

Using the Duff algorithm, the pixel of interest (assuming it is opaque) would have a resultant colour the same as the pixel in the first image, in sequence, to be composited, modified by the compositing operation. Because the compositing operation acts as a

---

<sup>2</sup>Equivalent, in this context, means that the depths of the two pixel are equal within some user defined tolerance  $\epsilon$ .

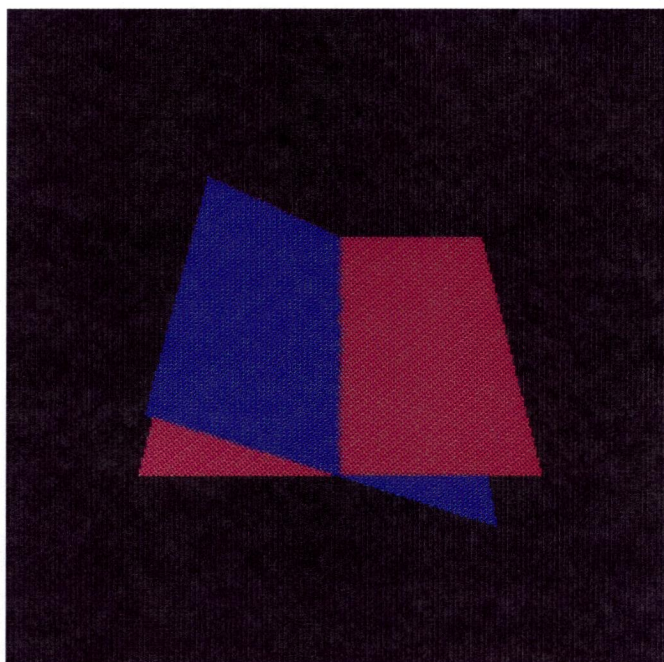


Figure 3.7: Interpenetrating squares of differing colours



filter for the depth values, it is unlikely that this choice of pixel will cause colour aliasing to appear. For example, consider a single row of pixels in two depth images where the depth values in the first image are increasing, the depth values in the second decreasing, and the depth values equal at pixel  $p$  on the row. After compositing *without* filtering the depth values, the colour of the row of pixels will undergo an abrupt change at pixel  $p$ , the point at which the depths are equal. Performing the same composite with depth filtering results in pixel  $p$  having a colour that is a blend of the colour of the pixels from the original images at  $p$ . While this may not seem significant for single images, in an animation, abrupt colour changes (depending on the motion) can distract a viewer and reduce the quality of the animation. The filtering done by the compositing algorithm will generally produce more desirable results than if filtering is not performed.

In some cases, compositing may be performed with objects where a large number of pixels have equivalent depths. Consider a composition involving two images of the same square, but with the square in one image red, the other blue. If the assumption is that blending is desirable, as with the previous example, the composite image would show a magenta square. Because there is no real world example of where differing colour objects occupy the exact same space, this may seem a reasonable treatment. However, were this part of an animation where one square was moving with respect to the other, the effect would be very disconcerting, since a large region of a color not present in the original scenes would be introduced. Thus some selection of colour must be done, rather than a blending. The Duff compositing algorithm performs this by selecting the colour of the first input image where two pixels' depths are equivalent, as noted earlier, and using this in the final calculation after filtering the depth values. In this case, however, all depth values are equivalent, so the filtering does not affect any pixel depth values, and the colour of the first input image results for all equivalent depth pixels, giving the consistent colour selection desired.

While no mention is made of the behaviour of compositing equal depth pixels in [duff85], it turns out that the compositing algorithm does, in fact, give the result that is appropriate in the above situations.

## Chapter 4

### Visible surface reconstruction

In order to reshow compositing images, world coordinate information must be extracted from the images, and some notion of the shape of the object must be inferred from this data. Because a depth image does not store enough information to completely describe an object, the only information about an object comes from those portions that are present in the picture, and thus only the visible surface of the object can be reconstructed. Of course, other assumptions about the shape of the object can be made based on either human intuition, various other aspects of the image, such as existing shadows, or changes in the visible surface of the object over a sequence of images. Since there is much work being done on all of these methods ([horn86], [ball82], many others), and the purpose of this thesis is to verify a process that can be used to reshow images (and not to solve all parts of that process), they are not discussed here in any detail. For the purposes of this thesis, a surface reconstruction algorithm giving a reasonable model of the visible surface of objects is sufficient. This visible surface model is then used by a modified ray tracer to reshow the images to be composited (see *Chapter 5*).

For arbitrarily complex images, the reconstruction of shape from visible surfaces is very difficult. Separate objects in the image must be identified and reconstructed separately, since if object  $a$  occludes object  $b$  in the scene, a surface reconstruction algorithm may consider the two objects as one, and create a model that does not reflect the separation of the objects. This would give rise to errors in the reshowing calculations

because light that would pass between objects would be blocked by the reconstructed one, as shown in *figure 4.1*. To properly separate objects, some sort of image segmenta-

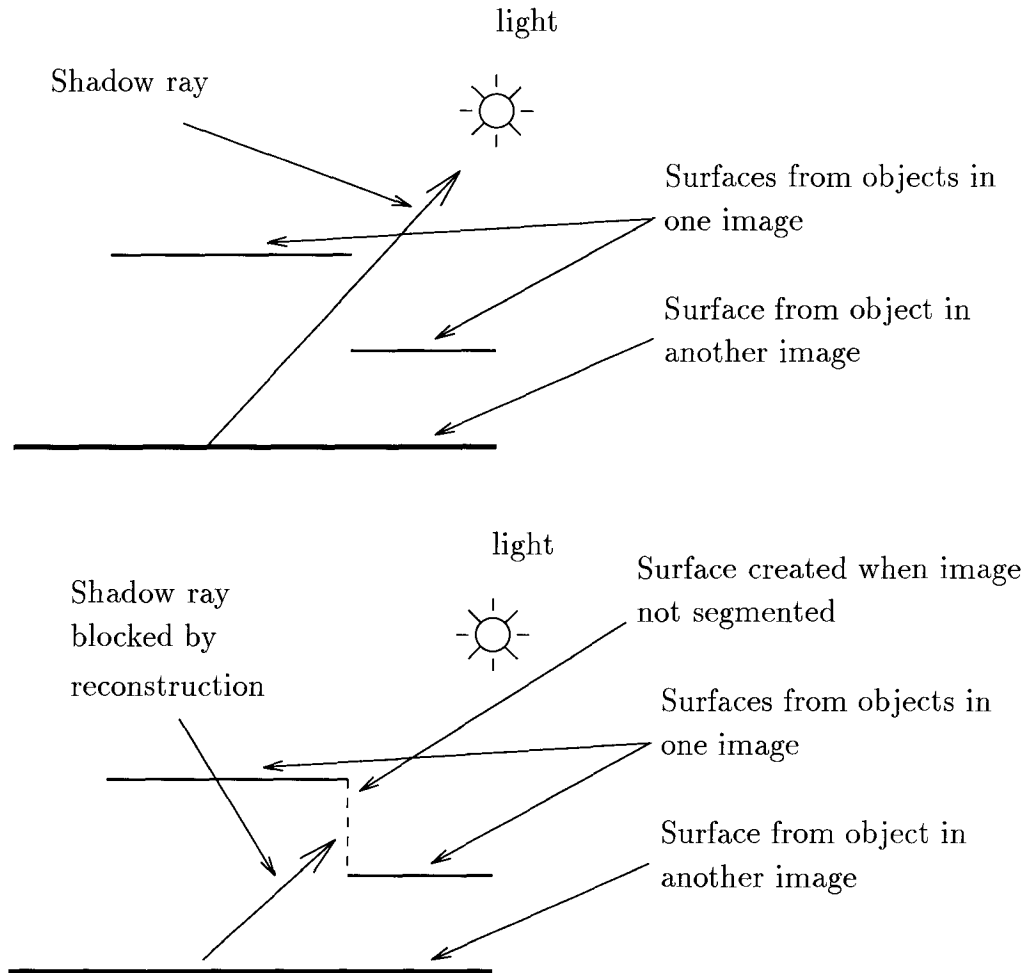


Figure 4.1: Reshadowing problems created by not segmenting images before performing surface reconstruction

tion must be done. Work on this has been going on in the area of computer vision for some time, but the accurate segmentation of arbitrary images is still a long way from being usable. Because of the complexity of the problem, this thesis deals with the shadowing problem within the restrictions described in *Chapter 1*, that is the lighting and

viewing parameters being the same in all images to be composited, and the scenes are composed of non-occluding objects. The image segmentation problem is important and still relevant however, and various aspects of it are discussed later in this chapter.

