Real-time Augmented Reality

Laboratory for Computer Vision and Media Technology

Aalborg University · AAU 2004

Group 922 - Mikkel Sandberg Andersen and Tommy Jensen

FACULTY OF ENGINEERING AND SCIENCE
AALBORG UNIVERSITY

INSTITUTE OF ELECTRONIC SYSTEMS

TITLE:

Real-time Augmented Reality.

TIME PERIOD:

9th semester,
September 2nd to January 4th, 2005

PROJECT GROUP:
922 - 2005 CVG9

GROUP MEMBERS:

Mikkel Sandberg Andersen

Tommy Jensen

SUPERVISOR:

Claus Brøndgaard Madsen

NO. OF COPIES: 5

NO. OF PAGES IN MAIN REPORT: 127

NO. OF PAGES IN APPENDICES: 13

NO. OF PAGES IN TOTAL: 140

ABSTRACT:

The purpose of this project is to implement a real-time Augmented Reality (AR) system and to explore the possibilities of moving towards photo-realistic rendering in AR.
Relevant factors, including state of the art, of moving towards photo-realistic rendering are explored, and two areas are chosen, illumination of virtual objects with the use of the Irradiance Volume and shadow casting based on a radiosity solution. Both methods are image based, and thefore a solution to automatically generate environmetal maps with a tracker and a camera is explored.
An AR framework has been developed. Shading of arbitrary diffuse objects, with the use of the Irradiance volume has been implemented. Furthermore shadow casting has been implemented through a reduced radiosity solution. A standalone application for automatic generation of spherical environment maps, has also been developed.
The system produces a good global illumination approximation, that can be rendered real-time. Furthermore the shadow casting developed for the system, is able to create simple soft shadows in the augmented scene, at reduced performance.

Synopsis

Formålet med dette projekt er at implementere et realtids Augmented Reality (AR) system og undersøge mulighederne for at flytte et sådant system i retningen af fotorealistisk rendering.

Relevante faktorer, inklusive state of the art, for at gå mod fotorealistisk rendering udforskes og to metoder vælges. Illuminering af virtuelle objekter ved brugen af Irradiance Volume og skyggekastning baseret på en radiosity løsning. Begge metoder er billedbaserede, så derfor undersøges muligheden for a generere anvironment maps automatisk når et kamera med påmonteret tracker er til rådighed.

Der er, i dette projekt, udviklet et AR framework. Shading af arbitrære diffuse objekter ved hjælpe af Irradiance Volume er implementeret. Ydermere er skyggekastning implementeret gennem en reduceret radiosity løsning. Der er yderligere lavet en applikation til, automatisk, at generere sfæriske environment maps.

Systemet er i stand til at producere en god global illumineringsapproksimering, der kan renderes i realtid. Yderligere er skyggelægningen, der er udviklet, i stand til at lave simple bløde skygger i en augmenteret scene, dog med reduceret ydeevne.

Preface

This report is the documentation of the work of group 922, 9th semester specialisation in Computer Vision and Graphics (CVG) 2004, Laboratory of Computer Vision and Media Technology, Aalborg University.

The target audiences for this report are people with an understanding of 9th semester Computer Vision and Graphics, or people with knowledge or an interest in the field of real-time, image based, ilumination and shadowing in Augmented Reality.

The report is comprised by four parts: Introduction, Methods, Results and Discussion, and Appendix. The appendix is placed last in the report as is the over all class structure of the system.

Source references are on the Harvard-form, an example of this is: [Sherman and Craig, 2003]. Chapters and sections starting with a source reference is based on this source.

Attached is a CD-rom on which the following can be found:

- Report
- Source code
- Source binaries
- Class diagram

Aalborg University, Denmark. January 4th 2005.

Mikkel Sandberg Andersen

Tommy Jensen

Contents

I Introduction **11**

1 Introduction **13**

1.1 A short history of computer graphics 13

1.2 Virtual or Augmented reality? 13

1.3 Applications of Augmented Reality 15

2 Moving towards photo-realism in Augmented Reality **17**

2.1 Human Visual Perception 17

2.2 Illumination 19

2.3 Illumination Environments 20

2.4 Shadowing 21

2.5 Other considerations 22

2.6 Example of an indoor scene 23

3 Defining the project **25**

3.1 The Aim of the Project 25

3.2 Technology 26

3.3 System description 27

4 Results preview **31**

II Methods **37**

5 Camera **39**

5.1 Camera calibration 39

5.2 Vignetting 41

6 Tracking **43**

6.1 The Polhemus Isotrak2 43

6.2 Euler Angles 43

6.3 Calculating orientation vectors 44

6.4 Other issues in tracking 45

6.5 Tracker data in this project 45

7 Pre- and Post-processing **47**

7.1 Full-screen Aligned Quad 47

7.2 Occlusion Handling 48

7.3 Anti-aliasing 50

7.4 Motion Blur 51

7.5 Video Noise 52

8 Modeling Real Scene Illumination **55**
8.1 Image-Based Lighting . . . 56
8.2 The Applied Method for the system . . . 59
8.3 Creating an environment map . . . 60
8.4 Mapping environment to Geometry . . . 62
8.5 Updating an Environment Map . . . 65
9 Illuminating a scene **67**
9.1 Direct Illumination . . . 68
9.2 Global Illumination . . . 70
9.3 Illumination algorithms used for the system . . . 74
10 Irradiance Volume **77**
10.1 Radiance and Irradiance . . . 77
10.2 Creation of an Irradiance Volume . . . 78
10.3 Use of Irradiance Volume in the system . . . 83
11 Radiosity **89**
11.1 Radiosity matrix . . . 89
11.2 Progressive refinement . . . 90
11.3 Patch subdivision . . . 90
11.4 Accelerating Radiosity . . . 93
11.5 Form factor calculation . . . 95
11.6 Use of Radiosity in the system . . . 96
12 The Final system **101**
12.1 Offline process . . . 101
12.2 Online process . . . 101
III Results and Discussion **105**
13 Test **107**
13.1 Irradiance sampling . . . 107
13.2 Irradiance shading . . . 108
13.3 Radiosity Shadows . . . 110
13.4 Flexibility . . . 113
13.5 Comparison with rendered image . . . 115
13.6 Performance . . . 116
14 Conclusion **119**
14.1 Purpose of the project . . . 119
14.2 Methods . . . 119

14.3 Results 120
14.4 Future possibilities 121
Bibliography **122**
IV Appendix **125**
A Euler Transform **127**
B Lambertian Shading **129**
C Intersection testing **133**
D System Class Diagram **138**

CONTENTS

Part I

Introduction

CHAPTER 1

Introduction

1.1 A short history of computer graphics

Computer graphics has seen an impressive development within the last few decades. The introduction of the personal computer has revolutionised the way computer graphics is thought of. Initially driven by an academic interest, real-world graphical applications, such as computer games, became wide spread during the late eighties and early nineties. The ground was set, when the first dedicated graphics accelerator chip was introduced in the mid ninties, facilitating extra processing power to create impressive graphics. The Graphics Processing Unit (GPU), a term coined by NVIDIA Corporation by analogy to the CPU, introduced a separate programming capability which initially was ably to carry out simple tasks, but has evolved significantly and still is evolving in the direction of a higher level of programmability.

The development of computer graphics has had two separate directions for some time. One direction creating - low resolution - real-time 3D animated graphics used in virtual reality applications, and one creating high quality photometric imagery e.g. as used in the film industry. Evidently, high quality is also desirable in real-time 3D animations but until recently both CPU and graphics hardware has not been able to provide sufficient computational power. Although it is still not possible to implement the methods of very high quality Computer Generated Imagery (CGI), it is feasible that it is only a question few years before the two directions will merge into one.

1.2 Virtual or Augmented reality?

The term Virtual reality (VR), introduced by Jaron Lanier in 1989, describes an environment that is simulated by a computer. VR can be displayed in several ways. Projecting the 3D world onto a regular 2D screen or by providing a 3D view with either the use of stereoscopic goggles, so-called head mounted displays (HMD's) or passive/active stereo imaging (e.g. like the VR Media Lab 6-sided CAVE). VR usually provides some degree of manipulating the virtual world, either by the use of a standard keyboard or more advanced sensory devices such as a 3D mouse (e.g Wanda) or vision based gesture recognition.

Augmented Reality (AR) was introduced by [Wellner, 1993] in 1993 as the opposite of virtual reality. Instead of immersing the user into a purely synthetic world, the purpose of AR is to augment the real world with some sort of additional information, such as virtual objects. Milgrams Taxonomy for mixed reality [Milgram and Kishino, 1994] defines the following three axes.

- Extend of world knowledge (Tracking)
- Extend of presence (level of immersion)
- Reproduction Fidelity (Computer Graphics)

To visualise the relationship of AR and VR, [Milgram and Kishino, 1994] presents the following relationship:

Real World (telepresent) - Augmented Reality - Mixed Reality - Augmented Virtuality - Virtual World (immersed - synthetic).

The real-time branch of AR can, as well as VR, allow for manipulation of the virtual objects. AR finds application in many areas including non-realtime rendered movies and augmented still photos. In Figure 1.1 and Figure 1.2 examples of non real-time and real-time respectively Augmented Reality.

Figure 1.1: An example of non-real-time augmented reality used in the film adaptation of J.R.R. Tolkien's Lord of The Rings. (Image by New Line Cinema)

Figure 1.2: An example of real-time augmented reality. (Image by [Gibson et al.,])

An additional element of AR, with respect to VR, is how to involve the real world. This can be done by the use of a Head Mounted Display with the ability to project an image into the visual path leading to the viewers eye. Another approach is to record the real world with a camera, adding the augmentation

and displaying this on a 2D screen. The fact that AR mixes real world with virtual objects requires much attention in the area of simulating real world features like light and shadows, as to enable the virtual objects to seamlessly blend into the real world.

Much research work in AR has been performed in the last decade, but for some reason AR has not reached the same level of popularity as Virtual Reality. The reason for this can be found in both the preparation and actual execution of an AR-system. Usually AR involves costly tracking and display equipment and a large amount of labour in producing the prerequisites such as a 3D model of the augmented area, setting up the augmented area with controlled lighting, etc. This, among other factors, is the reasong AR systems have been restricted to research facilities.

1.3 Applications of Augmented Reality

The applications for Augmented Reality extends into many areas.

One area where 3D graphics is already used extensively is in architectural visualisation and urban planning. This enables users to see a given construction in situ. This provides architects and urban planners to evaluate the consequences, with respect to e.g. the surroundings, of a given construction layout. An illustration hereof can be seen in Figure 1.3.

Figure 1.3: An example of Augmented reality in urban planning. (Image by [ARTHUR, 2004])

An emerging area is the use of AR in education and entertainment, or combined the so-called edutainment. Imagine when visiting a historical site it will be possible to experience a reconstruction of a historical battle first-hand, or see how ruins looked originally. One of the fastest growing industries these days is the computer gaming industry. The industry has already embraced Virtual Reality, so AR seems to be the next natural evolutionary step in computer games when the technique has matured. With AR a new level of interactivity, feedback, and realism may be reached.

CHAPTER 2

Moving towards photo-realism in Augmented Reality

The purpose of this chapter is to introduce several concepts in the area of Augmented Reality (AR). Furthermore some of the problems in creating photo realistic AR will be described.

2.1 Human Visual Perception

All though the Photo Realistic rendering field is built on a foundation of mathematical and physical formulations, human visual perception plays a central role in the creation of convincing Computer Generated Imagery (CGI). This is especially the case in AR, where CGI is required to blend seamlessly with real imagery. Hence, developing a basic understanding of human visual perception as a first step in this project is relevant. Given the complexity and breath of this subject, only a few aspects of the topic will be covered.

2.1.1 Image formation in the eye

Source: [Gonzalez and Woods, 2002]
Figure 2.1 shows a simplified cross section of the human eye. The eye is nearly a sphere, with an average diameter of app. 20 mm. The eye is enclosed by three membranes, with incoming light passing the cornea, the lens and then finally hitting the retina, located at the back of the eye.

When the eye is properly focused, light from an object outside the eye is formed as an image on the retina. Image sensing is afforded by the distribution of discrete image receptors on the retina. There are two classes of receptors: Cones and rods. The cones are located in the central area of the retina and are very sensitive to colour, there are between 6 and 7 million cones in each eye. The number of rods is much larger, some 75 to 150 million are distributed over the entire retinal surface in each eye. The rods do not sense colour, but senses low levels of illumination. The rods serves to give an overall picture of the field of view.

2.1.2 Image perception

Source: [Gonzalez and Woods, 2002]
While the eye on the surface may share similarities with the electronic equivalent, the camera, it is not quite as simple. One part of the human perception is the image formation, another is the way images are interpreted by the human brain. There are several human perception phenomena which cannot yet be fully explained, an example of one of these follows.

Because a digital image is displayed as a discrete set of intensities, the eyes' ability to discriminate between different intensity levels is an important consideration when presenting computer generated

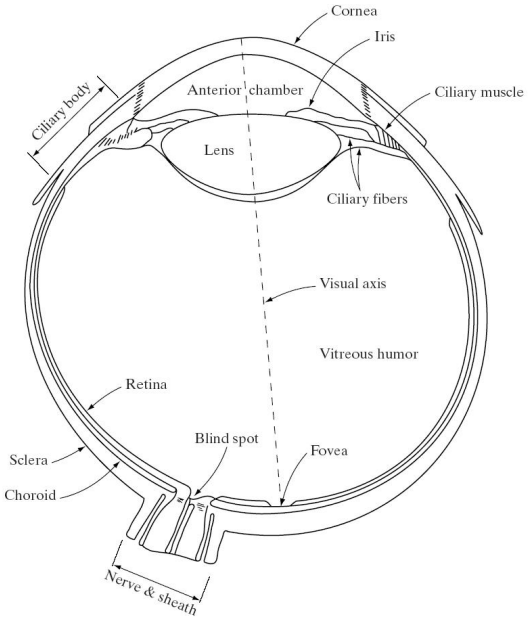
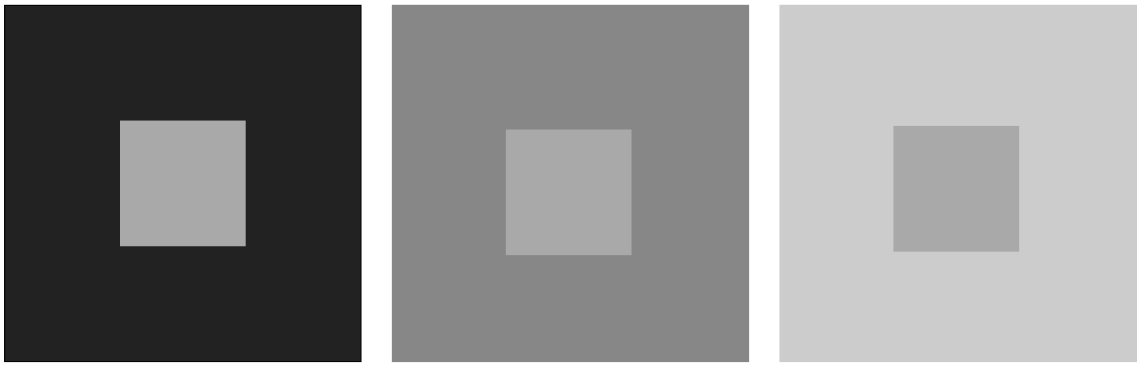


Figure 2.1: A cross section of the human eye (image by [Gonzalez and Woods, 2002]).

imagery. An example on a phenomena, related to this is called simultaneous contrast, can be seen in Figure 2.2. It is related to the fact that a region's perceived brightness does not depend simply on its intensity.



(a) *(b)* *(c)*

Figure 2.2: Examples of simultaneous contrast. The inner squares all have the same intensity, but they appear progressively darker as the background becomes lighter (image by [Gonzalez and Woods, 2002]).

Conclusively, it is clear that human visual perception is not simply a question of physically correct models of light transport in a given scenario. A significant part of human perception is controlled by human recognition, interpretation, and experiences. Thus an important consideration in creating photo realistic AR is not necessarily to create a physically correct rendering, but most importantly to create images that appear correct to the user.

2.2 Illumination

Light is the basis of all vision, and following also in computer graphics. Light is included in the complex area of quantum physics. All-though this is a complex area, is is important to develop a basic understanding of the principles of light behaviour.

Light is electromagnetic radiation with a wavelength ranging from ultraviolet at 100 nm up to infrared at 1 mio. nm. In between lies the spectrum of light that is visible to the human eye, which ranges from about 380-770 nm.

Through time there have been many suggestions as how to describe the nature of light. The modern theory is the wave-particle duality, introduced by Einstein in the early 1900s, which is a consequence of quantum mechanics. This theory holds that light and matter simultaneously exhibits properties of waves and particles (photons). This means that some effects of light cam be explained through wave theory, e.g. reflection and diffraction, and other effects, e.g. absorption, due to its particle nature.

This in consequence means, that what the human eye see, is a result of emitted light particles (photons) interacting with the surrounding environment. In general, light leaves some light source, e.g. a lamp or the sun, is reflected from many surfaces and then finally reflected to our eyes, or through an image plane of a camera.

Within computer graphics, illumination is often divided into two categories, local and global illumination.

The contribution from the light that goes directly from the light source and is reflected from the surface is called "local illumination". So, for a local illumination model, the shading of any surface is independent from the shading of all other surfaces.

A "global illumination model" adds to the local model the light that is reflected from other surfaces to the current surface. A global illumination model is more comprehensive, more physically correct, and produces more realistic images.

Some of the effects that are caused by interaction between light and surfaces are listed here:

- Reflection
- Refraction
- Diffraction
- Dispersion
- Polarisation
- Coherence
- Scattering
- Shadowing

These are all relevant contributors to the lighting conditions of a given scene. They must therefore be modelled and applied when required. Obviously, taking into account, all aspects of the physical structure of a scene and the objects within the scene, requires both a great deal of information and also a great deal of computational power to process this information.

To reduce the amount of information to be stored, there are several ways to approximate the lighting conditions. These methods are categorised as light source estimation methods. An approach to modeling scene illumination is the image based lighting. This requires image data covering the entire augmented location, viewed from some central location. The image data is analysed to determine how to model the real scene illumination so that it can be recreated and used when lighting virtual objects. Image based lighting usually involves the steps found in Figure 2.3.

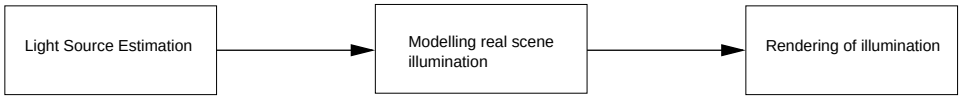


Figure 2.3: The process of image based lighting.