## 4.1 Simple surface reconstruction

This section details the method developed for reconstructing visible surfaces from depth information. The surface created is intended only to be used as an illustration that the method for reshadowing composited images is a valid one, and is not intended to be an ideal solution to the object reconstruction problem.

In the introduction to this thesis, it was stated that all of the viewing parameters for both images to be composited are the same, and are known. The knowledge of these parameters comes directly from the raytracing software used to produce a depth image (see *Appendix A*). Along with the depth information, the raytracer stores other information about a scene, such as the window size, distance from window to eye, eye location and bank, as well as lighting information, such as the location, type, and colour of all the lights in each scene. With this information, coupled with the depth of each pixel, a simple surface reconstruction is fairly straightforward. First, the original world coordinates of each pixel covered by an object in the scene must be obtained, and then these points joined to form a surface.

### 4.1.1 World coordinate retrieval

To retrieve world coordinates of pixels from a depth image, a modified raytracing algorithm can be used. When raytracing, a ray is cast from the eye position through a pixel into the scene. The ray is then checked for intersections with all objects in the scene, and if an intersection is found, the world coordinate of that point, as well as the distance

along the ray at which it occurred, is returned. In the raytracer used to render depth images, (called *optik*, see [aman87], [buch92], and *Appendix A*) this intersection depth is recorded as the depth of the pixel through which the ray was cast. Thus the depth map for the image represents depth from the eye point, but an option exists within *optik* that allows this distance to be projected onto a ray perpendicular to the image plane. In order to retrieve the world coordinates of the scene, this process must be performed in reverse. Calculating the various vectors needed from the eye and window information is done exactly as it is for standard ray tracing, since the viewing parameters are known ([whit79], [whit80], [fole90]). This results in two vectors of interest, namely the unit

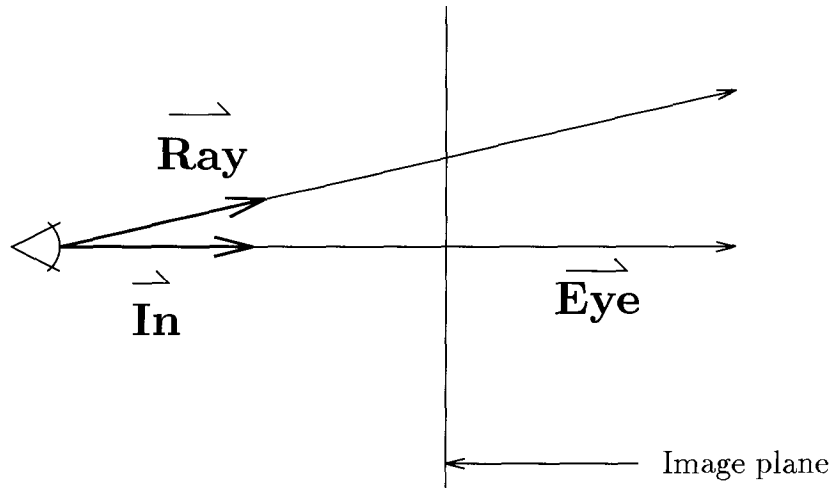
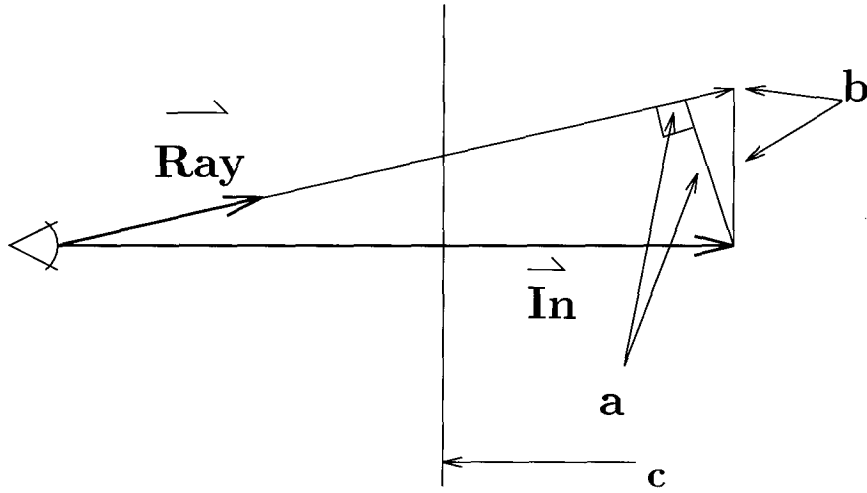


Figure 4.2: Viewing vectors for the scene

vector representing the direction of the ray from the eye through a pixel  $p$  on the window (the vector  $\vec{Ray}$ ), and the unit vector that is perpendicular to the viewing plane through the eye point (the vector  $\vec{In}$ ). It is also necessary to know the eye position relative to the world coordinate origin,  $\vec{Eye}$ , that is stored in the depth file (see *figure 4.2*). Once this is done the world coordinates of pixel  $p$  (which shall be denoted as  $P$ ) can be found for each of the two ways the depth values are stored.

If the depth value, call it  $d$ , is stored as the distance along the eye ray, the world coordinates of  $p$  can be described as  $P = (d \cdot \vec{Ray}) + \vec{Eye}$ . The calculation of  $d \cdot \vec{Ray}$  gives the world coordinates of  $p$  relative to the world origin, since  $\vec{Ray}$  is a unit vector in the correct orientation, and  $d$  is a scalar. The addition of  $\vec{Eye}$  moves the origin of the eye ray to the eye position, and is necessary because the depth is computed from the eye location.

Where the depth value is stored as the distance along the ray projected onto the vector  $\vec{In}$ , some additional calculations are required. First the scalar  $d$  multiplies the unit vector  $\vec{In}$ . The resulting vector,  $\vec{T}$ , gives the depth of  $p$  from the viewing screen. Because the vector  $\vec{Ray}$  is a unit vector,  $\vec{T}$  cannot simply be projected on to it. The correct treatment



- a) Simple projection
- b) Desired projection
- c) Image plane

Figure 4.3: Vectors for calculating world coordinates

for  $\vec{T}$  that will give the proper  $P$  is  $P = \vec{Ray} (|\vec{T}|^2 / (\vec{T} \cdot \vec{Ray})) + \vec{Eye}$ . This calculates

where along  $\vec{Ray}$  the depth would have been when projected on  $\vec{In}$  originally, and moves the resultant vector to the eye position.

#### 4.1.2 Surface construction from world coordinates

Once the world coordinates for each pixel are found, a simple surface over the points can be constructed. The method used here is to construct a simple triangular mesh where each triangle shares every vertex with three other triangles. This is done because of the way world coordinates for pixels are retrieved. Since each retrieved point occupies a space in a regular grid (the window) with respect to all the other points, a triangulation can be trivially constructed. The triangular mesh has also been chosen due to the simplicity of intersection calculations required when creating the shadow map to reshow the images.