2.3 Illumination Environments

Illumination in AR can roughly be divided into two categories of scenes which are subsets of a general illumination case. These are non enclosed spaces with very few local illumination sources meaning almost all illumination is global illumination. The other category includes enclosed environments with a high amount of surfaces and many local illumination sources. Figure 2.4 illustrates these two scenarios exemplified by an indoor and an outdoor environment.

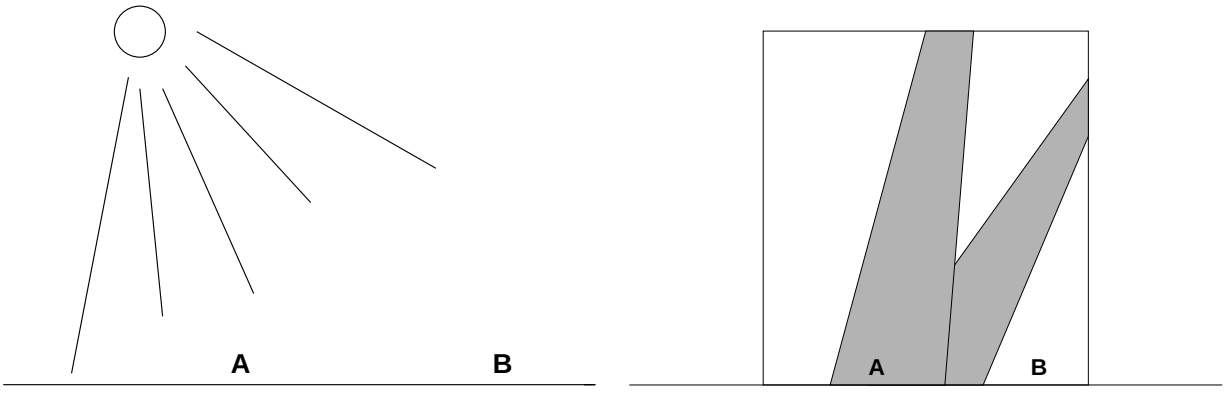


Figure 2.4: (a) An outdoor scene lit by with primarily global illumination. (b) An indoor scene lit by both global illumination and several local illumination sources.

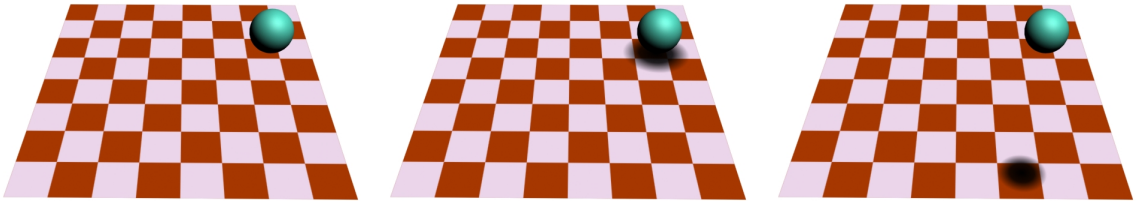
The two scenarios in Figure 2.4 differ in complexity. The indoor scene being significantly more complex. In the outdoor scene spatial lighting conditions will not differ significantly because the illumination source is distant. The opposite is the case in the indoor scene. The reason for this is that illumination is based on local illuminations sources, windows and lamps, and also global sources, reflective surfaces.

Consider points A and B in Figure 2.4.a. The primary illumination source is the sun and derivatives of sunlight, e.g. scattered light from the atmosphere, and reflections from surroundings. Since all light sources are distant, placing an object in either of the points will only affect lighting conditions minimally, if at all. The case is the opposite in Figure 2.4.b. As can be seen from the illustration, the lighting conditions in points a and b differ significantly. Partly because of sunlight becoming apertured

by the windows, creating directed light, but also because objects are close to both the direct light sources and the indirect, such as walls and floors.

2.4 Shadowing

Shadows are an important element of realism in Augmented Reality. In 3D images shadows are essential to provide the viewer with visual cues about object placement. In Figure 2.5 the importance of shadows in providing a sense of object placement can be seen.



(a) *(b)* *(c)*

Figure 2.5: Three illustrations of shadow importance. The object has the same placement in all three figures. (a) Object with no shadows. Viewer has no sense of placement. (b) Object is distant from the viewer. (c) Object is closer to the viewer.

The terminology used in this section is illustrated in Figure 2.6. Occluders are objects which cast shadows onto receivers. Umbra is a fully shadowed region and penumbra a partially shadowed region.

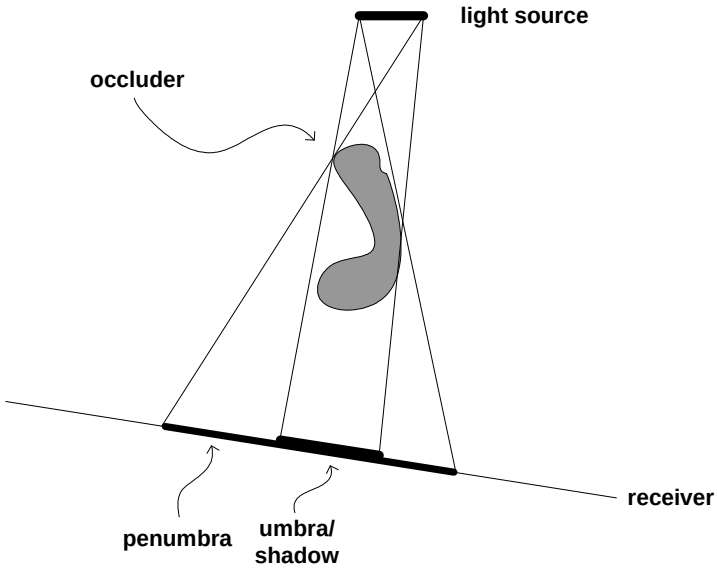


Figure 2.6: Shadow terminology: light source, occluder, receiver, shadow, umbra, and penumbra. (Image by [Akenine-Möller and Haines, 2002]

When dealing with real world shadows, many of the principles in shadow methods often used in real-

time graphics will not appear realistic, because they are constructed for a limited number of ideal point light sources. Complex real world lighting scenarios are composed of many area light sources. Outdoor scenes are lit by both direct sunlight but also atmospherically scattered sunlight, which is a very large area light source. In indoor scenes walls, floors and windows all constitutes light sources. All of these factors result in both multiple shadows and shadows with soft edges.

Generating correct shadows calls for a detailed analysis of a given scene, determining which surfaces, both light sources and reflective surfaces, contribute to the irradiance in a given point in the scene. among the methods for doing exactly this is Radiosity.

2.5 Other considerations

2.5.1 Occlusions

Enabling real world objects to occlude virtual objects and vice versa, is essential in creating convincing Augmented Reality, but there are several things working against attaining a perfect occlusion handling. Ideally it is possible to model every aspect of an scene to be augmented. This is feasible for a controlled laboratory setup, but quite difficult when dealing with locations outside the laboratory. Some of the major problems in this field are

- Steady tracking of camera or Head Mounted Display to ensure perfect alignment of occlusion mask
- Level of detail of occlusion mask vs. system performance
- Dynamic changes in the scene, e.g. people walking through the scene

Not all of the above apply to both outdoor and indoor scenarios, previously mentioned in section 2.3. The complexity of occlusion grows with the complexity of the chosen scene. Thus an indoor scene may require more elaborate 3D models than an outdoor.

2.5.2 Reflections

When dealing with computer graphics it is often convenient to assume that all surfaces are opaque and diffuse, meaning that no light is refracted and they reflect equally much light at any point. This is far from the case in a real world scenario. Often a given scene is rich on surfaces which are to some degree reflective, be it a glossy tile floor, window glass, water or actual mirrors. Realism in scenes with many reflective surfaces is dependent on whether the virtual objects behaves as real objects, i.e. are reflected on relevant surfaces.

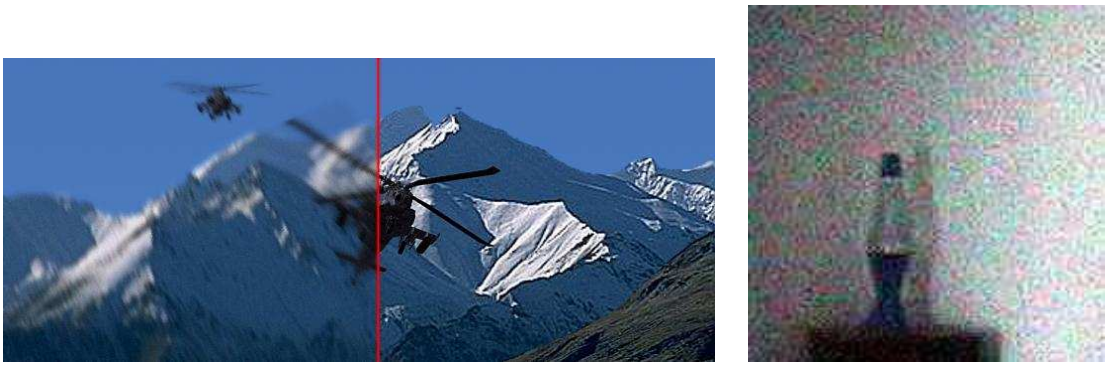
2.5.3 Camera phenomena

Two phenomena inherent in video footage are motion blur and video noise. These do not exist in computer generated imagery and as a consequence must either be removed from the source footage or applied to the rendered graphics.

Motion blurring is caused by the movement of an object across the screen during a frame. This is due to the fact that a video image is not a picture of an infinitesimally small time unit but rather the result of an

exposure of normally several milliseconds, determined by the shutter speed. Motion blurring appears as smeared streaks across the image. This effect cannot easily be removed but it can be imitated in computer graphics.

Video noise is caused by both static induced onto the imaging plane (Charge Coupled Device (CCD)) and compression errors in the camera. Video noise appears as both a grainy and unsettled image. This is difficult to remove, but can also be imitated and applied in post processing to the virtual elements in the scene.



(a) *(b)*

Figure 2.7: Two figures illustrating common camera phenomena. (a) Motion blur. (b) Video noise, (image by [Ruether, 2002]).

A phenomena which is apparent in computer graphics is aliasing. This effect results in jagged edges and can clearly be seen on any computer generated edge, which is not parallel to any of the two axes. The effect is not nearly pronounced in video footage, since edges are blurred during the image forming process. It is possible to generate anti-aliased graphics, this is important for the augmented objects to appear as a part of the real scene.

2.6 Example of an indoor scene

In Figure 2.8 an example of an indoor scene can be seen. The scene provides examples of nearly all phenomena described in the previous sections.

The scene is illuminated by several light sources. Large window sections and skylights provide sunlight during daylight, else incandescent lights provide light. The white tile floor is a considerable indirect light source. Shadowing is complicated due to the many direct and indirect light sources. Objects in the scene cast multiple shadows with soft edges.

Occlusions are fairly straightforward when dealing with straight lines and other inorganic shapes, thought it is worth noting that creating occlusions with the lamps seen in the figure is highly dependent on precise tracking and the positioning of the camera. Humans walking through the scene constitutes highly organic and dynamic shapes which are near impossible to model in advance, which in terms make occlusions highly difficult.

Finally, this particular scene is rich on reflective surfaces. The white tile floor is somewhat reflective, while the glass sections are highly reflective.

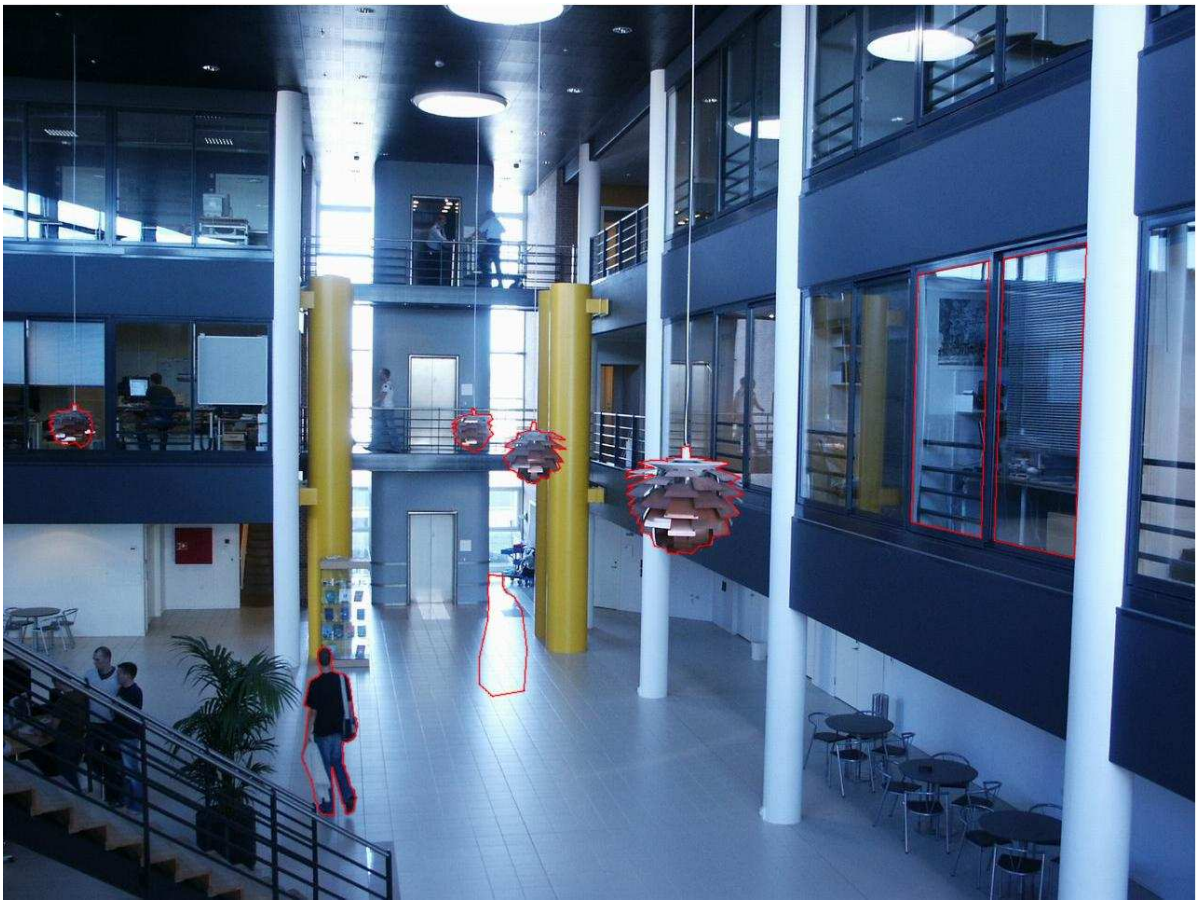


Figure 2.8: Indoor scene used in the project. Problem areas are marked in red.

CHAPTER 3

Defining the project

3.1 The Aim of the Project

The goal of this project is to implement a real-time Augmented Reality (AR) system. The objective of the system will be to place virtual objects in a real scene, enabling these to interact with existing light, shadow, and occlusion. The user will be able to interact with both the viewing direction and the virtual objects.

Amongst the several problem areas within the development of a real-time AR system, select areas will receive in-depth attention in this project, these areas are concentrated in the field of creating convincing illumination and shadowing in real-time. This area has been chosen, as it is an important part of seamlessly blending virtual objects with the real world.

A phenomena within real-time illumination that currently receives much attention in research, is Global Illumination (GI) which is a general term for methods that approximates real world illumination. Real world illumination is not a trivial task. Complex interaction between not only direct but also reflected light and light scattered within materials requires both complex mathematical models of the world and processing of a large amount of data for each picture rendered.

To make real-time GI feasible, an approximation to the actual lighting conditions in the scene is needed. Usually this can be accomplished by analysing a panoramic (360° x 180°) still image representing the entire surrounding environment, a so-called environment map. This method has no way of handling dynamic changes in the lighting conditions. This project will describe a method for dynamically updating an environment map, so as to be able to update the approximation the the lighting conditions.

An important factor in computer graphics are shadows, as they provide the viewer with visual cues about object placements. Methods for generating believable image based shadows in real-time will be investigated.

The real-time demands in the context of this system are soft real-time demands. This, in consequence, means that it is necessary for the viewer to be able to interact with the system in a way that does not impair the illusion of the virtual objects blending seamlessly with the real scene. To achieve this effect we have decided that a frame rate of at least 20 frames-per-second is required.

The effort in this project will be in investigating the existing state of the art within the areas described in this section, and how to combine these to achieve a functioning system. The focus of this project will be on evaluating how these methods work in a collaborative system. Furthermore it will investigate how to implement an AR system that will function in a non-controlled indoor environment with respect to dynamically changing light conditions.

This project is limited to rendering diffuse objects. This means that reflective surfaces, like metals, and translucent/refractive objects, like glass, can not be rendered. This delimitation results in a substantial reduction of system complexity.

Furthermore, the interaction with the environment of the augmented scene in this system will be limited to virtual objects casting shadows onto the real world. This means that virtual objects will not cause

reflections on surrounding real world objects, neither will they cause effects such as colour bleeding.

Virtual object interaction is limited to collision detection. This means that virtual objects will not be able to cast shadows on each others, neither will the cause other effects such colour bleeding.

3.2 Technology

The emphasis of this system is, as described earlier, on implementing a real-time system. Keeping the real-time demands in mind, when choosing both operating system, algorithms and code language is essential.

Real-time demands limits both the types of algorithms that can be used and the complexity of the scene we are able to augment. Some algorithms are too computationally heavy to be used in an online system, and are best fitted for offline rendering for e.g. movie effects.

The movie special effects industry adopted the concept of Augmented Reality relatively quickly. while the augmentations in movies often look very convincing, these results are not easily achieved. They are based on 3D models with very high level of detail, and often tracking and occlusions are done manually frame by frame by accomplished professionals. These techniques are not applicable for a real-time system.

The Augmented System in this project seeks to find the balance between advanced illumination and shadowing algorithms and a reasonable frame rate. The primary way of achieving this will be through extensive use of pre-calculated data where it is possible. This means that the system will be comprised of an offline and an online part. The offline part includes the pre-calculation processes of analysing and modeling scene illumination and shadowing. The online part should ideally be reduced to rendering scene graphics, shaded with the help of the pre-calculated data .

Since the main idea of the project is to implement a system, that will function outside a research lab it has been decided that the target platform is standard consumer class workstations. the system is implemented in the object oriented programming language C++, and will run on Microsoft Windows platform. MATLAB will be used for camera calibration.

3.2.1 Hardware

The video feed used in this augmentation process is captured through the use of a Philips ToUCam 840K. The resolution of the video feed is maximally 640x480 pixels at a frame rate of 30 fps. Because the camera is a web cam using the USB 1.1 standard for video transfer, the video data is compressed in the camera, and is therefore prone to a certain amount of video noise. Furthermore the capture pipeline in Windows XP imposes a delay from the time a picture is captured till it is available.

For this system it is necessary to continuously determine the absolute orientation of the camera, as to match both the augmented objects and the 3D scene to the real world. In this system the tracking is done using the Polhemus ISOTRAK 2 magnetic tracking system. This tracker has fairly low accuracy and resolution, but has been chosen due to the fact that it was the best of those readily available.

3.2.2 Graphics API

When choosing a graphics API, there are two major directions: Raytracer based or rasterization based. All major API's these days are based on rasterization rendering, hence API's based on this technique

will be considered.

Currently two major graphics API exists: the Open Graphics Library (OpenGL) and Microsoft Direct3D. Direct3D is an integrated part of the DirextX API, proprietary property of Microsoft, for use on windows machines only. OpenGL was originally developed in 1992 by Silicon Graphics, as a descendant of an API known as Iris GL for UNIX. It was created as an open standard, and is available on many different platforms, including windows and linux. With the release of DirectX8 came the concept of vertex and pixel programs (or shaders), which enables the programmer to replace certain parts of the rendering pipeline with custom code. OpenGL also implements vertex and pixel programs through extensions.

Summarizing, there is no difference in what can be achieved with the two API's. Direct3D has a homogeneous hardware independent extension interface, while OpenGL does not. This means that depending on which extensions are used in OpenGL software will become more or less hardware dependent. For this project OpenGL has been chosen, since the authors have the most experience with this API.

3.3 System description

When designing a live-feed AR-system there are some general problems that needs to be taken into consideration. The system must create a graphics overlay with virtual objects on a live video feed. The position and orientation of the camera providing the video feed is provided by the means of a hardware tracking unit. Occlusion between virtual and real objects are handled through the use of a pre-modeled 3D representation of the scene.

3.3.1 Rendering pipeline

The system consists of two main parts, an off-line that is carried out during system initialisation, and an on-line part, in which the actual rendering takes place. The two augmentation pipelines can be seen in Figures 3.1 and 3.2 respectively.

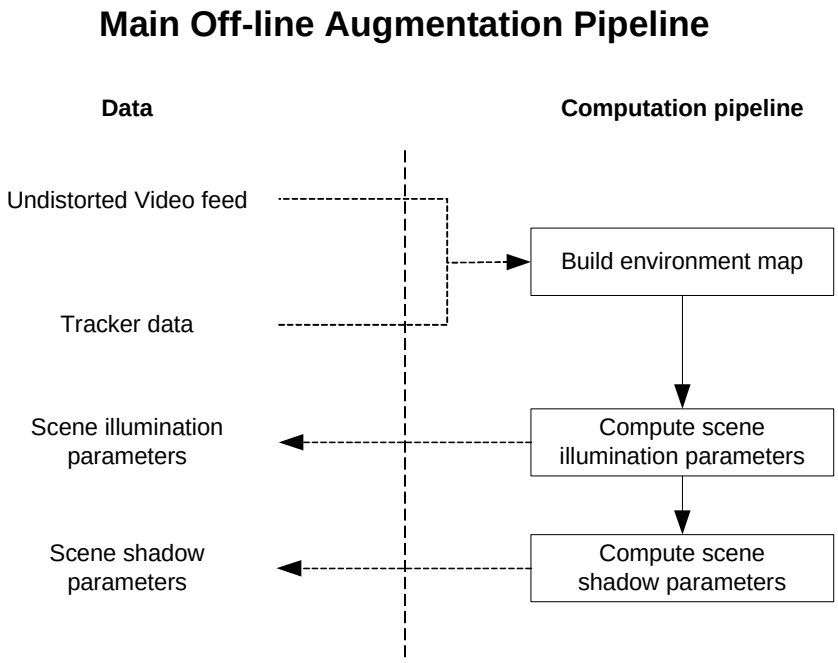


Figure 3.1: An outline of the off-line part of the augmentation pipeline-

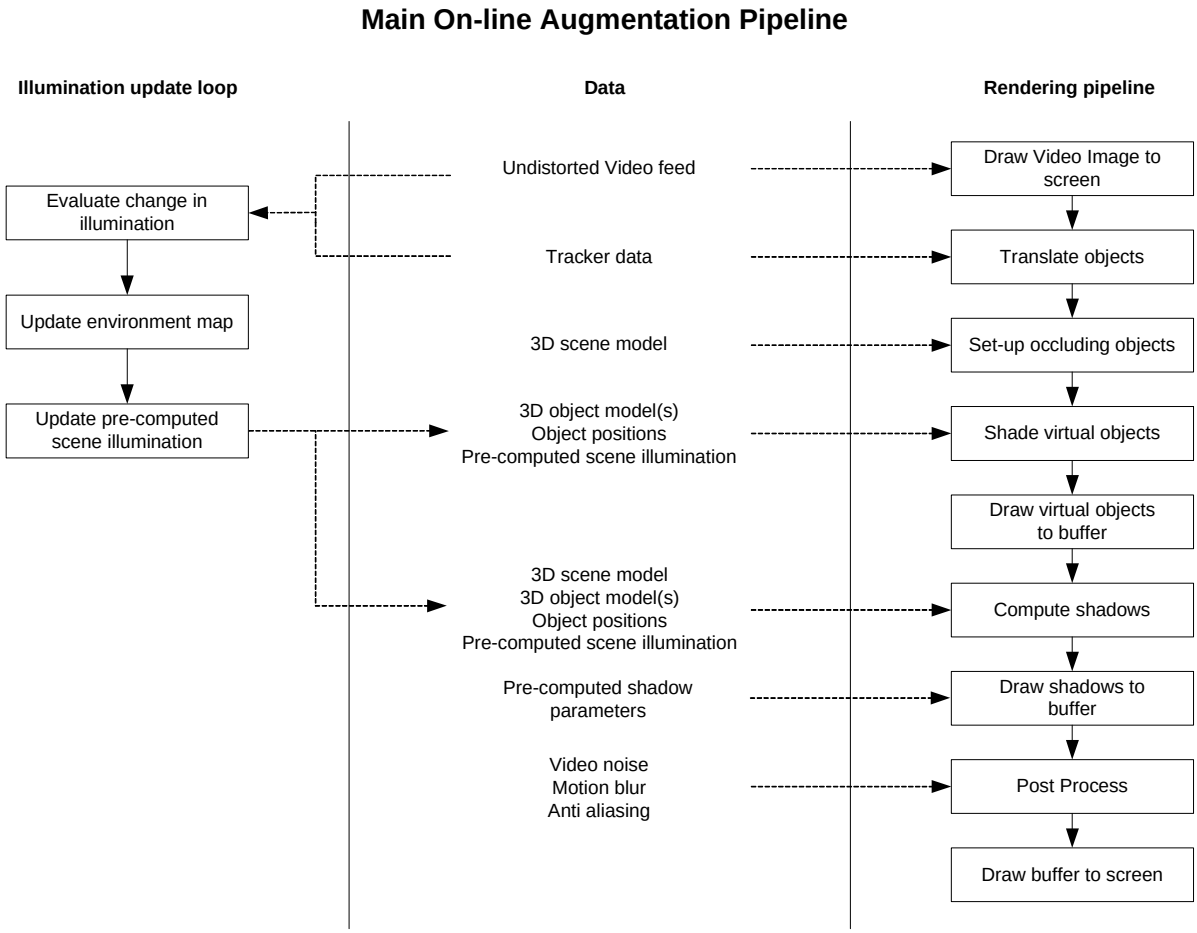


Figure 3.2: An outline of the on-line part of the augmentation pipeline.

CHAPTER 4

Results preview

This chapter presents the results of the project, to give the reader a better understanding and overview of the project before the methods, used to create the images, presented in this chapter, are described in detail.

This project is concerned with the rendering of virtual objects within a real environment, with the intent to create the illusion that the virtual object is a integral part of the environment in which it is placed.

The first step in rendering the objects for the scene, is to create an environment map of the scene. The environment map of the scene used in the results for this project is seen in figure 4.1 on the following page.

To showcase the results of the project a dragon figurine has been selected to perform as test object. In figure 4.2 on the next page and 4.3 on page 33 the dragon is placed at two different positions in the lobby scene selected for the project. The two figures shows that the shading of the object is slightly changed when moved. The shading of the figure is in both cases dominated by the bright floor, while the ceiling with the spotlights is not very significantly represented in the shading. The reason for this is that the method used to shade the objects makes a low resolution sampling of the environment, which seldom samples the directions of the lamps. If the lamps are hit by a sample, it will not be very noticeable if all the adjacent samples are black. This problem can be remedied by using High Dynamic Range Images as environment maps, and by making sure that every light source is sampled. Figure 4.3 also shows that the figurine is occluded by the 3D model of the environment.

In figure 4.4 on page 33 the environment map has been modified increase the light coming from above, to more precisely match the real lighting of the scene. The modified environment map is applied to the shading of the virtual object in the scene, and result is seen in figure 4.5 on page 34, where the object is shaded as if light is coming from above at the same time as the floor is shading from below. A shadow is rendered into the scene, which adds to the illusion that the object is actually placed within the scene. The shadow cast is not an accurate shadow, because the current implementation is limited to support shadow casting using the objects bounding sphere.

In figure 4.6 on page 34 the virtual dragon figurine is scaring an innocent bystander, who is unaware the object in fact does not exist. The object has been toned down with an albedo value to better match the scene.

In figure 4.7 an environment map with a bright sun has been recorded, and the effect on the test object is seen in figure 4.8. Furthermore figure 4.8 shows that the system is not necessarily locked to a certain camera position, if a user wishes to move the camera, it is possible.

In figure 4.9 on page 36 the object has been placed in a new scene shaded by the environment map seen in figure 4.10 on page 36. This is done to show the system is generic, and can be used in a wide range of environments given information about it, like an environment map and a 3D model.

Figure 4.1: The environment map of the scene used in the results.

Figure 4.2: The dragon is placed at two different positions in the lobby scene selected for the project, for shading comparison. Shadows are disabled for the shot.

Figure 4.3: The figurine has been moved to a new position, to show shading changes, and occlusion handling.

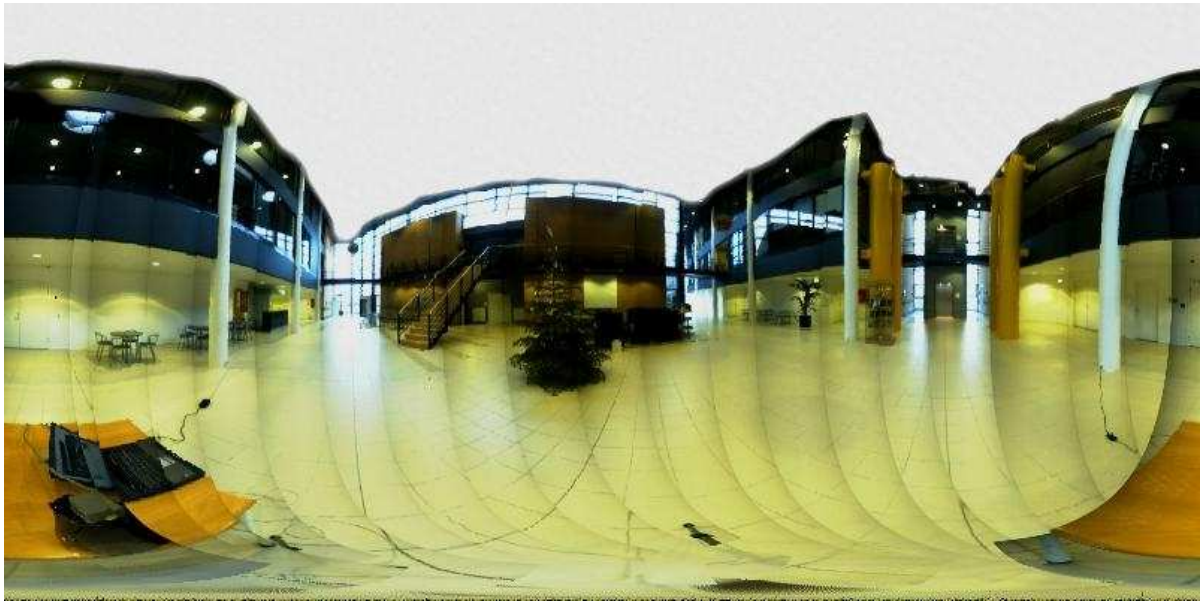


Figure 4.4: The modified environment map

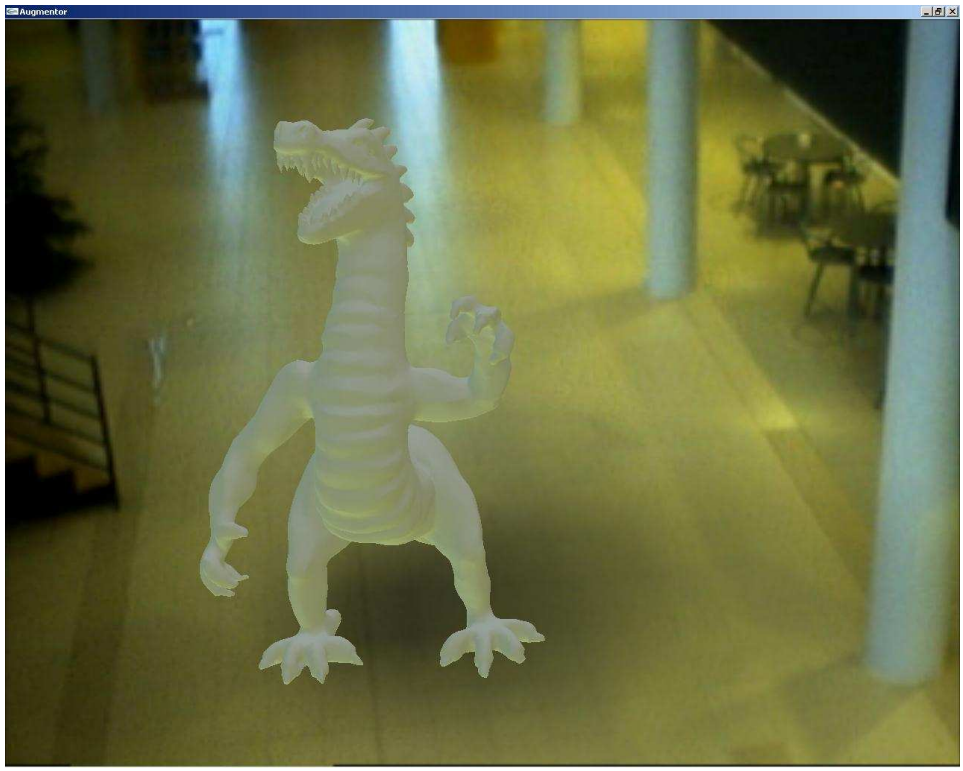


Figure 4.5: The object is shaded as if light is coming from above at the same time as the floor is shading from below, with an added shadow cast by the objects bounding sphere.

Figure 4.6: The virtual dragon figurine is scaring an innocent bystander.

Figure 4.7: An environment map recorded with a bright sun.

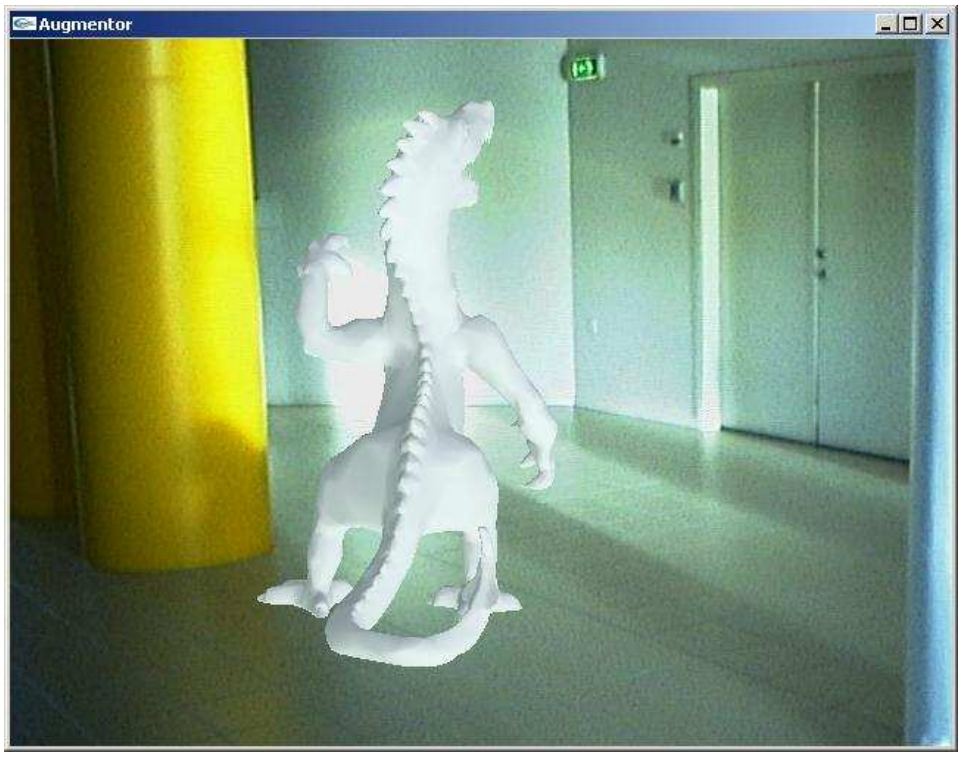


Figure 4.8: The figurine shaded by the bright sun.