For the case of lower left corner sampled pixels, consider that each pixel has defined world co-ordinates at the corners. If the simplest construction was done, each pixel would be a square on a surface. This does not work, however, if the corner points are not co-planar. Since this is the more likely case it would be possible to use a bilinear patch to represent the surface, but bilinear patches require complex intersection calculations (see *Chapter 5*), and would not be usable if one vertex in the pixel had no depth value, since any patch with a null vertex would not be reconstructed (as explained later in this chapter). If each pixel is split into two triangles any null vertex would again give rise to problems since neither of the triangles would be usable. This leads logically into breaking the pixel into four triangles. While some triangles would not be used, there is a very good chance that some of the pixel information would be reconstructed. To do this, an extra vertex must be introduced at the centre of the pixel where the four triangles share a vertex as in *figure 4.4*. The coordinates of the centre vertex are found by adding the coordinates of the surrounding four points and dividing each component by four.



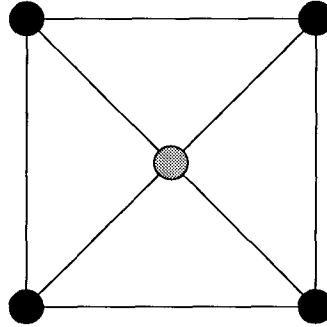


Figure 4.4: Reconstructing four triangles in one pixel from corner sampled depth data: black circles are known depth data, they grey circle is the interpolated depth data

A similar method can be used to reconstruct centre sampled pixels. Since the center vertex coordinates are already known, the four corner coordinates to construct the four triangles must be found. Each corner of the pixel has a total of four surrounding pixels adjacent to it. Since each of these pixels has a point associated with its centre, the average of all the centre points around the corner under consideration are used to calculate its value (see *figure 4.5*). Repeating this for each of the four unknown corners of the pixel allows us to make four triangles. There may be cases where the world coordinates of some vertices in a pixel cannot be retrieved because the depth value is undefined. When this occurs, any triangle with a null vertex is should not be included in the reconstructed surface, and the calculation of interpolated vertices must reflect this. Thus if there are only three non-null vertices (assuming corner sampled depth values) the vertex created by interpolation should only use the non-null data, and the coordinates of the centre vertex are found by adding all three valid vertices and dividing by three, instead of four.

Applying this procedure to a variety of images shows that certain types of objects are reconstructed poorly. Objects that slope sharply away from the camera position will display jagged edges on those parts furthest away from the camera (see *figure 4.6*). This is caused by the way the depth information is sampled at each pixel. For edge sampled

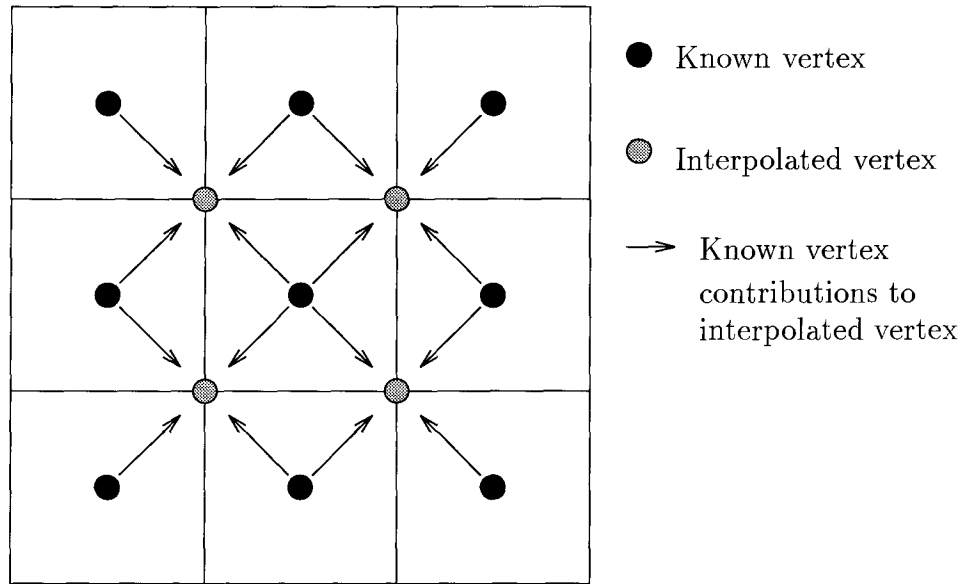


Figure 4.5: Reconstruction of triangles for centre sampled depth data: black circles are known depth values, grey circles are interpolated depth values, arrows show which known values contribute to the interpolated values

images exhibiting this problem, pixels on the back edge are surrounded mainly by pixels with no depth value. This causes the reconstruction algorithm to throw out several of the triangles of the pixel's reconstruction. If this is done consistently along the rear edge, only one triangle per pixel could be reconstructed, leaving a jagged edge. Centre sampled images show this aliasing as well, since triangles of a pixel would also be thrown out under the same conditions. One way to solve this problem is to extrapolate the sampled depth from the pixel to all corners that would otherwise have no depth value. While this stops the algorithm from discarding triangles, it gives the reconstruction a *flanged* look (triangles whose values were extrapolated at null vertices would be parallel to the image plane, and not follow the slope of the rest of the object) where it was jagged before (*figure 4.7*). Some way might be developed to estimate depth information for null vertices from the existing vertices and triangles, but this enters into the more complex



Figure 4.6: Edge problems when reconstructed objects slope sharply away from the camera. Original view is on the left, 90 degree rotation on the right.

realm of general surface reconstruction and thus is not investigated by this thesis. The jaggedness exhibited does not detract from the reshadowing method, since utilising a better surface reconstruction would eliminate the problem.

The visible surface reconstruction method put forward here gives a fairly good approximation of the surface. It is only useful for range images for which uniform dense depth data is present. If a non-uniformly sampled or sparsely valued depth image is used, the reconstructed surface will show some significant problems. This method is also susceptible to some sampling problems. If a very small object is reconstructed the shape of the object may be obscured by the discontinuities between adjacent triangles on the surface. Even for surfaces without these problems, the reconstructed triangle mesh may not be as accurate as some would like. Even though the reconstruction procedure proposed here is meant only to verify that the overall reshadowing method is feasible, it would be possible to perform some smoothing on the mesh to get a more accurate representation (a spline-based surface, for example). Other methods of retrieving surface information



Figure 4.7: Edge problems for sharply sloped objects. Left shows standard reconstruction, right shows reconstruction where sampled depth values are assigned to all parts of the pixel.

from depth data have been investigated by many people ([hopp92], [schm91], [tana92]), and perhaps some image space methods, such as shape from shading and surface from motion ([horn86] gives an overview) may also provide superior surface reconstruction to the procedure used here.

## 4.2 Image segmentation

As mentioned before, this thesis does not deal with the most general case of compositing two arbitrarily complex images. This is mainly due to the problems encountered when trying to retrieve some notion of where objects are relative to other objects in the scene. Much work has been done on the image segmentation problem in the field of computer vision, and the techniques developed can be useful for reshadowing purposes. This section puts forward what should be included in a complete solution to the image segmentation problem, and discusses some implementation problems.

### 4.2.1 The ideal solution

An ideal solution to the image segmentation problem for compositing purposes would result in each pixel in an image belonging to a known object. An object would be known if its boundaries with all adjacent, occluding, or occluded objects are known. If all of this information was available, it would be easy to apply the simple surface reconstruction algorithm (given previously) to each object in turn, by treating each object as if it were the only thing in the image. When the surface reconstruction is complete for all objects, they could be merged into the same scene and viewed in three dimensions.

### 4.2.2 Possible approaches

Boundaries of objects can be determined by some form of edge detection (such as the one given in [cann86]). Typical edge detectors operate on an greyscale (luminance only) image, and find areas of the image where there is a steep intensity gradient. This is done by using a two dimensional filter over the image and finding where the second derivative of the intensity changes from positive to negative (a zero-crossing). The various edges found by this method are then joined together, and, in the ideal case, show the boundaries of objects in the scene. If the depth information is considered as intensity information, that is each depth value is mapped to a displayable greyscale value, this method can be used to find depth discontinuities and thus detect objects (*figure 4.8* and *figure 4.9*).