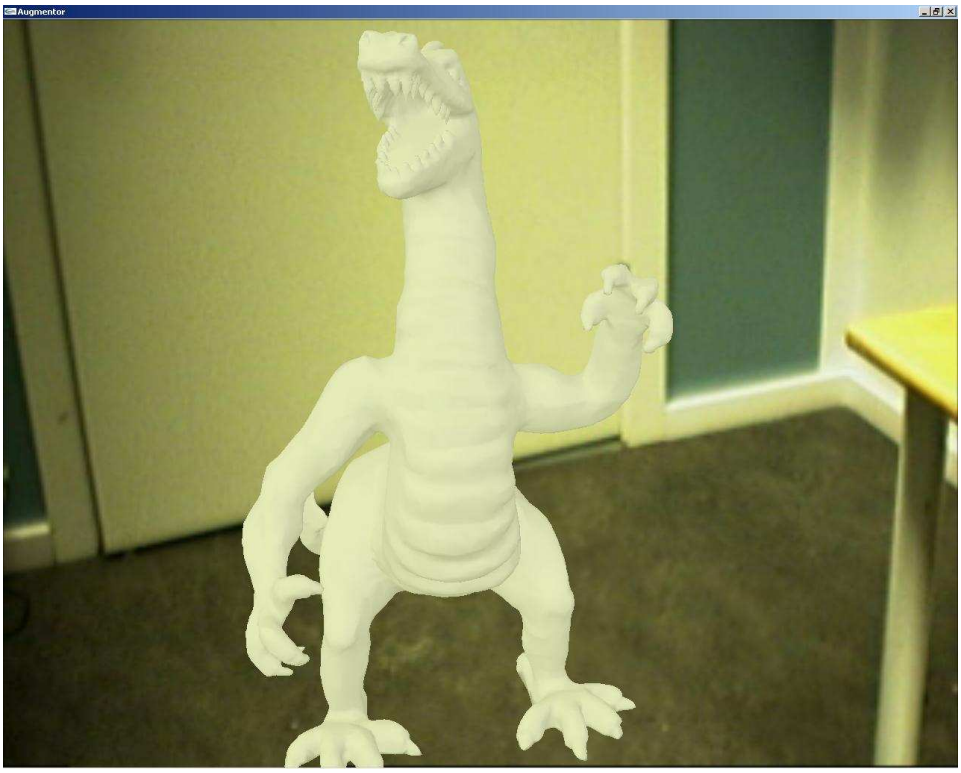


Figure 4.9: The object is placed in a smaller confine, close to walls. The scene is a group room.

Figure 4.10: The environment map for the group room scene.

Part II

Methods

CHAPTER 5

Camera

This chapter will describe why camera calibration is performed. Furthermore some of the problems with the camera that is used are addressed.

5.1 Camera calibration

The image formation in a camera differs significantly from that in a computer graphics rendering system. The real world image formation is affected by real world inaccuracies. This means trouble when computer generated imagery is to be seamlessly blended in with real imagery.

Basically both imaging systems work by the pinhole principle, the difference being that in real world cameras lenses are introduced instead of a small hole. This allows for a greater amount of light to pass onto the imaging plane, thereby reducing the exposure time. An illustration of a simplified camera can be seen in Figure 5.1

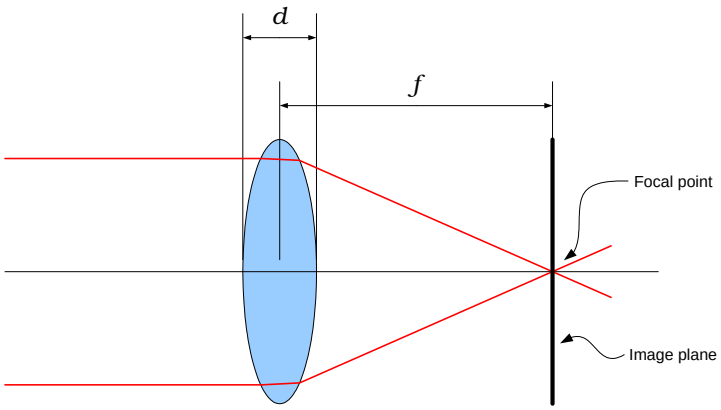


Figure 5.1: A simplified camera model. f is the focal length.

The calibration tool analysed here is “The Camera Calibration Toolbox for Matlab” [Bouguet, 2004] which also has a C implementation in OpenCV. The camera model used in this calibration tool operates with two contributors of distortions in a camera. These are the optics and inaccuracies in the imaging plane, in our case a CCD chip (Charge Coupled Device).

The optics of a camera will mainly cause two types of distortion in an image. Radial and tangential distortion. Radial distortion is caused by the non-linear characteristics of a lens, which means that pixel coordinates in fact represents an arced coordinate system. The tangential distortion is caused by “de-centering” of the lens elements in a compound lens and other manufacturing defects. The tangential distortion is often discarded by the justification that most lenses manufactured these days do not suffer from de-centering.

An exaggerated illustration of these distortions can be seen in Figure 5.2.

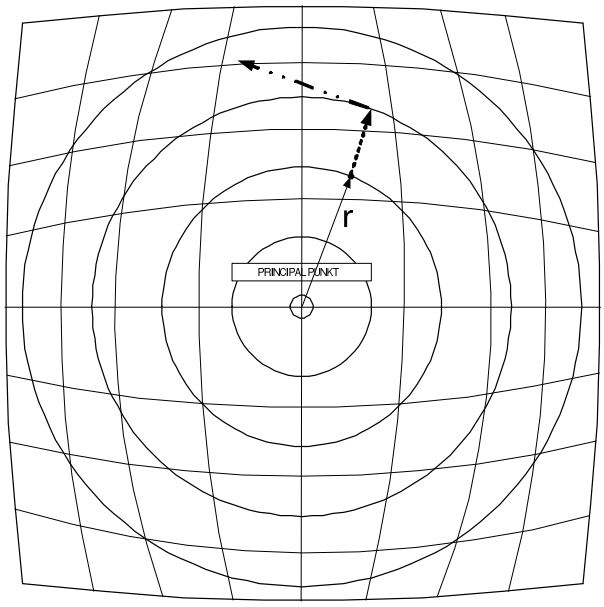


Figure 5.2: Radial and tangential distortion

The Philips ToUCam 840 webcam used for this project clearly shows radial distortion, as can be seen in Figure 5.3.

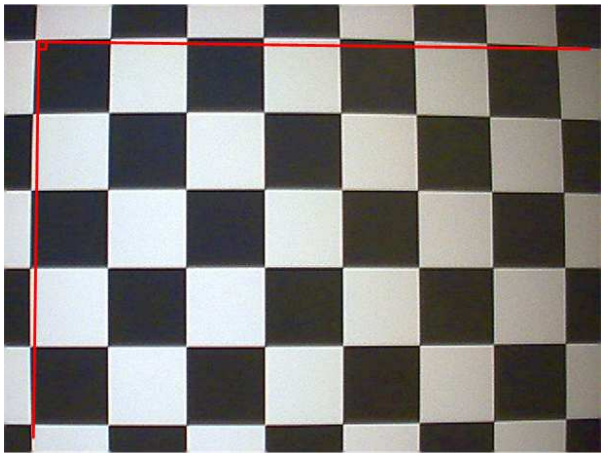


Figure 5.3: A checkerboard image taken with the Philips ToUCam 840. Radial distortion is clearly visible.

The main problem when blending virtual objects into real scenes is the radial distortion. The computer rendered objects do not suffer from distortion which consequently means that there will be a noticeable difference between the real and the virtual objects. The error will be most pronounced when trying to align straight virtual objects to the real scene, these objects could be occlusion planes which are an essential part of augmented reality systems.

There are two solutions to enabling good alignment in an image by using the intrinsic camera parameters. Un-distorting the image or distorting the graphics. The latter would entail rendering graphics to a texture, distorting it and the composite the two images. The first has a ready-to-use solution built into OpenCV.

The calibration will produce a set of intrinsic or internal camera parameters. These are the principal

point which is the center of the lens, the focal length which is the distance from the image plane (CCD) to the principal point. Furthermore there are two sets of distortion parameters, skew coefficients which describe angular skew in the CCD array and the distortions, which are the radial and tangential distortions.

Furthermore the calibration returns the focal length of the camera, denoted f in Figure 5.1. This is used by the undistortion process, but is also used to calculate Field of View (FOV) of the camera. The FOV is used to match the OpenGL camera to the real camera. Assuming that the camera image is equiangular, FOV is calculated using equation 5.1.

$$FOV = 2arctan\left(\frac{u_t}{2f}\right) \tag{5.1}$$

Where u_t denotes the width, in pixels, of the image plane (CCD).

5.2 Vignetting

The webcam used in this project suffer greatly from what is known as vignetting. Vignetting is something that occurs in all photographic lenses. It means the darkening of all edges relative to the center. Usually two types of vignetting occurs: Optical and mechanical vignetting. Optical vignetting is an inherent problem in all lenses. It occurs as a result of the way light is refracted in the lens. Depending on the lens construction the phenomena can be more or less prominent. Mechanical vignetting is caused by light being partially blocked in the camera, this can, among other factors, be caused by misaligned lens hoods. Vignetting is very pronounced in figure 5.3.

CHAPTER 6

Tracking

This chapter will describe how tracking is used in this project. This involves how tracking is performed, and how the data is used to match real world video footage to the virtual world.

To be able to correctly align the graphic overlay in an augmented reality system, it is necessary to determine the exact direction the camera is oriented. This chapter concerns a system consisting of a camera fixed to a orientational tracking unit.

6.1 The Polhemus Isotrak2

[iso, 2001]
The Isotrak2 system by Polhemus is a 6-DOF (Degrees of Freedom) magnetic tracking system. The system consists of a single transmitter, which sequentially generates 3 perpendicular magnetic fields. The technology is based on generating near field, low frequency magnetic field vectors from three antennas in the transmitter. The system can handle 2 separate receivers, which are devices able to detect the field vectors using three concentric remote sensing antennas. This means that each receiver in the system delivers both positional and orientational data.

This project will not use the positional data, only orientation data. The receivers deliver all-attitude angular coverage with a resolution of 0.1°. The static accuracy is 0.75° RMS[1]. The relatively large deviation precludes displaying a completely stable video overlay, but with a sliding window averaging the accuracy will suffice for this project.

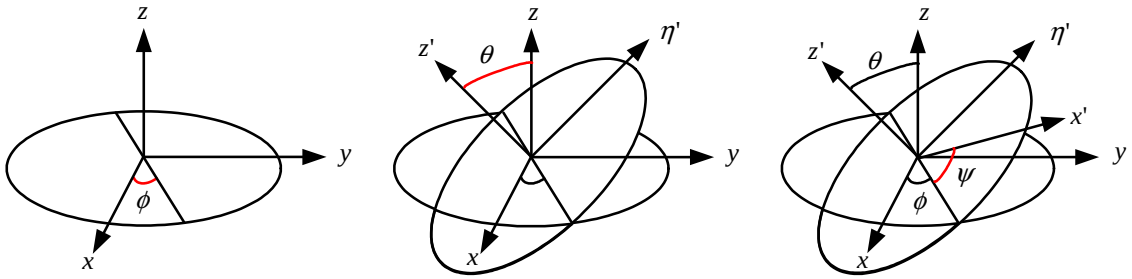
6.2 Euler Angles

[Weisstein, 2004]
For each sample, the Isotrak2 as a standard generates 6 parameters. These are 3 Cartesian, positional, coordinates and 3 orientational angles. The orientational angles are returned in Euler angles: Azimuth, elevation, and roll. According to Euler's rotation theorem, any rotation may be described using three angles (ϕ, θ, ψ).

$$\begin{aligned} \phi &\quad \epsilon \quad [-\pi, \pi] \\ \theta &\quad \epsilon \quad [0, \pi] \\ \psi &\quad \epsilon \quad [-\pi, \pi] \end{aligned}$$

Azimuth, ϕ is a rotation around the z-axis, elevation θ about the x-axis, and roll ψ about the z-axis again. These three rotations are illustrated in Figure 6.1.

[1] RMS (Root Mean Square) in the context of tracking devices is synonymous with standard deviation



(a) *(b)* *(c)*

Figure 6.1: Euler angles illustrated. (a) Azimuth, ϕ (b) Elevation added, θ (c) Roll added, ψ. (Image by [Weisstein, 2004])

The Euler transform can be seen in appendix A on page 127.

Euler angles have some limitations, which involves a phenomena called a gimbal lock. This happens when rotations are made so that one degree of freedom is lost. This occurs when two rotational axes of an object are oriented in the same direction. This will result in wrong rotations.

Another approach to rotations is through the so-called Quaternions. These do not exhibit the gimbal lock problem.

6.3 Calculating orientation vectors

Orientation vectors in OpenGL are two vectors, the target vector, $\vec{t}$, describing in which direction the camera is pointing and the up vector, $\vec{u}$, describing the up direction for the camera. These two vectors are calculated from the Euler angles as shown in equations 6.2 and 6.3.

$$\vec{d} = [0, 0, -1]^T \tag{6.1}$$

The target vector is given by equation 6.2.

$$\vec{t} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{pmatrix} \vec{d}_{.x} \cdot cos(\phi) + \vec{d}_{.y} \cdot (sin(\theta) + \vec{d}_{.z} \cdot cos(\theta)) \cdot sin(\phi) \\ \vec{d}_{.y} \cdot cos(\theta) - \vec{d}_{.z} \cdot sin(\theta) \\ -\vec{d}_{.x} \cdot sin(\phi) + \vec{d}_{.y} \cdot sin(\theta) + \vec{d}_{.z} \cdot cos(\theta) \cdot cos(\phi) \end{pmatrix} \tag{6.2}$$

The up vector is given by equation 6.3.

$$\vec{u} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{pmatrix} sin(\psi) * cos(\phi) + cos(\psi) * sin(\theta) * sin(\phi) \\ cos(\psi) * cos(\theta) \\ sin(\psi) * sin(\phi) + cos(\psi) * sin(\theta) * cos(\phi) \end{pmatrix} \tag{6.3}$$

6.4 Other issues in tracking

Delay is a problem, that is imposed into the frame grabbing pipeline by both hardware limitations and the operating system on the computer. This poses a problem if the tracker in the system is able to provide data faster than the camera. This will result in the 3D models in the augmented system will be ahead of the image data. A solution to remedy this problem is impose a matching delay into the tracking pipeline.

Another issue is the displacement from the rotational points of the camera mount and the image plane in the camera. This will, if not compensated, impose a small error on the actual placement of the camera.

6.5 Tracker data in this project

For use in OpenGL, Euler angles can be described in terms of a rotation matrix as described in 6.2 on page 43, which is multiplied with the OpenGL model view matrix. This will ensure that the OpenGL "camera" is matched exactly to the orientation of the tracker.

Another approach is to calculate a set of two vectors for use with the gluLookAt() function, with one pointing in the direction in which the camera is looking and the other pointing up. This will yield the same result as with the method first mentioned. The latter method is used in this project.

CHAPTER 7

Pre- and Post-processing

This chapter examines which issues that needs to be addressed before and after the virtual object is rendered to the frame, which improves the systems ability to merge real and virtual objects.

To successfully merge virtual and real objects in real-time, there are several issues that needs to be dealt with. The following sections will explore the fundamental pre- and post-processing steps in the augmented reality system rendering pipeline, like the rendering of the video image to screen, the rendering of occlusion objects, anti-aliasing, motion blur to the image, and video noise rendering.

7.1 Full-screen Aligned Quad

Source: [openGL.org, 2004] and [Behrens, 1995]
The system captures video from a camera, and represents it for the user. This section looks into how the video is represented in openGL to always appear on the screen. The camera distorts the video signal as seen in chapter 5 on page 39, and therefore the program must undistort the signal before it is rendered to the screen. The undistorted video-signal needs to be displayed full-screen underneath the computer graphics.
The straightforward way to do this is to set both the Projection and Modelview matrices to the identity, and draw an equivalent GL_QUADS primitive.
The pseudo-code for such an operation would be:

Pseudo code 7.1 The creation of a Full-screen Aligned Quad

1. Push MODELVIEW matrix down by one in the stack and duplicate current matrix.
2. Load identity matrix into MODELVIEW matrix on top of the stack.
3. Push PROJECTION matrix down by one in the stack and duplicate current matrix.
4. Load identity matrix into PROJECTION matrix on top of the stack.
5. Get current image from video capture
6. Render a quad in the z-plane to color buffer.
7. Pop PROJECTION matrix stack, and replace current matrix with the one below it on the stack.
8. Pop MODELVIEW matrix stack, and replace current matrix with the one below it on the stack.

This will render a quad that occupies the entire screen, and whatever projection and model matrices already set up before the operation will be restored.

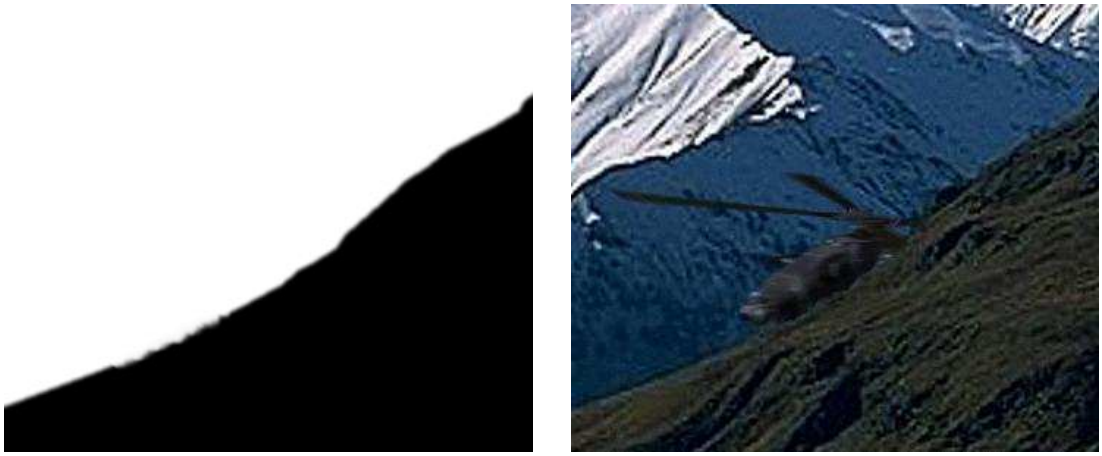
The operation renders the image to the color buffer only, so whatever depth information rendered to depth buffer remains.

7.2 Occlusion Handling

When merging two images, it is desirable to be able to control which parts of one image is to be blended onto the other, and how much they are blended. Masks are the typical way to handle occlusions. An example of this is seen in figures 7.1 and 7.2.

(a) *(b)*

Figure 7.1: Two images, where the virtual helicopter is supposed to be flying around the hill.



(a) *(b)*

Figure 7.2: An example of how a mask is used to merge parts of one image onto another

In three dimensions the problem of masking becomes more apparent, and can no longer be handled with traditional masking. Occlusion between real and virtual objects is a way to create "3D masks", and are essential for the illusion of the virtual objects being a part of the real scene. The challenge is very simple to overcome, if a 3D model of the real scene is available. The depth buffer handles occlusions

between rasterized fragments in OpenGL, and may be used to handle the occlusions between real and virtual objects as well. By rendering the model of the real objects to the depth buffer only, the depth buffer will contain the depth information of the real scene, when the virtual objects are rendered to both depth and color buffer. This way occlusions are handled automatically by the graphics API. In figure 7.3 the lobby scene and its corresponding occlusion 3D model is seen.