Some problems occur with this approach however. When edge detection is done on a greyscale non-depth image, abrupt changes in intensity due to texture or shading are usually identified as edges. With depth images, there is no texturing or shading information present, but now other problems occur. When separate objects occupy similar areas in depth (such as a slightly sloped ramp on a desk), they are often not identified as separate objects. Complex objects can also induce problems when detecting intensity



Figure 4.8: Original image

edges. If the object has areas where there is a rapid change in slope direction (ie: a triangular prism viewed from above the long axis where two rectangles meet), there is an edge that would be detected, but it is not a boundary edge. These different types of edges must somehow be differentiated, which is difficult. The problems make it difficult to properly identify boundary edges.

Once all edges are found, each boundary edge must be joined with those from the same object to form the complete boundary of that object. In an ideal case, each boundary edge would touch only two other boundary edges that belong to the same object. Since the ideal case rarely happens, and many edges that are not boundary edges could touch the boundary edge at various places, a way of identifying the boundary edges must be found, along with a method for correctly determining which adjoining edge is a boundary edge on the same object. If this can be done then the only remaining barrier to the

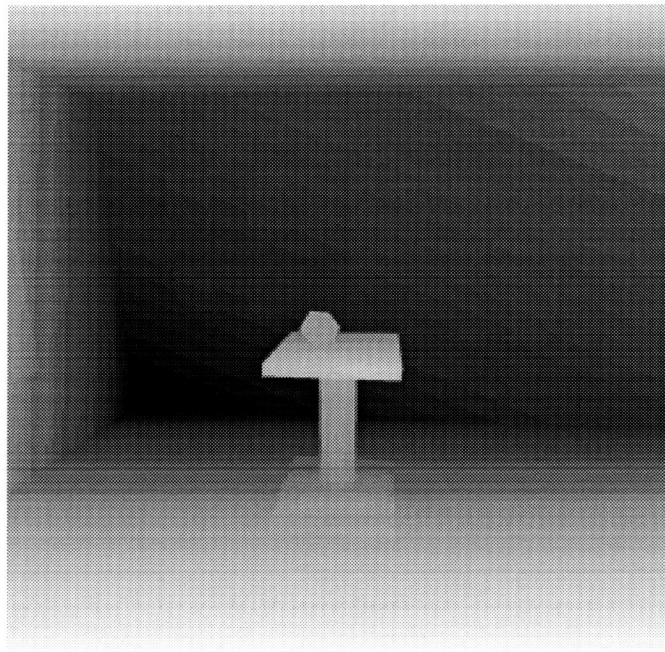


Figure 4.9: Associated depth map shown as greyscale

segmentation of the image would be to identify occluding objects. For example, if the image was of a sphere in front of a rectangle, it would have to be determined if the rectangle was actually continuous behind the sphere or not so that any boundary edges detected could be joined together properly.

People working in the area of computer vision have tackled these problems for many years, and will continue to do so, hopefully coming up with better segmentation methods that can be applied to the compositing process. When this is done, the reshadowing of composited images will be more usable.



## Chapter 5

### The reshadowing process

Creating shadows on composited images is a fairly straightforward process. Using the visible surface reconstructed from one image,  $a$ , (via the method presented in the previous chapter), the locations of shadows cast by this surface onto the objects of the other image,  $b$ , are found using a simple raytracing method. This generates a shadow map, which is an image of the same dimensions as the ones to be composited, containing a black pixel wherever a pixel in image  $b$  would be shadowed by the object in image  $a$ . The shadow map generated will be used to place shadows on image  $b$ , and must be filtered somehow to remove all hard edges and jaggies which are caused by the binary nature of the shadow map, as first produced. Once image  $b$  has been reshadowed the two pictures are composited to produce the final image. In the case of images that are appropriate to use for mutual shadowing as mentioned in *Chapter 1*, the reshadowing process is repeated twice, with the images swapping roles the second time through. This results in both images being reshadowed, and the compositing is then performed.

#### 5.1 Creating the shadow map

The purpose of creating a shadow map is to show where shadows should be cast on image  $b$  from the reconstructed object in image  $a$ . This situation is analogous to the shadow determination problem when raytracing a scene. When a scene is raytraced, a ray is cast from the eye into the scene, where it may intersect some object. If this is the case, the

point of intersection is found, and another ray is cast from that point to the light. If this ray intersects any object, then the point from which it was cast is considered to be in shadow. For depth images, it is a simple matter to retrieve the world coordinates of each pixel in the image (see Chapter 4). The world coordinate is the intersection point of the eye ray from the original rendering with an object in the scene. As with raytracing shadows all that is needed is to cast a ray from that point to the light, and find any intersections along it. If an intersection is found, a black pixel is written to the shadow map in the same location as the pixel from image *b* being considered.

The existing raytracer, *optik* (see *Appendix A*), was modified to create the shadow map because many procedures that already exist are needed or readily adaptable for this purpose. A routine was added to read in a depth image and to convert all depths to world coordinate information using the same method as for surface reconstruction. The light position was also retrieved from the depth image header and added to the scene. Using the existing interface for *optik*, the triangles making up the reconstructed surface can be easily read into the scene. A shadow ray between a world coordinate point from the depth image and the light is created using existing routines in *optik*, and the original image coordinates of the point are noted. The triangles from the reconstruction are checked for intersections with the cast ray using the ray intersection routines in *optik*, and if there is an intersection, a black pixel is written to the final image at the position that was noted before. If there is no intersection, a white pixel is written. This process is repeated for every pixel that has a depth value from the original image *b*. Because there are likely to be many triangles in the visible surface reconstruction, using *optik* allows the existing ray-intersection acceleration techniques to be used ([woo89], [aman87]), reducing the time taken to produce the shadow map.



Figure 5.1: A shadow map created by the modified raytracer

## 5.2 Filtering the shadow map

The shadow map produced by the above method contains only black or white pixels. If this were used to reshadow an image, the result would be a hard edged shadow that would have jagged edges, depending on its orientation. Because the shadow produced by many rendering methods is soft-edged (to mimic real world shadows), the shadow created here should be soft edged as well. To do this, a simple filtering technique is used, which also smooths jaggies or other artifacts that can result from shadows that do not follow a row or column of pixels, as in *figure 5.1*.

To smooth the shadow map, it is easiest to darken the white pixels that border black ones. The simplest way to do this is to darken the pixel being considered by a percentage equal to the ratio of black to white pixels surrounding it. If there were four white, and four black pixels in the shadow map surrounding the one of interest, it would be darkened to fifty percent of its original value. If there are no black pixels in the area, then nothing



Figure 5.2: A shadow map after filtering has been performed

is done. Doing this for all pixels in the shadow map completes the filtering (see *figure 5.2*). Slightly better results can be obtained using different weighting on the simple box filter. If every white pixel is given a percentage of black equal to twenty percent times the number of surrounding black neighbours (up to five black surrounding pixels, where the white pixel would be changed to totally black), the intensity falloff around black edges becomes closer to a smooth gradient than the initial simple filter. This appears less jagged to the human eye, and can thus be judged better than the initial method (*figure 5.3*). By varying the filter width (from a single pixel to many) differing smoothing effects can be created that is tailored to a particular person's taste, since not everyone will agree on what amount of filtering is best. In an animation created using the first method, where a rotating square from one image was placed above a stationary square from another (*figure 5.4*), the filtering was sufficient to mask any jaggies from the original shadow map.



Figure 5.3: The result of applying a weighted filter to the shadow map

### 5.3 Using the shadow map

To reshadow an image, the antialiased shadow map is used as an overlay. Because the shadow map is the same size as the image to be reshadowed, each pixel in the image has a corresponding pixel in the shadow map. For every pixel  $p$  in the image, the same pixel in the shadow map is evaluated. If the pixel is white, nothing is done to  $p$  in the image, but if it is not white, reshadowing occurs. This is done by comparing the colour of the pixel in the shadow map to black. If the pixel in the shadow map is black, the corresponding pixel in the image is reduced in intensity by half, in order to approximate the default shadowing that *optik* produces. If  $p$  in the image originally had red, green, and blue values of 150, 90, and 60, respectively, the reduction would change the values to 75, 45, and 30. If the pixel is between pure black and white, then the amount of black in the pixel, expressed as the black value divided by the maximum black value, is halved, and used as the percent by which to darken the original pixel. This method of

reshadowing produces a soft shadow on the image that is a very close approximation of a shadow made by *optik* under default conditions (as in *figure 5.4*).

When raytracing the original scene, the selection of parameters for surfaces (specularity, ambient illumination, etc), and light intensity can cause the reshadowing process to produce shadows that are less acceptable. The shadows may reduce the intensity of pixels too greatly, or not enough. To overcome this, a measure of control has been added so that the default intensity reduction can be altered by employing the equation  $((S * (1 - D)) + D) * c$ , where  $S$  is the shadow map value at a pixel  $p$  between 0 (black) and 1 (white),  $D$  is the intensity reduction to be performed when  $S$  is 0, and  $c$  is the colour of  $p$  in the image to be reshadowed. This allows the user to adjust the effect of the shadow to some degree. As an extension to this work, a more interactive interface should be created, allowing the user to change the reshadowing (not the shadow position, merely the intensity) and view it to judge whether it is satisfactory. As it is now, the interface is manageable, and reshadows images well for default *optik* images. The completion of the reshadowing process occurs when the reshadowed image and the image from which the surface was reconstructed are composited together using the method described in *Chapter 3*. In the case of two images shadowing each other, the reshadowed version of both images are composited.

#### 5.4 Reshadowing already shadowed images

Throughout this chapter, the reshadowing process has been discussed as if no shadow existed in either original image. If original shadows do exist, problems will occur when using the method presented here. If the shadow map created had a shadow on pixel  $p$  in an image, and  $p$  was a shadow pixel from the original image, it would be additionally darkened by the overlay process. This would result in an image where newly created

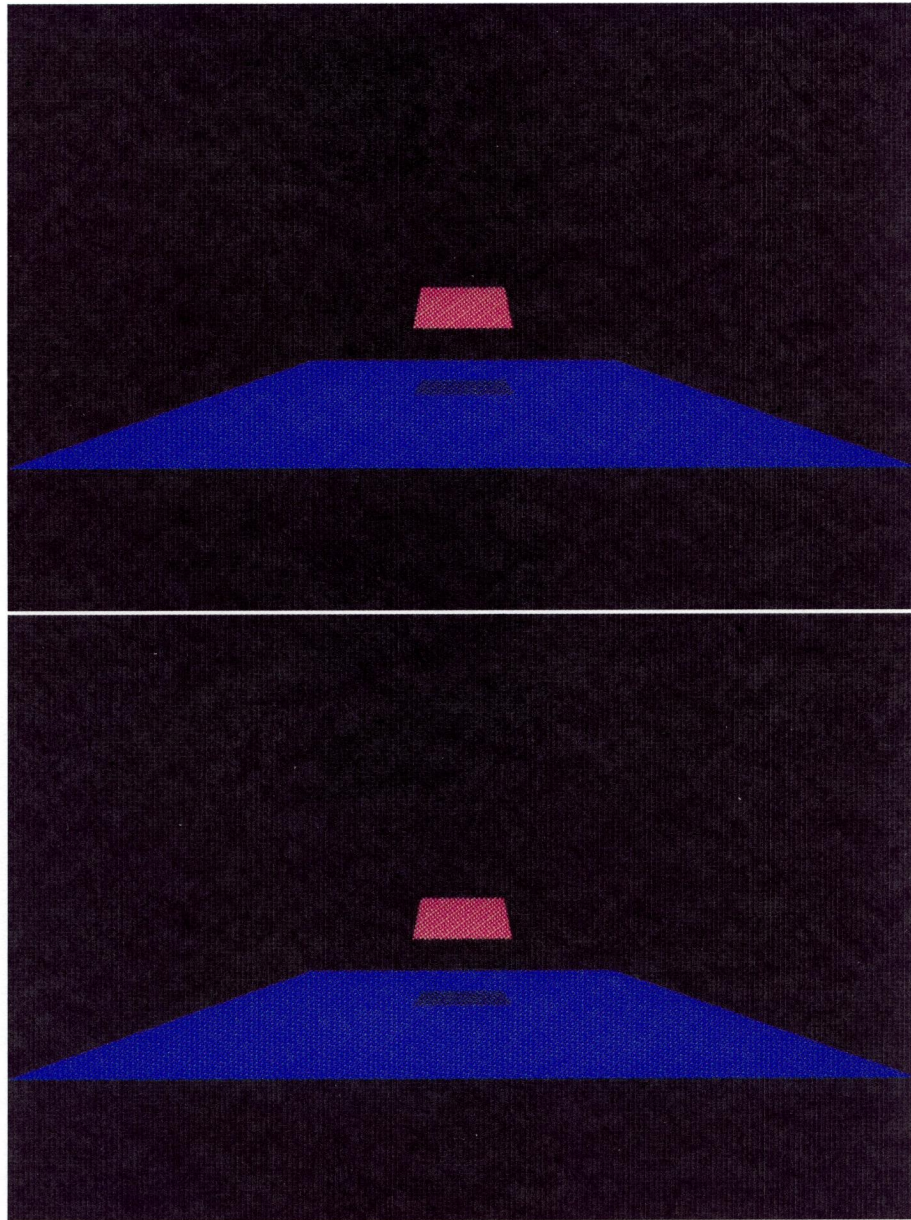


Figure 5.4: Rendered and composite images of the same scene. The rendered image is on the bottom

shadows and original shadows could be differentiated by the original shadows being darker than the original ones, which is undesirable. To prevent this from happening, a couple of steps could be taken. First, each pixel in an image could have an extra bit of information stored with it, a 1 if the pixel is in shadow, or a 0 if it is not, or optionally use one byte to denote the amount of shadow present. This extra information would be determined by the renderer when the image is first created. When compositing, this information would be used to determine if a pixel which would be reshadowed should be darkened. If the pixel should be reshadowed, the compositor would then switch the shadow bit on so that further compositing operations would know the pixel is in shadow. The second method for preventing reshadowing of already shadowed pixels is to reconstruct the scene every time it is composited, and to create a shadow map for the scene *without* considering any other images or reconstructions. After doing this, another shadow map would be constructed as described in this chapter. The first shadow map would then be compared to the second, and wherever there was a black pixel in both, that pixel would be turned white in the second. The modified shadow map would then be overlaid on the image to be reshadowed, with the result being that since pixels already in shadow are noted as white on the shadow map, they would not be reshadowed, and thus the output image will look better.

The first method is a very good one, with the exception that added storage is required to keep the shadowing information. While the ratio of shadow information to colour information would be quite small (one byte, or 256 levels of grey, of shadow information versus four bytes of *rgba* information for the image), the extra data would not be used for most normal image operations, wasting some space. Since a person would not necessarily know the use to which an image might be put in the future, most would include this shadow information on the off chance that it may be needed at some time. The second procedure outlined above has some obstacles to successful implementation. First of all,



the problem of image segmentation needs to be solved before it can be determined where shadows are in a single image. Secondly, the problem of complexity arises. When determining shadows some surface reconstruction must be performed, which takes time as well as space, whereas the first solution presented above merely takes more space. Also, shadow map creation via this method will not be as accurate as one created from the renderer itself, due to the reconstruction process. The advantage this method has over the renderer created one is that a shadow map can be made for any depth image, without having to store information while rendering, but it is currently impractical. Many of the shadowing problems noted here are also discussed in [gers87], as well as those related to the more general illumination issue.