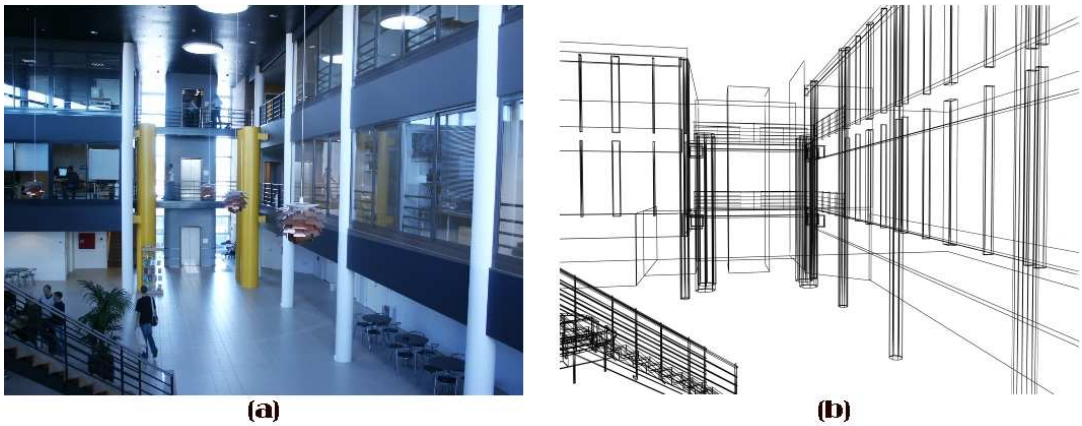


Figure 7.3: The lobby scene and its corresponding occlusion 3D model.

In the project occlusion handling has been implemented, and an example of it in action is seen in figure 7.4. A black sphere is set to rotate around the position of a brick. A 3D model has been made of the brick, and used as occlusion object. As seen in the figure, the occlusion makes sure that when the sphere is placed behind the brick, it will appear as if it is behind the brick.

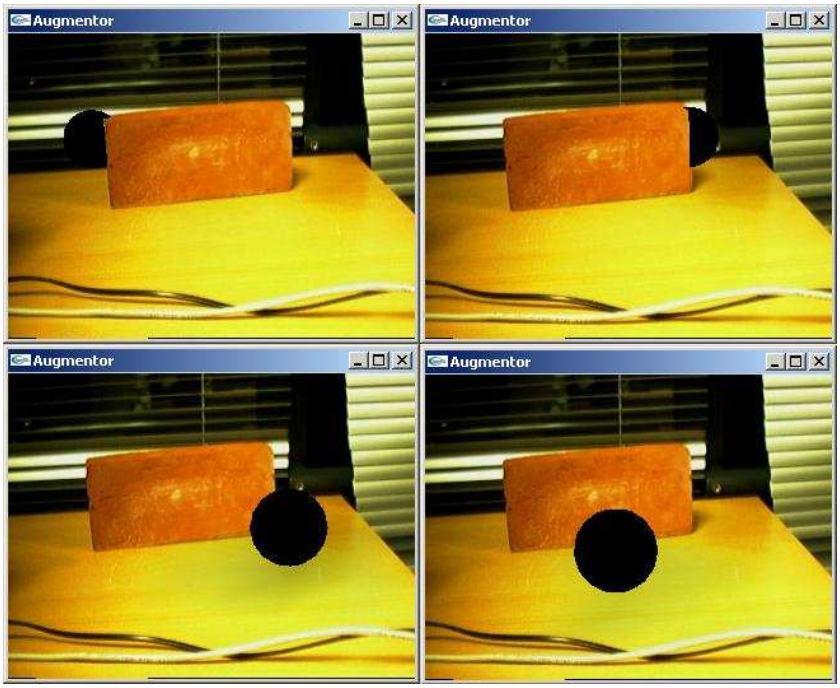


Figure 7.4: A virtual sphere is set to rotate around a real brick.

7.3 Anti-aliasing

Source: [Bergen County Academies, 2004] and [SigGraph, 2004]
Anti-aliasing is the process of blurring sharp edges in pictures in order to get rid of jagged edges. After an image is rendered, some applications automatically anti-alias images. Edges in the image are detected, and then blurred with the adjacent pixels to produce a smoother edge.

In order to anti-alias an image when rendering, the computer has to take samples smaller than a pixel in order evaluate exactly where to blur and where not to. For example, if one pixel is found to be placed on the edge of two objects, sub-pixel samples are made for that pixel, and checked how many are from each object, the relationship between the number of sub-pixels hitting each object is the mixture relationship between the two surfaces for the edge pixel. The resulting color values from the subsamples are averaged into a resulting blurred pixel, when viewed from a distance gives a smoother edge effect. To find which pixels need sub-sampling, the stencil buffer can find the silhouette of each object, which helps the process of detecting edge pixels.

Another similar method is supersampling, where more than 1 sample is performed for all pixels. These samples are at regularly spaced intervals. For example, if an image is computed at 2x by 2x or 4x by 4x points and displayed at 1x by 1x pixel resolution. Sampling at 2x by 2x for a 1x by 1x image increases the number of samples and graphics computations (and Z-buffer requirements for scan-line graphics) by a factor of 4. An alternative method is to sample at the corners and center of each pixel. This only increases the number of computations by a factor of two but requires increased overhead for bookkeeping (since the samples from the previous row must be stored).

Standard supersampling can be performed accelerated by utilization of the accumulation buffer.

Rather than combining pixels with unweighted average a weighted filter can be used.

An example of the effect of anti-aliasing is seen in figure 7.5.

Figure 7.5: An example of the effect of anti-aliasing

7.4 Motion Blur

Sources: [Elias, 2003a] and [Akenine-Möller and Haines, 2002]
Motion blur is an effect seen in images of scenes where objects are moving. It is mostly noticeable when the exposure of the camera is long, or if objects are moving rapidly.

A camera works by exposing a light sensitive image plane to a scene, for a very short period of time. The light from the scene causes a reaction on the image plane, and the result is a picture of the scene. If the scene is changing during the exposure, the result is a blurred image.

Motion blur is seen in almost anything moving caught with a camera. But it is seldom noticeable, but like many artifacts of images, its absence is noticeable. The presence of motion blur adds to the realism.

Additionally, an image with motion blur holds more information that one without. Compare the two images in figure 7.6:

(a) *(b)*

Figure 7.6: An example of how motion blur gives information about the movement in the scene.

The two frames shows an identical scene, but in one the camera is travelling forward quickly, in the other, the camera is moving to the left.

Motion Blur is very similar to anti-aliasing in images, where one method is to render an image in a much larger version, than the requested result, then reduce the size by averaging groups of pixels into one.

The method for creating smooth images is known as spatial anti-aliasing, and the method for creating smooth motion in animations is known as temporal anti-aliasing. Temporal anti-aliasing is analogous to anti-aliasing of images.

To achieve temporal anti-aliasing, the application must be able to create more images than needed presented for the user. In the Augmented reality project, this means creating more images than the camera can produce at the same time. If 8 times as many images as required for representation is generated for one frame, a four second animation would require 800 frames to be rendered, given a frame rate of 25.

The method is then to take the 8 intermediate frames that is used to create one frame, and mix them together evenly.

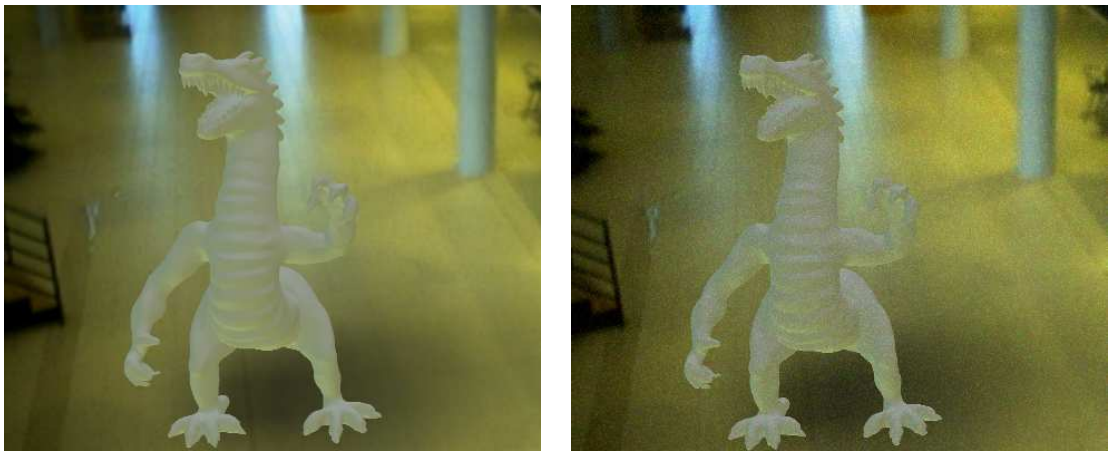
In an accelerated system this process can be performed using the accumulation buffer. Then the 8 frames could be added to the accumulation buffer each time a frame is rendered. This process is however counterproductive in real-time rendering, because it obviously lowers the frame rate. But the accumulation buffer can be utilised cleverly, and less costly.

If 8 frames of a model in motion has been generated and stored in the accumulation buffer, and then displayed. At the ninth frame the model is generated again and accumulated, but at the same time the first frame is rendered again and subtracted from the accumulation buffer. The buffer now has 8 frames of a blurred image, frames 2 to 9. The next frame, frame no. 2 is subtracted and no. 10 is added. This way only two renderings per frame is needed to continue to obtain motion blur.

7.5 Video Noise

Another artifact of the cameras inability to capture what it sees perfectly, is the graining in a picture taken of a scene. The graining is among other things due to the cameras compression, imperfection in the optics and CCD.

If a virtual object is superimposed on a real image or video feed, the things that reveals it to be virtual is when it stands out from its surroundings, like if it is perfectly rendered, while the background is noisy and grainy. Figure 7.7 shows an example with an object without and with noise added.



(a) *(b)*

Figure 7.7: An example of a virtual object in a real image (a) without and (b) with noise added to the image

The film grain is an important effect, that can help creating the illusion, that the camera is actually recording the object that appears on the screen, even though it is virtual.

There are two approaches to using the film grain on the augmented image. The noise is either added to the whole image or only the part of the image containing virtual objects.

It is desirable that there is a correlation between the noise affecting the virtual object and the noise affecting the real objects. If the noise is different on both parts of the image, the virtual object will still stand out in the context of the image.

If the noise is applied to the entire image, it can be assumed, that there is a relationship between the noise

on the virtual image and real image. If the virtual noise is stronger than the noise on the real image, the viewer will notice the virtual noise more than the real, and the virtual object will be appearing to have the same noise as the background. To create the full screen noise, noise images can either be added to the accumulation buffer per frame, or transparently textured to a full screen quad in front of the image.

The idea of creating virtual noise on the entire image has the tradeoff, that the entire image quality is degraded as a result. The advantage is that if the noise images are pre-rendered the full screen quad assures that the noise is not dependent on the camera resolution.

If the grain is only going to affect the virtual objects, the cameras noise must be simulated, if the noise on the virtual object is to fit into the noise on the rest of the image.

One way to obtain the noise of the camera, is to shoot series of images with the camera, and analyse how much each pixel varies over time. This information can be used to create a series of noise images for the camera. Another way would be to extract a transfer function of the camera, which can then be used to generate the noise.

One image of this series is the randomly chosen for each frame, and added to the accumulation buffer, to simulate noise on the image. But the noise must only affect the virtual objects of the image, so a mask telling which parts are virtual and which are not is needed. To get this, the stencil buffer can be utilised, which gives the silhouette of the object. This information is used to add grain only in the areas marked by the stencil buffer.

Another way to create the illusion, that the virtual object is part of the image, would be to recompress the image after the virtual part is added. The compression will affect both parts and the image may appear as if its coming from the camera.

CHAPTER 8

Modeling Real Scene Illumination

The purpose of this chapter is to evaluate methods for light estimation, and to infer usable methods for this project. This in terms means a review of existing state of the art.

A critical problem when creating an augmented reality system, is to locate the light sources, that illuminates the scene. In terms of blending real and virtual objects, correct estimation of the physical light sources is essential to how well the virtual object blends into the environment. The virtual light has to illuminate the objects correct, and with the right intensity, and the shadows projected on the scene from the light sources must also fall correctly, to maintain the illusion, that the virtual object is in fact part of the real scene. If the setup of the virtual light is inaccurate, it will be possible to detect inconsistencies between the real and the virtual parts of the image visually.

When modeling the illumination of a real scene, and applying it onto virtual objects, there are three steps that must be taken:

1. Measuring
2. Storing
3. Representation

To obtain information about the light sources, it can either be found manually, by measuring the position of the physical light sources, making a complete light model of the scene, or by applying an image based method to estimate the illumination of the scene.

To create a realistic light setup of the scene, it is necessary to gather the following information about each light source in the scene illumination:

1. Intensity/color
2. Shape/size
3. Position
4. Direction

There are several ways to estimate the illumination of a scene from an image. [Sato et al.,] developed a method to estimate the position and intensity of a scenes light sources by taking one photograph of a diffuse ball. By analysing the illumination of the ball and the shadows it cast, multiple light sources of a scene can be determined. For a project as this, a huge environment with multiple lights, estimating it with the mentioned method is nearly impossible, as the lighting changes depending where in the environment it is measured. At the same time the lighting of the scene changes dynamically over the course of a day, and as the [Sato et al.,] method is an offline method it is not usable to estimate lights in this scope.

The manually measured model is not very efficient, as the dynamic lighting changes are difficult to take into account. At the same time the calculations of such a model must be considered very CPU-intensive, and if it had to be dynamically updated, the application would be likely to have a very poor performance.

8.1 Image-Based Lighting

Image-based lighting (IBL) methods, are used to produce realistic lighting effects in rendered images, and to essentially replace the more traditional lighting setup. In this chapter a number of IBL estimation methods will be examined.

8.1.1 Environment Map

To represent the surrounding environment in IBL, environment maps are utilized.

There are several ways of storing environment maps. In computer graphics two representations are predominantly seen (Source: [LightWorks-User.com, 2004]):

Cubic environment maps: which are composed of six separate images that represent the six orthogonal directions in world space.

Spherical environment maps: which are composed of a single 'fish-eye-lens' (i.e., extremely wide angle lens) type image.

A Spherical environment map is a longitudinal/latitudinal map of the surroundings seen from one point in space. An example of a spherical environment map can be seen in figure 8.1.

Figure 8.1: An example of a spherical environment map.

8.1.2 High Dynamic Range Image

HDRI stands for High Dynamic Range Image. What is used in common picture file formats are LDRI, or Low Dynamic Range Images. These traditional image formats represent a limited range of intensity values for each pixel (most commonly 0 to 255 for each of the red, green and blue channels), sufficient for on-screen images. High Dynamic Range Image formats allow a greater range of intensity values, by using floating-point values for each pixel. This allows HDR images to be used to represent light intensities present in the real world. The technique, known as image-based lighting (IBL), is designed specifically to produce realistic lighting effects in rendered images, and to essentially replace the more traditional lighting setup. Source: [LightWorks-User.com, 2004].

8.1.3 Light probe

[Debevec, 1998] has developed a method for creation of an environment map, that use special probes, which are typically highly reflective spheres. The method is to take pictures at different exposures of the probe, from different angles, and extract a high dynamic range radiance angular map from the reflection on the probe. The environment is "unwrapped" from the photographed sphere, and an angular map, as seen in figure 8.2a, is created. The map can be used to create an irradiance map. From the assumption, that the environment is far away, the angular map is utilized to illuminate objects within the scene, as seen in figure 8.2b.

(a) An example of an angular map used by [Debevec, 1998] extracted from images of a light probe.

(b) Objects illuminated with [Debevec, 1998] method.

Figure 8.2: The [Debevec, 1998] method

[Ramamoorthi and Hanrahan,] expanded upon the [Debevec, 1998] method by utilizing Spherical Harmonics to create the Irradiance maps from the angular maps. Normally a computation of the irradiance of an angular image is a very heavy process, therefore [Ramamoorthi and Hanrahan,] developed a method to analyse the environment map, and create 9 spherical harmonics, which can be utilized to

generate the irradiance map at very fast rates, at least 500 times faster, than the traditional irradiance filtering.

8.1.4 Estimating point light sources from Environment Map

[Madsen et al., 2003] has developed a method for estimating the position and radiances for a small number of point light sources from the analysis of an environment map. The method addresses the problem of using IBL in real-time. The approach is to lower the number of light sources affecting the scene. That is, to approximate the complex omni-directional lighting environment with a small number of light sources, so the resulting virtual lighting resembles the real lighting of the scene.

The method analyses different exposures of the scene, and detects what areas are the most significant in radiance, and what their position and intensity/color is. An example of how objects are illuminated with this method can be seen in figure 8.3a.

(a) Spheres illuminated by the [Madsen et al., 2003] method.

(b) The [Madsen et al., 2003] method used to create shadows in the scene.

Figure 8.3: The [Madsen et al., 2003] method finds point light sources, and use these to shade objects and create their shadow.

The advantage of this method is that the lights of the environment can easily be estimated and represented real-time, and the estimated lights can be used to create shadows in the scene, as seen in figure 8.3b.

8.1.5 Rapid Shadow Generation in real-world lighting environments

[Gibson et al.,] created a new algorithm that uses consumer available hardware to render shadows cast by virtual objects in a real lighting environment. The method shows how it is possible to generate soft shadows from direct and indirect illumination sources and compositing them into a background image at interactive rates. The method efficiently encodes how sources of light affects each point in the image, by using a hierarchical shaft-based subdivision of line-space. This subdivision is utilized to determine which light sources are occluded by virtual objects. By using facilities available on consumer-level standard graphics hardware to display the shadows, the contributions from the occluded sources are removed from the background. The method is a trade-off between rendering accuracy and rendering

cost, and converges towards a result that is subjectively similar to what can be obtained from ray-tracing based differential rendering.

The scene is subdivided into multiple patches, and in a pre-process all intensity transfers between patches are calculated. If a virtual object intersects the shaft between source patches and receiving patches. The intensities of the receiving patches are reduced by the intensity that it gained from the source patch, that is occluded.

(a) *(b)*

Figure 8.4: A virtual chair (a) and bunny (b) is moving around real-time, while photo-realistic shadows are cast. [Gibson et al.,]

The shadow generation process proposed by [Gibson et al.,], is a coarse evaluation of where shadows are cast, and then a fine-level evaluation, that allows quick construction of the shaft hierarchy using a small amount of memory.

8.2 The Applied Method for the system

This chapter explores which solutions exists to shade virtual objects based on an image extracted from the surrounding environment. In table 8.1 the advantage/disadvantage of the examined methods is listed.