With the double shadowing problem either ignored or worked around by the ways given here, images produced with this reshadowing method are nearly indistinguishable from those that were rendered from a complete scene under default conditions without a detailed side-by-side comparison of reshadowed and rendered images.

## Chapter 6

### Other applications and extensions

A logical extension of the resharding method presented in this thesis is to combine real video images with computer generated ones in the same way two computer pictures are combined. This would eliminate the need for teams of artists painstakingly creating shadows on images that combine computer objects and a real scene (the films Terminator-2 and Jurassic Park, for example) as is done currently.

#### 6.1 CAR - Computer Augmented Reality

As computers become faster and algorithms are developed and improved, it has become easier to use computer graphics to create special effects for film and video. Technology is approaching the point where computer graphics will be able to be seamlessly integrated into real images with very little effort. The methods that are being developed to do this can be grouped into an area that has been termed computer augmented reality (or CAR). Various aspects of CAR are being investigated at the University of British Columbia, of which this thesis will hopefully form a part. Problems dealing with global illumination (highlights, adding and removing lights etc), viewing parameter matching, and local illumination (shadows, reflections, etc) are all being investigated by the UBC CAR project, and a brief overview of all the problems involved will be presented here, along with the specific application of this thesis to CAR.

The main problem with CAR is to translate various aspects of the real image into

values that can be understood by computer graphics programs. For example, much needs to be known about the camera in the real image, such as the position it is in relative to the objects in the scene, whether it is tilted, zoomed, defocused and so on. Most computer graphics programs define the computer generated camera by several values, such as the point in space at which it is located, the point in the scene at which it is aimed, the angle of view, any rotations or tilts relative to the global axes, and the screen distance from the actual point where the camera is located. When taken together, all of this information forms the *viewing pyramid* of the scene. This viewing pyramid encompasses those parts of the scene that are visible to the camera, and is used to create the two dimensional image of the scene. In order to be able to effectively use the real image, the viewing pyramid for the camera that took the image must be determined.

The success of this determination depends largely on what can be inferred from the scene. It is not possible to retrieve the viewing pyramid from a scene unless something is known about the object in the scene. If the scene is a staged one, every object in the scene can be measured for size and relationship to other objects, then a computer model made of the real objects. This can then be manipulated until the objects in the computer scene match those in the real one. The viewing parameters of the computer model can then be taken as the viewing parameters of the real camera. If the real image is actually a part of a sequence, this process can be repeated for each frame of the film or video. This method works well, but needs the measurements of the real objects. It also takes large amounts of time, mainly by people, to match the real scene with the computer model. Ways of making the computer aid in the scene matching are currently being investigated, but the process is still mostly manual ([four93]). Means of determining the viewing pyramid without measuring the scene are more desirable than the previous method, since if you do not have to have the physical objects present, camera parameters can be determined for any images, not just staged ones. Several methods exist is the

computer vision field for tracking objects within a scene ([horn86], [lowe90]), and these can be used in conjunction with human methods. If a person outlines objects in a scene, the computer can then go and make a guess as to the objects' dimensions and relations to each other. Once this is done, the viewing parameters can be calculated, and tested with a defined object that is inserted in to the scene. If the user thinks the inserted object fits with the scene, the viewing parameters guessed at can be used as the parameters of the real camera. If the object does not fit, another guess, based on how the inserted object appears, can be made and the process repeated. Once the parameters of one image are known, the objects in subsequent images can be related to the ones in the known image by the tracking method mentioned previously. While this method is more widely useful than the manual measurement procedure, more problems can occur because of the difficulty of tracking objects between images. Also, while the objects are outlined by the user, and the relative sizes are easy to compute, the dimensions of the objects must be guessed at, which could take a large amount of time. If, when object detection algorithms become capable of doing so, the computer is used to determine object boundaries, the process would be sped up considerably.

Once the viewing parameters for the real image are known, the parameters for the computer generated image must be manipulated to match those of the real image, or vice versa. Changing the viewing parameters of one image to those of another is not a well defined problem. Since the parameters of the two images could be completely different, the transformation required might take more information than the image provides, and not be possible. Simple operations, such as coordinate transformations when all but the viewing point are the same, are easy to implement, but aspect ratio changes and depth scaling are much more difficult. One answer to all of these image manipulation problems is to construct the computer scene using the viewing parameters of the real image. This eliminates all of the effort required to massage one set of viewing parameters to another

at the expense of being able to use an arbitrary computer generated image or sequence.

Having somehow retrieved the viewing parameters between the real and computer generated images, and having a fairly accurate model of the real scene, some combination of the two images can be performed. If lights exist in the computer scene that are not visible in the real scene, the effects of these lights can be calculated, and the real image reshaded to show the new lighting effects. Likewise, if a scene model exists, and the location of the real lights are known, these can be used to shade the computer generated objects in relation to the scene. The location of shadows caused by computer generated lights shining on the real scene can be found by using the model of the real scene. Finally, all of these effects can be applied to the real image. Highlights can be added or removed, and shadows created or eliminated in their proper locations by use of image processing techniques. Correctly shaded computer generated objects can then be rendered and composited into the real scene by standard two dimensional compositing. Because the computer generated image is created specifically for the particular real image from a model of the real scene, the computer objects can be rendered so that they appear to interpenetrate and be occluded by real objects. This creates effects that make the final image or sequence look almost as if the computer objects and effects were present in the original scene ([four93]).

As an alternative to the methods for real scene modeling described above, stereo image pairs can be used to obtain a depth map ([horn86], [bake81]) of the images, or methods that operate on sequences of images ([shao88], [matt88]) might be employed to determine a depth map. This depth map can then be utilised by the reshadowing procedure described in this thesis as if the real image were, in fact, computer generated. Because the depth map of the real image will be sparser than that of a computer generated one due to the method of retrieval, some form of reconstruction should be performed on the depth data so that the reshadowing process can make use of it. Also, real images

will most likely be fairly complex, which presents problems for the current resharding method, exactly as in the case for complex computer generated scenes. As mentioned in *Chapter 1*, the solution to this problem is to come up with a good image segmentation algorithm so that objects in the image can be logically separated from one another for the reconstruction process. While the reconstruction method presented in *Chapter 4* works well for visible surfaces, it does not give a good notion of the overall shape of the object, which may be needed to retrieve the viewing parameters of the real scene. If only resharding is desired the viewing parameters need not be retrieved, but the depths of the computer generated image and real image must be matched in some fashion so as to prevent errors of scale when compositing. This might be achieved by using an interactive technique where the user dynamically scales the depths of one image against the other so as to get a satisfactory correlation between them. Problems with perspective and aspect ratio could occur as well, in which case the viewing parameters of the real image would have to be retrieved. In this case, and if the real image was to be used in other CAR related operations, some way of finding the shape of an object, and not just its visible surface, should be found, as discussed in *Chapter 4*. This would facilitate the recovery of the viewing parameters, as well as be of assistance when performing global illumination tasks. Once the two images have been altered enough so that resharding can proceed, the procedures described in this thesis can be employed to reshard the real image. If the computer generated image is to be resharded as well, some way of finding the light position in the real image must be found.

Despite all of the hurdles yet to be overcome, the method of resharding computer generated images presented in this thesis shows promise for applications using computer graphics to enhance real images. Research currently being conducted on CAR problems may also provide results that will be useful in broadening the scope of the resharding

method of this thesis to include general reshading of images, better surface reconstruction, and even image manipulation where necessary. It is also hoped that the procedures described here will be of use to those delving into the CAR field once the various limitations have been overcome.

## 6.2 What now?

As has been mentioned throughout this thesis, there are many things that should be done to make the reshadowing process truly useful. Most important of these is finding a good image segmentation method, so that arbitrarily complex depth images can be composited and reshadowed. Edge detection techniques may prove useful in doing this, as might modified surface fitting and discontinuity finding algorithms, such as [terz83], [marr84], [gamb87], and [terz88], which have been successfully applied to restricted domain problems. After this is done, a better reconstruction method should be found. Initially a better visible surface finding algorithm should be created, where a smoother surface is constructed from the points that will be at most as expensive to intersect in the shadow map calculations as the current triangular mesh. Some heuristic method should also be used to guess at the best way to reconstruct the complete object, using human input to verify the guesses, unless it becomes possible to easily retrieve the entire object description. Current shape from shading methods, as well as reconstruction from image sequences, coupled with the depth and colour information, will hopefully provide a basis for doing this.

Once it is possible to reshow arbitrarily complex images, a method of dealing correctly with multiple, extended, and area light sources should be found. This entails not only the creation of shadows cast from such light sources, but also the removal or lessening of those shadows that would be illuminated by lights if the scenes were composited. This

leads into the more general reshading problem of CAR, where methods from there could be borrowed to create new highlights on objects illuminated by a light from a composited image, as well as removing highlights from reshadowed areas of images. At the same time, ways should be found of overcoming problems encountered when compositing images with separate viewing parameters, again CAR research will hopefully be able to assist in this.

Reshadowing real images should be investigated in parallel to all of these improvements. Depth maps for real images should be obtained somehow, either from image methods, such as stereo image matching, or from other methods, such as laser range finding or sonic echo location. Once the real images have the information that is needed to composite two computer generated pictures, all of the things that can be done to computer images can be performed on the real image. Thus, any combination of real and computer generated images will be able to be composited using the same sets of tools. Ultimately this method should be used in some fashion by CAR research groups, providing a means of assisting both local and global shadowing calculations.



## Chapter 7

### Conclusions

This thesis has extended the three dimensional compositing method of Duff to enable objects within different scenes to interact with each other in such a way to create shadows on images as if all the objects in the scenes were present at once. While the procedure described here does not solve all problems associated with this process, it does show that reshowing of composited images can be done easily for images with restricted characteristics, and provides thoughts and possible methods of dealing with images of arbitrary complexity. A fast and reasonably accurate visible surface reconstruction algorithm was given that allows a modified raytracer to create a shadow map for use in reshowing composited images. Applications of this process to the area of computer augmented reality have been discussed. A short video was produced in the course of this thesis (of which *figure 5.4* is a part) that shows the results, both positive and negative, of the methods described.

The results of performing these processes to reshow images shows great promise. As it stands now, to produce a reshowed image with the desired qualities too many restrictions must be imposed on the images to make the methods presented in this thesis usable in anything other than a research environment. If the improvements suggested throughout this thesis are performed, the reshowing process will be made more robust and useful. This thesis has shown that the reshowing process can prove valuable in many aspects of image production and modification, and it is hoped that it will be an

important initial effort upon which others can build.

## Bibliography

- [aman87] J. Amanatides and A. Woo. “Optik Users’ Manual”. Technical Report DGP 1987-2, University of Toronto, August 1987.
- [bake81] H. H. Baker and T. O. Binford. “Depth from edge and intensity based stereo”. *Proc. 7th Int. Joint Conf. on Artificial Intelligence*, pp. 631–636, 1981.
- [ball82] D. H. Ballard and C. M. Brown. *Computer Vision*. Prentice-Hall, 1982.
- [boot86] Kellogg S. Booth, David R. Forsey, and Alan W. Paeth. “Hardware assistance for Z-buffer visible surface algorithms”. *Proceedings of Graphics Interface ’86*, pp. 194–201, May 1986.
- [buch92] John Buchanan. “[kmpsx]\*optik”. *Proceedings of the 1992 Western Computer Graphics Symposium*, pp. 147–152, April 1992.
- [cann86] J. F. Canny. “A computational approach to edge detection”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 8, No. 6, pp. 679–698, 1986.
- [carp84] Loren Carpenter. “The A-buffer, an Antialiased Hidden Surface Method”. *Computer Graphics (SIGGRAPH ’84 Proceedings)*, Vol. 18, No. 3, pp. 103–108, July 1984.
- [crow77] Franklin C. Crow. “The Aliasing Problem in Computer-Generated Shaded Images”. *Communications of the ACM*, Vol. 20, No. 11, pp. 799–805, November 1977.
- [crow81] F.C. Crow. “A Comparison of Antialiasing Techniques”. *IEEE Computer Graphics and Applications*, Vol. 1, No. 1, pp. 40–48, January 1981.
- [duff85] T. Duff. “Compositing 3-D Rendered Images”. *Computer Graphics (SIGGRAPH ’85 Proceedings)*, Vol. 19, No. 3, pp. 41–44, July 1985.
- [fium83] E. Fiume, A. Fournier, and L. Rudolph. “A Parallel Scan Conversion Algorithm with Anti-Aliasing for a General Purpose Ultracomputer”. *Computer Graphics (SIGGRAPH ’83 Proceedings)*, Vol. 17, No. 3, pp. 141–150, July 1983.

- [fole87] James D. Foley and Won Chul Kim. "Image composition via lookup table manipulation". *IEEE Computer Graphics and Applications*, Vol. 7, No. 11, pp. 26–35, November 1987.
- [fole90] J.D. Foley, A. van Dam, Steven K. Feiner, and John F. Hughes. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley Publishing Company, second edition, 1990.
- [fors88] David R. Forsey and Richard H. Bartels. "Hierarchical B-Spline Refinement". *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, No. 4, pp. 205–212, August 1988.
- [four93] A. Fournier, A. Gunawan, and C. Romanzin. "Common Illumination Between Real and Computer Generated Scenes". *Graphics Interface (GI) '93 Proceedings*, May 1993.
- [gamb87] C. Gamble and T. Poggio. "Visual integration and detection of discontinuities: the key role of intensity edges". AI-Memo-970, MIT AI Laboratory, Cambridge, MA, 1987.
- [gers87] Ron Gershon. *The use of color in computational vision*. Ph.D. thesis, Dept. of Computer Science, University of Toronto, 1987.
- [haeb90] Paul Haeberli and Kurt Akeley. "The Accumulation Buffer: Hardware Support for High-Quality Rendering". *Computer Graphics (SIGGRAPH '90 Proceedings)*, Vol. 24, No. 4, pp. 309–318, August 1990.
- [hopp92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. "Surface reconstruction from unorganized points". *Computer Graphics (SIGGRAPH '92 Proceedings)*, Vol. 26, No. 2, pp. 71–78, July 1992.
- [horn86] B. K. P. Horn. *Robot Vision*. McGraw-Hill, 1986.
- [jaha91] Ghassan Jahami and Michel Beigbeder. "A rendering compositing method". *COMPUGRAPHICS '91*, Vol. I, pp. 504–513, 1991.
- [levo77] M. Levoy. "A color animation system based on the multiplane technique". *Computer Graphics (SIGGRAPH '77 Proceedings)*, Vol. 11, No. 2, pp. 65–71, July 1977.
- [lowe90] D. G. Lowe. "Integrated treatment of matching and measurement errors for robust model-based motion tracking". *Proc. 3rd International Conference on Computer Vision*, pp. 436–440, 1990.