	Advantages	**Disadvantages**
[Debevec, 1998]	Photo-realistic	Does not find position of light-sources
[Madsen et al., 2003]	Fast Usable to create shadows	Models only the most significant lights
[Gibson et al.,]	Fast soft shadow generation Photo-realistic	

Table 8.1: Advantages and disadvantages for each proposed method

For the lobby-scene, which is the scene that will be augmented for this project, the lighting conditions

are changing temporally, as well as spatially. The spatial changes in the lighting means that the assumption from [Debevec, 1998], that the environment is far away and does not change if an object is moved, no longer applies. The correct position, size and intensity of the lights affecting the objects must be estimated, in order to create a proper effect, when objects moves around.

To this end, a priori knowledge of the environment is required. If an environment map was used in conjunction with a 3D model of the environment, the position could be found by mapping the environment map onto the 3D model, from the position it was recorded in the real scene, that is.

The [Madsen et al., 2003] method can be applied here, where it normally finds the angular position in regards to the point of environment map-recording, but not the exact distance to the light source, the combination with a 3D model, could shoot a ray in the direction found by the method. The first piece of geometry hit, is likely the position of the light source. This information can be used to create the shadows of the scene.

To estimate the position, size and intensity of possible light sources in the lobby-scene, the project will combine an environment map with a 3D geometric model of the lobby-scene. The environment map will be processed for possible areas, that provides light to the scene, while the rest will be thresholded away. This thresholded environment map will then be mapped onto the geometry. The geometry that has received a texture from the environment mapping, that is not filtered to black, are the possible light sources, where position, size and intensity is known for all of them.

8.3 Creating an environment map

In this project, Image Based Lighting (IBL) methods will be analysed, and an environment map will be created of the lobby-scene, augmented in this project. This environment map will be used to light the virtual objects, and cast shadows. To obtain this the environment map itself can either be used to shade the virtual objects or light sources can be estimated from the environment map.

The task of mapping a series of planar images to a sphere involves a warping, based on camera orientation, and a equirectangular mapping from the sphere to a planar latitude/longitude map. Normally, when creating environmental images, they would be manually placed. In this project tracking data for the camera orientation is available, hence we are able to automatically place and warp the images into the environment map.

8.3.1 Image warping

The process of projecting a series of planar images with a known field of view (FOV) onto a sphere is shown in Figure 8.5. It is assumed that the input images are rectilinear, which is a reasonable assumption, when images have been undistorted, as described in section 5.1 on page 39.

In Figure 8.5 the parameters used in placing the image are shown. ϕ, θ, and ψ (or azimuth, elevation, and roll) are Euler angles. f is the focal length, which is obtained during camera calibration. Based on the Euler angles a directional vector, $\vec{d}$, of length f is calculated. This will be the basis of the following projection.

From the tracking, described in chapter 6 on page 43, we have two parameters describing the orientation of the camera, a target vector, $\vec{t}$, parallel to the cameras optical axis and a vector, $\vec{u}$, in the up direction. From these parameters the 3D coordinates of pixels in the input images can be found by calculating a local coordinate system for the image, which is comprised by the two unit vectors $\vec{i}_u$ and $\vec{j}_v$.

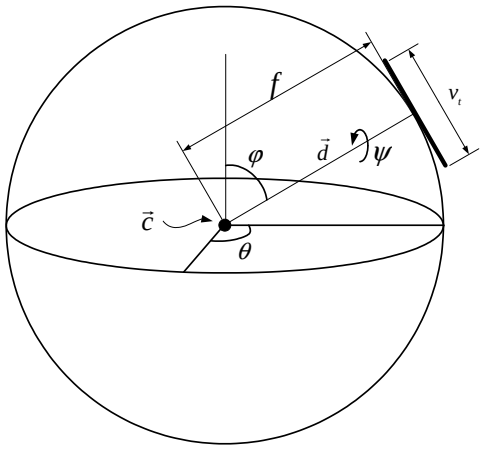


Figure 8.5: The process of creating an image mosaic. A series of planar images are mapped to a sphere.

$$\vec{d} = f\left(\frac{\vec{t}}{|\vec{t}|}\right) \tag{8.1}$$

$$\vec{i}_u = \frac{\vec{d} \times \vec{u}}{|\vec{d} \times \vec{u}|} \tag{8.2}$$

$$\vec{j}_v = \frac{\vec{u}}{|\vec{u}|} \tag{8.3}$$

The result of equations 8.1 and 8.2 are shown in Figure 8.6.

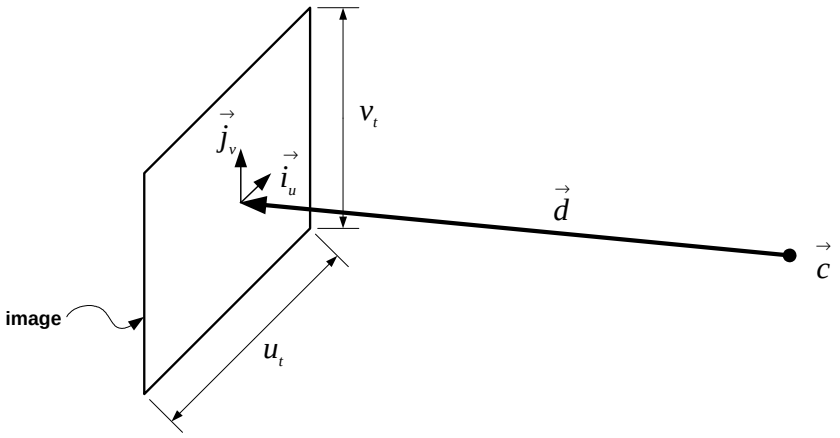


Figure 8.6: Pixel positions in three dimensions, taking into account roll, are found and mapped to a sphere.

When $\vec{i}_u$ and $\vec{j}_v$ are known, the angular position in the latitude/longitude environment map is calculated for each pixel in the input image, using equations 8.4 and 8.5.

$$\vec{d}_{new} = \vec{d} + \vec{i}_u \cdot p_u + \vec{j}_v \cdot p_v \tag{8.4}$$

$$(\theta, \phi) = \left(tan^{-1}\left(\frac{\vec{d}_{new.z}}{\vec{d}_{new.x}}\right) \quad , \quad cos^{-1}\left(\frac{\vec{d}_{new.y}}{\left|\vec{d}_{new}\right|}\right)\right) \tag{8.5}$$

The final step in creating the environment map is to assign the found angles to the longitude/latitude map.

8.3.2 Longitude/latitude conversion

Given an environment map with a dimension of p_u horizontal pixels, and p_v vertical pixels, the conversion from longitude, θ and latitude, ϕ, to any given pixel position is given as:

$$p(u, v) = (\frac{p_u}{2\pi} \cdot \theta \quad ; \quad \frac{p_v}{\pi} \cdot \phi) \tag{8.6}$$

An example of a spherical environment created with this method can be seen in Figure 8.7

Figure 8.7: An example of a spherical environment map.

8.4 Mapping environment to Geometry

Considering equation 8.6 reveals that in order to successfully map an environment map to the geometry-textures of a given scene, there are a some of factors that must be known. The position in the scene from where the environment map is recorded. How the azimuth orientation on the environment map in regards to the models orientation is. Both are factors that needs to be measured when recording an environment map of the scene, though they may be estimated by comparison of the model and the map afterwards.

There are two approaches to map an environment map to geometry. One approach is to run through each pixel on the environment map, and from the maps center of projection, the point in space the map was created from, rays are shot into space and the first geometry hit, is assigned that pixel. The second

approach is to create unique textures for each face in the scene, by finding the 3D position of each pixel in each texture on each face, and determining which longitude/latitude coordinate in the environment map corresponds to that position.

For this project, the latter solution has been pursued, because it assures all surfaces are assigned some sort of value, which can be used to extract the lights later, the calculation from 3D coordinate to a longitude/latitude coordinate is as well a useful one, that will be used later in the project.

First the conversion from spherical coordinates to 3D coordinates is utilised:

$$x = r \cdot cos\theta \cdot sin\phi$$

$$z = r \cdot sin\theta \cdot sin\phi$$

$$y = r \cdot cos\phi$$

Here the z and y coordinates are switched in regards to the conventional notation of spherical coordinates, which is a consequence of the fact, that in computer graphics, y and z are switched with respect to conventional math, to maintain x-y as the screen coordinates.

θ and ϕ are the only interesting parts of the spherical coordinates as they according equation 8.6 are the information needed to acquire the (u, v) coordinates in the environment map. If the 3D position of the environment maps center of projection is set as an offset vector $\vec{o}$, and the 3D position that is to be converted to longitude/latitude coordinates is $\vec{p}$, the direction $\vec{p}$ lies in, in regards to the offset $\vec{o}$ can be found by calculating:

$$\vec{p_0} = \vec{p} - \vec{o} \tag{8.7}$$

This operation resets the 3D point to be oriented in regards to the origin $(0, 0, 0)$. When this has been performed, r can be found by calculating the length of $\vec{p_0}$.

To find ϕ and θ the following applies:

$$\phi = cos^{-1}(\frac{\vec{p_{0.z}}}{|\vec{p_0}|})$$

$$\theta = tan^{-1}(\frac{\vec{p_{0.y}}}{\vec{p_{0.x}}})$$

When a vector is written with a suffix like $\vec{p_{0.x}}$, it means the x-component of the vector $\vec{p_0}$

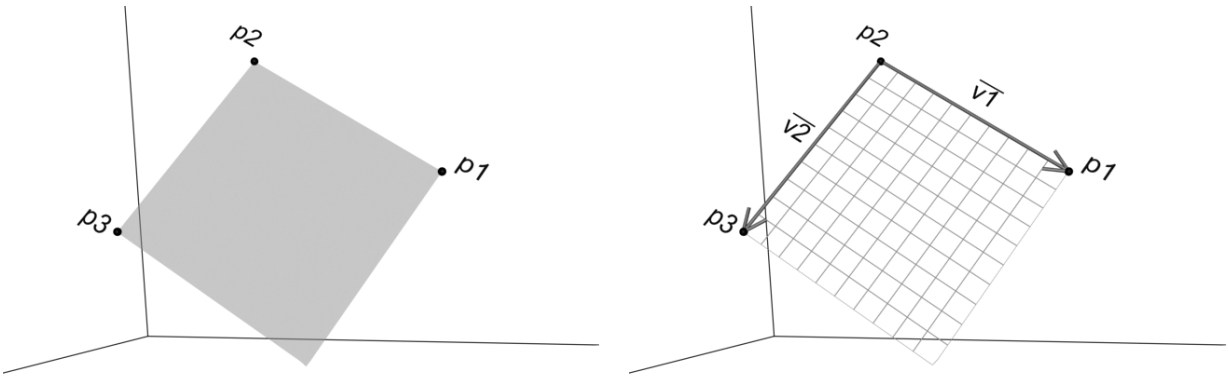
Combining these with equation 8.6, yields the calculations to get the longitude/latitude coordinates in an environment from an arbitrary point in the 3d space:

$$p(u, v) = (\frac{p_u}{2\pi} \cdot tan^{-1}(\frac{\vec{p_{0.y}}}{\vec{p_{0.x}}}) \quad ; \quad \frac{p_v}{\pi} \cdot cos^{-1}(\frac{\vec{p_{0.z}}}{|\vec{p_0}|})) \tag{8.8}$$

8.4.1 Coordinate transformation

The problem of finding a proper texture for an arbitrary plane existing within the scene is to remap between 3D coordinates and 2D coordinates. This may be solved by coordinate transformation. The principal of coordinate transformation is to have one coordinate system and map it onto another.
In this project the planes are represented by three points in 3D space $(\vec{p_1}, \vec{p_2}, \vec{p_3})$. An example of a plane may be seen in figure 8.8a.

The plane may be described with axes derived from the 3D points of the planes. Furthermore, the two axes may be described by the vectors spanned from point p2 on figure 8.8a where:



(a) A 3D plane represented by three coordinates.

(b) Vector $\vec{v_1}$ and $\vec{v_2}$ derived from points $\vec{p_1}$, $\vec{p_2}$ and $\vec{p_3}$.

Figure 8.8: The coordinate transformation for an arbitrary plane in space consisting of 3 points

$$\vec{v_1} = \vec{p_1} - \vec{p_2}$$
$$\vec{v_2} = \vec{p_3} - \vec{p_2}$$

Vector $\vec{v_1}$ and $\vec{v_2}$ can be seen in figure 8.8b.

The "horizontal" ($\vec{v_1}$) resolution is equivalent to the "vertical" ($\vec{v_2}$), and is denoted s. The half unit length of the vector is calculated in equation 8.10 and equation 8.10. The half unit length is used in order to address the center of the pixel.

$$\vec{v_x} = \frac{\vec{v_1}}{2 \cdot s} \tag{8.9}$$
$$\vec{v_y} = \frac{\vec{v_2}}{2 \cdot s} \tag{8.10}$$

The 3D position of the pixels within the face, which map the x and y position in the texture may be found using $\vec{v_x}$ and $\vec{v_y}$. If the vertical pixel position on the texture is denoted by x, and the horizontal is denoted by y, then the 3D position $\vec{p}$ of any pixel on a texture described by the three points may be found with:

$$\vec{p(u, v)} = (2 \cdot x - 1) \cdot \vec{v_x} + (2 \cdot y - 1) \cdot \vec{v_y} + \vec{p_2} \tag{8.11}$$

Combining this with equation 8.8 on the preceding page, can be used to create exact textures on any arbitrary face in a 3D scene.

Example:

In this example the resolution of the texture is 10×10, see figure 8.9 on the next page, therefore s is 10. $\vec{v_1}$ and $\vec{v_2}$ are scaled down to $\vec{v_x}$ and $\vec{v_y}$ as seen in the figure. To find the pixel at $(4, 7)$ the vector $\vec{p(4, 7)}$ is calculated using $(2 \cdot 4 - 1) \cdot \vec{v_x} + (2 \cdot 7 - 1) \cdot \vec{v_y} + \vec{p_2}$.

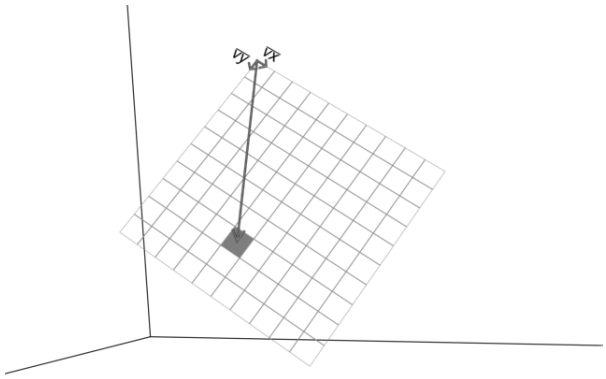


Figure 8.9: Vector $\vec{v_x}$, $\vec{v_y}$ and the vector to the point u=4 and v=7.

When dealing with a complex environment, such as the lobby-scene used in this project, occlusion of surfaces is a common problem. There are a number of solutions to get past this problem, where the simplest is to ignore occlusion, and map the environment to the surfaces that are in fact occluded from the environment map. Another solution is not to map anything to the occluded surfaces, and find their color by interpolating between the closest faces that received color. Furthermore the occlusions can be reduced by recording multiple environment maps from the same environment and combine these in the mapping process.

8.5 Updating an Environment Map

When updating an environment map from a dynamic lighted scene, several solutions has been considered.

One way is to record a set of environment maps from the scene, at different times of day, with different lighting, dependent on the weather. This set of environment maps are templates of different lighting conditions. To utilise these templates the system would have to analyse the images it is acquiring and determine which of the templates matches the lighting best, and switch to that environment map in the scene. When the scene lighting has changed enough to be visually different from the current employed environment map, set by a certain selection criteria value, the system chooses a new environment map that fits the best, and gradually changes the derived virtual lighting in the scene to this new setting.

Another way would be to utilise the [Debevec, 1998] method, by having a metal sphere in the scene, and a secondary camera that records it. The information from the probe is used to extract light information, which is used to update the virtual lighting of the scene.

Furthermore it is possible to create one environment map of the scene, recorded from the center of the scene, as the only one used in the system. During execution of the system, the camera is in placed another position than when the environment map was recorded. Therefore it will be necessary to calculate which parts of the environment map recorded from the center fits the different part of the images visible from the camera viewpoint. If rays are shot out through the pixels into the room from the cameras position, certain 3D points are obtained in the places, where the rays intersect scene geometry. if these 3D points are used in conjunction with equation 8.8 on page 63 it is possible to find out which pixel values in the initial environment map they represent. The pixels from the shoot rays are compared to

their correspondent in the environment map. When the difference between the pixels obtained from the camera, and the pixels in the environment map is above a certain set threshold, the mean difference is calculated, and the environment is updated with this difference, and it will thereby look as much as the environment, as possible by changing the intensity.

The lobby-scene is a very dynamic environment, lighting-wise, and creating a precise real-time dynamic update of an environment map, is an entire project on its own. For this project the problem of handling the dynamic environment changes will be detecting major changes in the environment, and adapt to them using the latter described method.

The implementation will evaluate the lighting of the scene, by taking chunks of the images from the camera, and calculate which areas in the environment map used for the illumination is the equivalent to the chunks obtained from the camera. These will be compared, and the overall intensity difference between the two areas is calculated. This process will be performed at different angles of the camera, and when a certain percentage of the map has been covered, the average intensity difference of these areas is calculated. This calculated intensity difference is used to adjust the brightness of the virtual objects that are rendered to the screen. In the system this brightness adjustment will be performed by changing the albedo value of the objects.

CHAPTER 9

Illuminating a scene

This chapter extends on the uses of light estimation, described in the previous chapter. This chapter analyses different algorithms for shading and shadowing objects.

When the right position, shape and intensity of the lights affecting the real scene has been estimated, the next step is to use this information to shade and shadow the virtual objects correct, so they do not seem out of place, at least not because of the lighting.

Light is electromagnetic energy, and is the part of the spectrum, that is visible. A photon is the basic part of light, and is a discrete quantum of light energy. A mediums index of refraction sets which velocity photons can be transmitted from it, the mediums index of refraction varies with the wavelength of the photons. The mediums surface properties dictates a photons behavior, when intersecting it, the medium can either reflect, refract or absorb the photon. Most mediums reflect photons at certain wavelengths, and absorbs at others, this is what causes surfaces to have a certain color.

To simulate light hitting a surface satisfactory, the effect of light and its interaction with materials must be processed. In computer graphics this simulation is however simplified to shading models, that is not completely physical correct, but yields a visually realistic result. This chapter deals with such models, in order to choose which is to be used to shade the virtual objects of the project.

Another important factor in creating realistic looking graphics are the shadows the objects are casting. In real time computer graphics, complex realistic shadows are difficult to create convincingly, but numerous attempts has been made, where some of them is described in this chapter.