- [marr84] J. L. Marroquin. "Surface reconstruction preserving discontinuities". AI-Memo-792, MIT AI Laboratory, Cambridge, MA, 1984.
- [matt88] L. Matthies, R. Szeliski, and T. Kanade. "Incremental estimation of dense depth maps from image sequences". *Proc. IEEE Conf. Computer Vision and Pattern Recognition, 1988*, 1988.
- [moln92] Steven Molnar, John Eyles, and John Poulton. "PixelFlow: High-speed rendering using image composition". *Computer Graphics (SIGGRAPH '92 Proceedings)*, Vol. 26, No. 2, pp. 231–240, July 1992.
- [nada87] Tom Nadas and Alain Fournier. "GRAPE: An Environment to Build Display Processes". *Computer Graphics (SIGGRAPH '87 Proceedings)*, Vol. 21, No. 4, pp. 75–84, July 1987.
- [naka86] E. Nakamae, K. Harada, T. Ishizaki, and T. Nishita. "A Montage Method: The Overlaying of the Computer Generated Images onto a Background Photograph". *Computer Graphics (SIGGRAPH '86 Proceedings)*, Vol. 20, No. 4, pp. 207–214, August 1986.
- [naka89] Eihachiro Nakamae, Takao Ishizaki, Tomoyuki Nishita, and Shinichi Takita. "Compositing 3D Images with Antialiasing and Various Effects". *IEEE Computer Graphics and Applications*, Vol. 9, No. 2, pp. 21–29, March 1989.
- [port84] T. Porter and T. Duff. "Compositing digital images". *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, No. 3, pp. 253–259, July 1984.
- [potm82] M. Potmesil and I. Chakravarty. "Synthetic Image Generation with a Lens and Aperture Camera Model". *ACM Transactions on Graphics*, Vol. 1, No. 2, pp. 85–108, April 1982.
- [potm87] Michael Potmesil and Eric M. Hoffert. "FRAMES: Software Tools for Modeling, Rendering and Animation of 3D Scenes". *Computer Graphics (SIGGRAPH '87 Proceedings)*, Vol. 21, No. 4, pp. 85–93, July 1987.
- [sale90] David Salesin and Jorge Stolfi. "Rendering CSG Models with a ZZ-Buffer". *Computer Graphics (SIGGRAPH '90 Proceedings)*, Vol. 24, No. 4, pp. 67–76, August 1990.
- [schm91] F. Schmitt, Xin Chen, and Wen-Hui Du. "Geometric Modelling from Range Image Data". *Eurographics '91*, pp. 317–328, September 1991.
- [shao88] M. Shao, T. Simchony, and R. Chellappa. "New algorithms for reconstruction of a 3D depth map from one or more images". *Proc. IEEE Conf. Computer Vision and Pattern Recognition, 1988*, 1988.

- [shaw89] Christopher D. Shaw, Mark Green, and Jonathan Schaeffer. "Anti-aliasing issues in image composition". *Proceedings of Graphics Interface '89*, pp. 113–120, June 1989.
- [shaw90] Christopher D. Shaw, Mark Green, and Jonathan Schaeffer. "A parallel graphics system using raster image composition". *Proceedings of the 1990 Western Computer Graphics Symposium*, March 1990.
- [tana92] H.T. Tanaka and F. Kishino. "Recovering and Visualizing Complex Shapes from Range Data Adaptive". *Visual Computing (Proceedings of CG International '92)*, 1992.
- [terz83] D. Terzopoulos. "The role of constraints and discontinuities in visible-surface reconstruction". *Proc. 8th Int. Joint Conf. on Artificial Intelligence*, pp. 1073–1077, 1983.
- [terz88] D. Terzopoulos. "The computation of visible surface representations". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 4, pp. 417–438, July 1988.
- [whit79] T. Whitted. "An improved illumination model for shaded display". *Computer Graphics (Special SIGGRAPH '79 Issue)*, Vol. 13, No. 3, pp. 1–14, August 1979.
- [whit80] Turner Whitted. "An Improved Illumination Model for Shaded Display". *Communications of the ACM*, Vol. 23, No. 6, pp. 343–349, June 1980.
- [woo89] Andrew C. H. Woo. "Accelerators for Shadow Determination in Ray Tracing". M.Sc. Thesis, Department of Computer Science, University of Toronto, 1989.

## Appendix A

### Depth Files

Throughout this thesis there have been many references to the *optik* depth file format. While the procedures given can be applied to virtually any depth file format, it has been useful to give some concrete examples using the *optik* format. Here the *optik* format shall be explained in more detail, and other uses for the depth file format will be outlined.

#### A.1 Optik depth files

*Optik* is a program that renders mathematically modeled scenes using ray-tracing techniques based on [whit80]. It, and the variations developed by students in the course of their work, is described in [aman87] and [buch92]. One of the variations of *optik* was created by Chris Romanzin (one of the authors of [four93]) to write out a file with depth information in addition to the regular *rgba* information. This depth file uses the existing *rgba* storage routines in order to store depth information for every pixel in the corresponding image. Included in the header of the image, which describes the size of the image and the number of bits of information in each pixel, is information that completely describes the viewing pyramid and information about lighting, such as location, intensity colour, and so forth. This extra information is used to reshow images in the manner described by this thesis.

In *optik*, it is possible to store between one and three bytes of colour information, with an optional byte of  $\alpha$  information. To take advantage of this variability to store

the minimum amount of depth information that the user needs, the depth data can also be stored as one to four bytes per pixel of information. In order to do this, the depth information, as calculated by *optik* from object-ray intersection points, must be put in a form where meaningful information can be stored at many resolutions. The depth data is thus mapped into  $2^{8n}$  discrete values, where  $n$  is the number of bytes per pixel to use to store the depth information, defined by the user. Because the depth for a pixel is originally calculated as a floating point number (from the ray-object intersection), a mapping scheme is used where the original floating point value can be retrieved so that it is possible to composite depth images that are stored with differing numbers of bytes per pixel. This is done by storing the minimum and maximum depth values of the scene in the depth map header. The floating point depth value is then mapped to the  $2^{8n}$  values, with the  $2^0$  value indicating pixels with the minimum depth, the  $2^{8n} - 1$  value indicating pixels that have the maximum depth value, the  $2^{8n}$  value indicating that the pixel has no depth information (that is no ray-object intersection was found), and other floating point depth values mapped to the appropriate integral value by the formula  $V = ((fp - min)/(max - min)) * 2^{8n} - 1$ , where  $V$  is the integral value,  $fp$  the floating point value,  $min$  and  $max$  are the minimum and maximum depth values, respectively. While there are many other possible methods for storing depth data, this is the one used by *optik*.

The way the depth values are computed can also be controlled by the user, allowing for some flexibility in using *optik* depth files with other depth file formats. The user may choose that the floating point depth value be stored as the distance along the ray from the eye through the pixel (as determined by the ray-object intersection calculations), or this value may be projected onto a vector perpendicular to the eye plane before being stored in the depth file. Various methods of obtaining the depth information when oversampling (as discussed in *Chapter 3*) are also implemented, namely taking the minimum depth



from all samples, the maximum depth from all samples, or choosing a consistent sample number.

Additional modifications have been made in the course of work on this thesis. A provision has been made to allow *optik* to directly read in depth map files, automatically retrieve world coordinates from them, and send shadow rays from each of these points to the light sources defined in the depth map header. These rays are then used to check for intersections with reconstructed objects (read in as a group of *optik* objects). This version of *optik* also creates a shadow map, of the form discussed in this thesis, with values that depend on the outcome of the aforementioned intersections.

## A.2 Other uses of depth files

There are other uses that can be made of the depth information besides employing it for compositing purposes. Potmesil and Chakravarty ([potm82]) use depth information to create a depth of field affect in images. Duff in [duff85] uses the depth information to create a fog effect. Each of these methods have been implemented in the course of the work on this thesis, but because they are not directly relevant to the subject of this thesis, and because the original implementations are documented elsewhere, the details do not have to be presented here.

Other possible application of depth files can be seen when considering applications of various effects at the object or scene level. Because the depth values give partial object information, it may be possible to create postprocessed effects that simulate those created as modeled effects. For example, treating each depth value and corresponding pixel as a separate object, some particle system calculations could be performed, allowing objects to explode, drop, bounce, and so forth, without a need to know any extra information about a scene. Glows around objects calculated from the depths might be a useful effect.

These are just some of the uses to which depth files might conceivably be put, and it is hoped that some effort may be put into the exploration of said uses at a later date, or by others.