One of the early attempts at creating shadows included a patch of darkened texture, normally round in shape, that projected onto the floor below the shadow-casting objects. This approach is called "shadow blob" but is seldom used anymore as they are not very aesthetically pleasing to the realism of a given scene.

Normally a light source may be simplified to being a single point in space, from where light emits. But when mixing real imagery with computer generated imagery this simplification poses problems. A point light source will always give hard shadows, meaning the border between what is in shadow (umbra) and what is not in shadow, will be hard. While these hard shadows may look good in some applications they are not realistic looking, and when mixed with real imagery these shadows will look fake in comparison. The traditional computer generated shadows created from point light sources lacks a penumbra area, an area that marks a soft transition between the umbra area and the fully lit area. Many algorithm exists to do this, but the real time application of soft shadows has been limited for many years, as it is a heavy process.

There are 4 different main groups, in which to classify light sources within the field of computer graphics. Directional, positional, spots and area light. The three first, are simplified versions of real life lighting, while Area light is the class all physical light sources belongs to. For this project, the area light is estimated, as seen in the previous chapter, and therefore lighting methods utilizing area lights are the only ones considered in the following chapter.

Area lights emit light energy in all directions from their surface, which can take any shape or size.

Simulating area light is slightly more complicated, than the other groups of light mentioned. This is due to the fact that any physical object has an infinite number of points on its surface theoretically, from where photons are emitted. So by simulating area lights, is equivalent to simulating an infinite number of point lights placed within the bounds of an object. The common approach to this problem is to place a finite number of light sources evenly across the surface of the lighting object.

The following sections evaluates various illumination techniques and takes a look at which solutions yields realistic visualisation of the virtual objects.

Lighting and shadow casting algorithms can be very roughly divided into two categories; Direct Illumination and Global Illumination.

9.1 Direct Illumination

Direct Illumination is a term that covers the classical lighting methods. A scene consists of two types of entity: Objects and Lights, the lights cast light onto the objects, unless there is another object in the way, in which case a shadow is cast.

There are various techniques in this category: Shadow Volumes, shadow mapping, Ray Tracing. But they all suffer from similar problems, see table 9.1.

Method	Advantages	Disadvantages
Ray Tracing	- Can render both mathematically described objects and polygons - Allows rendering of advanced volumetric effects	- Slow - Very sharp shadows and reflections
Shadow volumes	- Can be modified to soft shadows	- Complex implementation - Very sharp shadows - Polygons only
Shadow mapping	- Light implementation - Fast	- Sharp shadows - Aliasing problems

Table 9.1: Problems and advantages for direct illumination (Source: [Elias, 2003b])

Direct illumination is not a realistic way to mimic the characteristics of real-world lighting. These methods can produce realistic images, but they can only do this when given a scene with point light sources, and perfectly shiny or perfectly diffuse objects. In the real world, it is possible to see objects that are not directly lit by light sources, shadows are never completely black. Direct illumination handles this relation by adding an ambient light term. Thereby all objects in a given scene receives a minimum amount of uni-directional light. In the lobby-scene project all surfaces are already naturally lit, so if these direct illumination algorithms can be extended to create soft shadows, they may be enough to fool the viewer into thinking of the as realistic shadows.

9.1.1 Shadow mapping

Source: [Kilgard, 1997]
The basic idea in Shadow mapping is to put a camera in the position of the light source, and let it record depth information from that point of view. This depth information is then compared to the depth

information from the users viewpoint, and it is possible to ascertain which areas of the scene that is in shade, and which areas are not.

The algorithm is described with simple pseudocode:

Pseudo code 9.1 The creation of shadows using depth comparison.

1. Render scene from the light-sources point of view.
2. Store depth values of the closest fragments in a texture.
3. Project depth texture, which is the shadow map, onto scene.
4. Render scene from cameras point of view.
5. At each fragment, compare the projected depth texture with the current depth buffer.
6. If two compared depth values are the same the area is not shadowed.

This method has a number of trade-offs:

Finite resolution

The detail level of the shadows are dependent of the resolution the shadow map has been recorded with the camera. The lower the resolution the shadow map was recorded with the more pixelated the shadow will be. To utilise shadow maps one would be forced to make a trade-off between performance and shadow details.

Aliasing

As the recorded depth map is a discretely sampled map from both the light and view-port, there may be areas where artifacts such as an object self-shadowing in places that should have no shadow. This problem however is solvable if shadow mapping were to be utilised.

9.1.2 Shadow Volumes

Source: [Akenine-Möller and Haines, 2002]
Shadow volumes is one of the most commonly used real-time shadow casting algorithms today.

The idea behind shadow volumes is to cast shadows onto arbitrary objects by clever use of the stencil buffer. From a given point in space, which is the point light source, a volume is created by finding the silhouette of the object that is to cast a shadow, and extruding a 3D volume from the light source through the silhouette points, towards infinity. The part of the volume that is on the other side of the light occluding object than the light, is a volume of the shadow, the object will cast. The volumes themselves are purely theoretical, and thus invisible to the camera in the scene.

The implementation of the Shadow Volume method is to follow a ray through each pixel until the ray hits visible objects on the screen. While the ray is on its way to this object, a counter is incremented each time it crosses a front facing face of a shadow volume, meaning the counter is incremented each time the ray goes into a shadow. Each time the ray crosses a back facing face of the volume, the counter is in the same way decremented, to signify that the ray is outside a given shadow. When the ray hits the object to be displayed, the counter is checked. If the number is greater than zero, the pixel is in shadow, and otherwise it is not.

Shadow Volumes creates very sharp shadows, but the algorithm can be tweaked to create soft shadows, which has been done by [Vignaud, 2001] and [Andersson and Akenine-Möller, 2003].

9.1.3 Shaking the light makes soft shadows

[Vignaud, 2001] developed a method to use ordinary direct illumination hard shadows, and extended them to be soft shadows. The technique is simple, for each frame, the light casting the shadows is "Shaken". The basic idea is to display the shadows while shaking the light, and to blur them together. Normal hardware is not fast enough to display that many volumes at the same time, so time coherency is used to fake it. During one frame,the current volume is displayed, and blurred with the volumes of the previous frames. To shake the light, it is positioned at different positions around the center for each render pass per frame.

9.2 Global Illumination

In global illumination methods the problems associated with local illumination is addressed. Classic ray-tracers simulates light reflecting only once off each diffuse surface, with global illumination the many reflections of light bounces around a scene are simulated, to create a more realistic virtual light.

Each object in a ray traced scene must be lit by a light source for it to be visible, in a globally illuminated scene an object may be lit simply by it's surroundings.

When light hits a surface, some of the light is absorbed, and some is reflected. The surface properties of the object determines how much light is reflected. Surfaces that has been illuminated acts as light sources themselves, when they reflect light, light sources with reduced intensity. Classic ray-tracing can not be used to simulate global illumination, because ray-tracing only handles point light sources. So to perform global illumination with classic ray-tracing each emitting surface must be subdivided into an infinite amount of point lights. This means that diffuse surfaces becomes an infinite number of point lights when illuminated, and illuminating other surfaces from those point lights becomes difficult.

9.2.1 Bidirectional Reflectance Distribution Function

Source: [Akenine-Möller and Haines, 2002]
A Bidirectional Reflectance Distribution Function (BRDF) is a function that describes how light is reflected from a surface, it is used to describe material properties. Input to the function is incoming and outgoing azimuth and elevation angles, as measured on the surface. The wavelength of the incoming light is another input, which is equivalent to the color of the light.

The BRDF returns a unit-less value that signifies the amount of energy reflected in the outgoing direction, based on the incoming direction.

The BRDF is a relation between incoming and outgoing radiance, but does not explain how materials physically interacts with light.

An important property of the BRDF is the *Helmholtz reciprocity*, which states that the input and output angles can be switched, and the function value will remain the same.

The BRDF is an abstraction that describes how light interacts with a surface, but it is also an approximation of a more general equation, the Bidirectional Surface Scattering Reflectance Distribution Function (BSSRDF). The BRDF does not include the scattering of light within the surface, which can be seen

in e. g. marble. BSSRDF includes these properties, but only handles reflections of light, not transmission of light through the surface. This can be handled with two BRDF's and two BTDFs (T for Transmittance), by defining one of each to the two sides of the material, which makes the BSDF (S for Scattering).

With a BRDF and an incoming radiance distribution, the reflectance equation gives the outgoing radiance for a given viewing direction, relative to the surface. This is done by integrating the incoming radiance from all directions on the hemisphere on the surface:

$$L(\theta_o, \phi_o) = \int \int_{\Omega} f(\theta_o, \phi_o, \theta_i, \phi_i) L(\theta_i, \phi_i) cos(\theta_i) d\sigma(\theta_i, \phi_i) \tag{9.1}$$

Subscript i and o are the incoming and outgoing (view) directions. The L function is the radiance traveling in a given direction, and f is the BRDF. The integrals means to integrate over the local hemisphere Ω for the surface. The utilization of the equation is for all directions on the surface to determine the incoming radiance multiply it with the BRDF for the incoming and outgoing directions, scale it by the incoming angle to the surface, and integrate. The result of this operation is what radiance is seen for a given viewing direction.

The Lambertian BRDF is the ideal diffuse, which can be used to model the appearance of rough surfaces, with the characteristic that they reflect light equally in all directions, see figure 9.1.

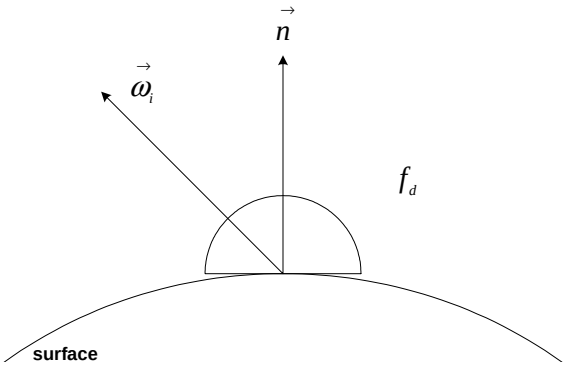


Figure 9.1: The Lambertian BRDF. Surfaces with this BRDF scatters light equally in all directions.

The Lambertian BRDF is given by:

$$f_d(\vec{\omega_i}, \vec{\omega_o}) = \frac{\rho_d}{\pi} \tag{9.2}$$

This equation is independent of both $\vec{\omega_i}$ and $\vec{\omega_o}$, the constant ρ_d is the reflectance, or albedo, and ranges from 0 to 1.

The counterpart to the diffuse BRDF is perfect specular BRDF. The ideal specular BRDF models perfectly smooth surfaces. This surface reflects all light in exactly the exact mirror direction, as in figure 9.2.

The specular BRDF is given by:

$$f_s(\vec{\omega_i}, \vec{\omega_o}) = \begin{cases} \rho_s & \vec{\omega_r} = \vec{\omega_o} \\ 0 & \vec{\omega_r} \neq \vec{\omega_o} \end{cases} \tag{9.3}$$

where ρ_s is the specular reflection (in the range 0 to 1).

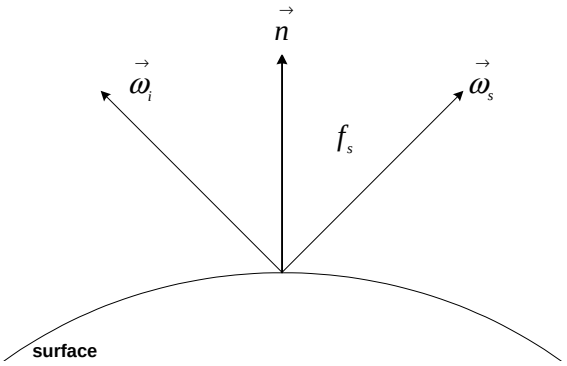


Figure 9.2: The perfect specular BRDF. Surfaces with this BRDF scatters light only in the mirror direction

In appendix B on page 129 Lambertian shading is described in detail.

9.2.2 Global Illumination Algorithms

A number of algorithms that simulates global illumination exists, but are all very computationally demanding. But during the past years global illumination algorithms has become more common in applications as the processing power available on consumer-level hardware has grown.

The big trade-off of global illumination is the speed. The multiple Global illumination calculations of a given scene makes the methods slow.

To exemplify the difference between global and local illumination the two approaches has been utilised to lit the scene seen in figure 9.3.

The scene is lit from outside, so the room would look as if it was lit by the sun shining in through the window. In the local illumination version, a spotlight is set to shine in. The parts that are not lit by the light can be lit by the ambient light, which will give the rest of the room a uniform look. To bring out the details the room is lit with an additional point light source in the middle of the room in figure 9.3a. The room is rendered using a ray-tracer.

In figure 9.3b global lighting is added. The source of light is a rendered image placed outside the window. The rendering is performed with a radiosity renderer.

The differences from the ray-traced image is apparent:

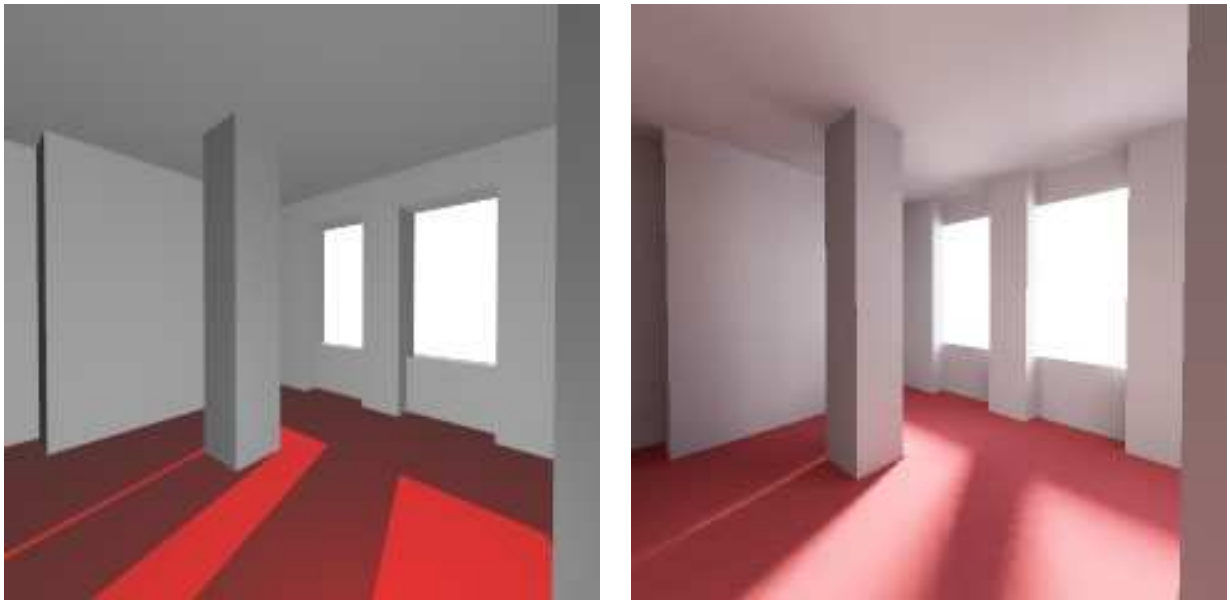
Light bounces on surfaces, lights the entire room.

Soft shadows.

Color bleed.

9.2.3 Radiosity

Source: [Elias, 2003b]
Radiosity is a technique that calculates the lighting in a complex diffuse-lighting environment, based on the scene's geometry. Because the radiosity calculations do not include the eye point of the viewer, the geometry and lighting in the environment do not need to be recalculated if the eye point changes. This



(a) A scene lit with local illumination.

(b) Same scene lit with global illumination.

Figure 9.3: Comparison of local and global illumination. [Elias, 2003b]

enables the production of many scenes that are part of the same environment, and a "walk through" the environment in real time.

The amount of light in the scene is constant - light which is emitted is always absorbed somewhere. Every object is subdivided into many, many patches (or polygons). Each patch has a specification telling the system how this patch will affect other patches, that is, how it reflects light.

The calculation of radiosity is based on static objects.

9.2.4 Irradiance Volume

Source: [Greger et al., 1995]
Irradiance Volumes is an approximation to global illumination in a scene. Irradiance volume is a volumetric representation for the global illumination within a space based on the radiometric quantity irradiance. The irradiance volume provides a realistic approximation without the amount of calculations needed with traditional global illumination algorithms. The method, that analyses the surrounding scene, samples it and approximates how the lighting conditions are in various key points in the scene. When a volume of such samples has been created, a random object can be shaded with a lookup in the volume.

The volume tells how the global illumination of the scene is affecting dynamic objects.

9.2.5 Monte Carlo Rendering

Source: [Shirley, 1998]
Monte Carlo Ray Tracing, or sometimes mentioned as path tracing, is a Monte Carlo Technique. For

a particular light source configuration, the reflected light is computed utilizing Monte Carlo integration. In Monte Carlo Integration random points are picked over some simple domain D', which is representing a complicated domain D.

Classic path Tracing performs performs this integration over the entire hemisphere for a surface, in order to take direct and indirect illumination into account. Monte Carlo Ray Tracing uses Monte Carlo Integration to simplify the hemisphere integration.

The advantages/disadvantages for various global illumination methods may be seen in table 9.2.

Method	**Advantages**	**Disadvantages**
Radiosity	Realistic light for diffuse surface Simple concept, easy implementation Can be accelerated with 3D hardware	Slow Bad point source handling Bad handling of shiny surfaces
Monte Carlo	Very good results Can simulate all optical phenomena	Slow Slightly difficult Tricky to optimise
Irradiance Volume	Good results Fast	Does not handle specular reflection

Table 9.2: Problems and advantages for global illumination methods.

9.3 Illumination algorithms used for the system

In this project the various shading methods will be used to shade a virtual object set in a real scene. To accomplish the task of merging the real and the virtual parts of the image, the real image must affect the virtual object, and the virtual object must affect the real scene.

The virtual object must be shaded according to it's surroundings in the image. The virtual object and real objects must also affect each other with their shadows, for the illusion to be convincing.

In this project, the lighting of the surroundings is not calculated, but measured with an environment map.

Radiosity and Monte Carlo rendering are two ways to calculate the light bounces of a scene. In the lobby scene the lighting is already handled naturally, so these methods can be used to calculate the how the shadows from the various light sources, measured from the real scene, acts in the scene.

Both Radiosity and Monte Carlo rendering are traditionally offline methods, that calculates the lighting of the scene in advance. This means they normally only handles static scenes with static objects. If an object is dynamic the updating of the calculated environment must be dynamic as well. [Gibson et al.,] created a method to use pre-calculated trajectories for the lights paths to real-time show soft shadows generated from the environment. The various sources show that radiosity is a faster algorithm than Monte Carlo that still yields convincing shadows. A very important demand to the resulting system of this project is that it runs real-time. Therefore the project will make use of the idea behind radiosity to create soft shadows in the environment, based on dynamic objects within the scene.

The radiosity can not shade dynamic objects unless it is possible to make advanced radiosity calculations for each frame. Therefore another algorithm is needed to shade the dynamic objects of the scene.

Irradiance Volumes was originally created as a supplement to global illumination algorithms. The Irradiance Volume method calculates the lighting of an entire virtual scene in advance. The irradiance volume method then analyses the scene, and shades objects accordingly. This is a very fast method to shade objects based on their surroundings.

In this project the lighting of the real scene is measured, by creating an environment map of the surroundings. If the irradiance volume is tweaked to sample the radiance from such an environment map, it can be used to shade the virtual objects in the scene, so they match up with the lighting of the surroundings.

CHAPTER 10

Irradiance Volume

This chapter explores the Global Illumination approximation algorithm called Irradiance Volume. Irradiance Volumes is an approximation to global illumination in a scene. Irradiance volume is a volumetric representation for the global illumination within a space based on the radiometric quantity irradiance. The irradiance volume provides a realistic approximation without the amount of calculations needed with traditional global illumination algorithms. The method, that analyses the surrounding scene, samples it and approximates how the lighting conditions are in various key points in the scene. When a volume of such samples has been created, a random object can be shaded with a lookup in the volume. The basic assumption of the irradiance volume is that the objects the irradiance volume shades, do not themselves affect the lighting of the scene.

Source: [Greger et al., 1995]
A realistic image synthesis has always been a major goal in the field of computer graphics. One such realistic synthesis is the concept of global illumination, as opposed to local illumination. Traditionally global illumination accounts for light interaction between surfaces in an environment, so both direct and indirect light sources are taken into account. These global illumination algorithms provide realistic effects but often at the price of computational expense.

A method to create the effect of global illumination exists, that makes a reasonable approximation with high performance, which is called Irradiance Volume. This is used in many applications where the visual appearance is more important than the accuracy of the generated illumination.

The visual appearance of a scene with diffuse surfaces can be computed from the reflectance and irradiance at the surfaces.

In irradiance volume the concept of irradiance from surfaces is extended into enclosure. The irradiance volume represents a volumetric approximation of the irradiance function. The volume is built in a preprocess step, and can be used by an application to approximate the irradiance at a given location within the environment quickly.

10.1 Radiance and Irradiance

Radiance is the density of light energy flowing through a given point in a given direction. The luminance is the point viewed from a certain direction. The radiance in all directions at a given point $\mathbf{x}$ makes a directional function called the radiance distribution function (RDF). There is a RDF at every point in space, which is why radiance is a five-dimensional quantity defined over all points and directions, two directional and three spatial.

The reflective properties and the incident radiance of a surface determines its outgoing radiance. The incident radiance of the surface is defined by hemisphere of incoming directions called the field-radiance function (FRF).

If a surface is diffuse then its radiance is defined by the terms of the surface irradiance, H:

$$H = \int L_f(\omega) cos\theta \quad d\omega \tag{10.1}$$

L_f is the field radiance incident from direction ω, and θ is the angle between ω and the normal vector $\tilde{\mathbf{n}}$ of the surface.

If the radiance from all incoming directions to one point is sampled, it can be computed into the irradiance for all possible orientations, these irradiances can be plotted as a radial function. This is the irradiance distribution function (IDF) for a given point. This IDF can be computed for every 3D point in a given scene. A complete IDF is not practically possible, as it would demand sampling in a very high amount of directions and points, and the computational requirements are equally high. For high efficiency the IDF is approximated.

If the IDF is approximated within the space enclosed by an environment in a form that can be evaluated quickly, the costly explicit evaluation of the IDF, for each time an object moves, can be avoided. This approximation is the irradiance volume, and the assumption is that it is possible to approximate the IDF with enough accuracy to avoid visual artifacts.

The simple way to approximate the function is to evaluate a finite set of points, and use interpolation to determine the function between them.

10.2 Creation of an Irradiance Volume

Irradiance volumes must be used in conjunction with an existing lighting solution, as it samples the radiance in the lit environment. Radiosity could be one of such solutions. To built the irradiance volume the possibility to sample radiance at all points in all directions must exist. For a geometric environment built of polygons the sampling can be achieved with ray-casting techniques.

The traditional implementation of the irradiance volume, involves a grid of samples being spanned in the environment in which objects are to move. The sampling is performed uniformly in the directional dimensions, since the IDF in a point is always continuous. The spatial dimensions are a bilevel grid for adaptive sampling, the reason for this, is that speed is the primary concern rather than accuracy.

The pseudo code for the computation of the grid is:

Pseudo code 10.1 The creation of an irradiance volume grid

1. Subdivide the bounding box of the scene into a regular grid.
2. For any grid cell that contains geometry, subdivide into a finer grid.
3. At each vertex in the grid, calculate the irradiance distribution function.

The subdivision of cells containing geometry, is to alleviate the most likely source of errors in the irradiance interpolation, since most discontinuities are caused by interruptions of light flow by surfaces. The bilevel grid allows low density sampling in open areas, while a higher sampling is possible in regions with geometry.

The first-level grid will be a $3 \times 3 \times 3$ grid that is built within the environment. Any grid cell containing geometry is then subdivided into a second-level $3 \times 3 \times 3$ cell. The regular grid outer vertices must be placed slightly within the boundaries of the environment so they do not intersect pieces of geometry. An illustration of a grid/cell/sample structure is seen in figure 10.1.

To approximate the IDF at an arbitrary point in a scene, the following procedure is utilised:

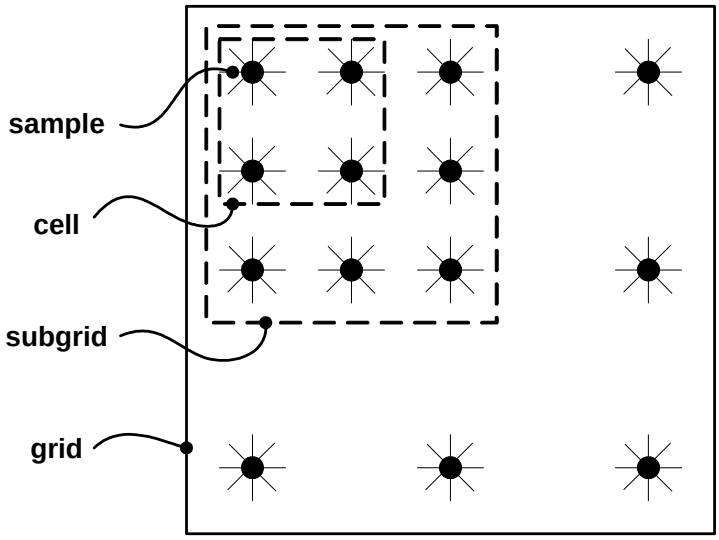


Figure 10.1: A grid is the overall structure of the storage of the irradiance volume. A grid contains cells. Each cell can either contain samples at its corners, or a new sub-grid-level, which contains sub-cells.

Pseudo code 10.2 The creation of an irradiance volume sample

1. Given a directional sampling resolution, choose a set of directions over the sphere of directions in which to sample the radiance.
2. In each chosen direction, determine and store radiance.
3. Use the stored radiance to compute the irradiance distribution function.

This is performed for each grid vertex, to form the irradiance volume.

To gather the radiance viewed from a point in space, the radiance is point sampled in a set of directions over the spherical set of directions $S^2(\theta, \phi)$. Each sample defines a region of constant radiance over S^2.

To represent the sphere of directions S^2, [Shirley and Chiu, 1994] provides a method to map a unit square to a hemisphere, and preserving the relative area of regions on the square. With this procedure, all directions represent an equal area on the hemisphere, which gives a mathematical advantage in the later calculations. This representation is better than the traditional longitude-latitude mapping, because mapped areas have better aspect ratio. The actual implementation will include two such hemispheres for each sample, to create one complete sphere. An illustration of the hemispheres that are stored as a square map is seen in figure 10.2.

The radiance $L(\mathbf{x}, \omega)$ is computed by casting a ray from point $\mathbf{x}$ in direction ω and evaluate the first surface, that is hit. If the surface is diffuse, the radiance is gathered from the surface, if not, the ray is reflected until a diffuse surface is found. A radiance sample can be seen in figure 10.3 on the following pagea.

The number of samples on the hemisphere, that are needed to create a good approximation to the IDF at a point depends on the surroundings. A scene containing many objects may require a high sampling rate to avoid that certain important features of the scene is not missed.

When all radiance values has been gathered at a point, they will be used to calculate the irradiance for each sampling direction on the sphere. The Irradiance and the radiance is calculated for the same resolution and the same directions. Irradiance is a smoother function than radiance, which means that a high sampling rate is not always needed, depending on the surroundings. In figure 10.3b the radiance

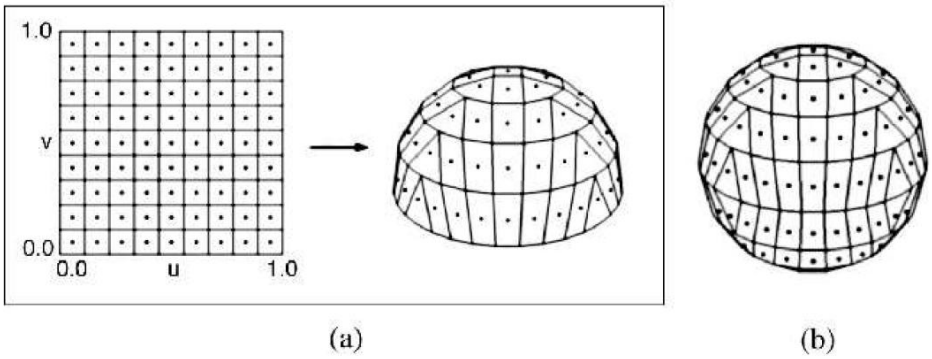
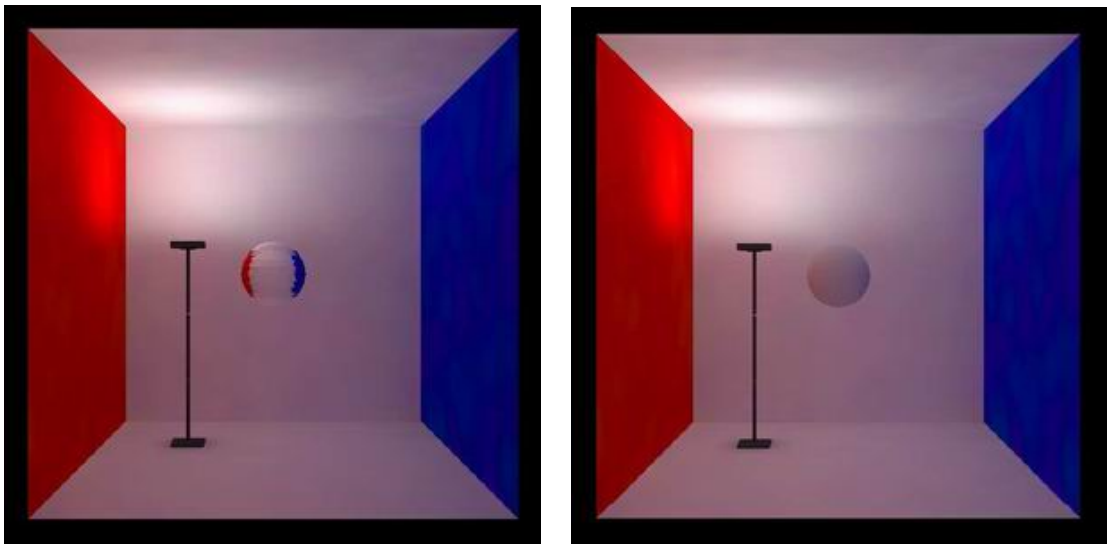


Figure 10.2: (a) shows the hemispherical mapping of a square map. (b) shows a sphere created from two of such hemispheres. (Image by [Shirley and Chiu, 1994])



(a) The radiance of the Cornell Box sampled from the center of the box

(b) The radiance sample converted to an irradiance sample

Figure 10.3: A sample in the middle of a geometric model as radiance and irradiance respectively. (Image by [Greger et al., 1995])

sample from figure 10.3a is seen converted to an irradiance sample.

For a given direction the irradiance is computed for a hypothetical surface whose normal vector points in that direction. With a set of directional samples $\{\omega_1, \ldots, \omega_n\}$ sampled at a given point $\mathbf{x}$ the irradiance in direction ω is calculated using equation 10.2:

$$H(\omega) \approx \sum_{i=1}^{N} L(\omega_i) max(0, \omega_i \cdot \omega) \Delta\omega_i \tag{10.2}$$

where L is the radiance in direction ω_i from point $\mathbf{x}$, $\Delta\omega_i$ is the solid angle of the cone each ω_i represents, that is, each direction per sample represents a surface area on the sphere of directions, this area becomes a cone when projected from the center. N is the number of directional samples, on the sphere.

The sphere of directions is subdivided into regions of equal size, it follows that $\Delta\omega_i \approx \frac{4\pi}{N}$ and equation 10.3 emerges:

$$H(\omega) \approx \frac{4\pi}{N} \sum_{i=1}^{N} L(\omega_i) max(0, \omega_i \cdot \omega) \tag{10.3}$$

Equation 10.3 is calculated for a number of directions at the sample point in order to obtain a set of directional irradiance samples. The irradiance is computed in each direction by summing the cosine-weighted contribution from each radiance sample on the hemisphere oriented in direction ω. The approximate irradiance form an approximation of the IDF at a given point in space.

The resulting IDF can like a radiance sphere be displayed as an irradiance sphere.

In an implementation the Irradiance Volume is represented as three distinct data structures: samples, cells and grids.

- **Samples:** Contains directional irradiance values for a particular point.
- **Cells:** Representing a box in space, bounded by eight samples, one at each corner. Can contain a grid to form a bilevel structure.
- **Grids:** 3D array of cells.

When the irradiance volume is queried for the IDF at a given point between the samples, the following steps are taken:

Pseudo code 10.3 Querying the Irradiance Volume

1. Calculate which cell in the first-level grid the point belongs to.
2. If cell contains a grid, calculate which cell in the second-level grid in which the point is contained.
3. Find which data value in the cell's samples corresponds to ω.
4. Given the points position within the cell and the eight corner sample values, interpolate to get the irradiance.

A completed irradiance volume can be seen in figure 10.4 on the next page.

The process of finding which cell that contains which point is simple, as the bounding box of the volume is known. To find the value corresponding a direction ω, the hemispherical mapping described by [Shirley and Chiu, 1994] must be inverted. This will convert the direction to a (u, v) coordinate in

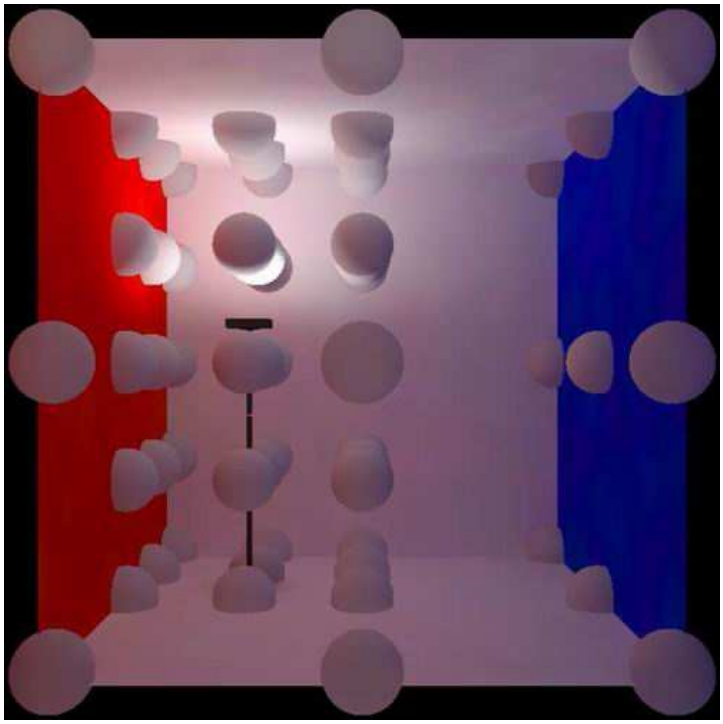


Figure 10.4: A completed Irradiance Volume. (Image by [Greger et al., 1995])

the rectangular array describing the sample. This is used in the query where a simple check is performed on ω to determine which of the two hemispheres of a sample it belongs to. Then the inverse mapping of the hemisphere is used to find out which pixel in the irradiance map the direction lies within. This value is obtained from all eight surrounding samples, and trilinear interpolation is used to find the irradiance in direction ω at a given point.

Depending on which objects that needs to be rendered in the scene, a specular reflection may be pertinent. The Irradiance Volume normally handles only diffuse objects, but can be extended to simulate non-diffuse effects by storing higher order moments than the irradiance. With this extension simulation of such phenomena as Phong-like glossy reflections is possible, given the specularity is not too high. The reason for this is that the main assumption in the irradiance volume is that it is possible to interpolate between samples to get a good approximation of the irradiance, but as the specular function increases, this assumption moves towards being invalid.

In order to build a volume of higher order, equation 10.3 is rewritten to:

$$H^n(\omega) \approx \left(\frac{n+1}{2}\right)\frac{4\pi}{N}\sum_{i=1}^{N} L(\omega_i)max(0,\omega_i \cdot \omega)^n \tag{10.4}$$

The specular part is obtained by putting the reflected light to an order, in this case n is the order of the moment. ω is a cosine-weighted average of radiance samples on the ω-oriented hemisphere. As n increases, it has the effect of weighting the integration of the radiance towards ω, which means, with a high enough order, the specular reflectance map will equal the original radiance map.

10.3 Use of Irradiance Volume in the system

In the system developed for this project the implementation of the Irradiance volume function is generally as described in this chapter. With the exception that the radiance is given by an environment map instead of the traditional use of Irradiance volume, where it functions in conjunction with a method, like radiosity, that has already computed the radiance in the various parts of the scenes.

The irradiance volume created for the system takes as input a simple 3D model of the lobby scene. The lobby scene has been recorded as an environment map for the system and a 3D model has been created, that has a minimal set of faces that still describes the main walls of the lobby, and is at the same time a "closed" model. Closed meaning that if positioned inside the modeled scene, no matter which direction a ray is shot, it will always hit geometry. This is important, because if the ray does not hit anything in a given direction, the function will assume that the radiance coming from that direction is black, which is never the case in a given real scene like the lobby scene. The 3D model of the lobby scene can be seen in figure 10.5. The model has omitted the bridge, stairs and pillars of the real lobby scene, in order to speed up the creation of the irradiance volume, but the algorithm is able to handle a scene with arbitrary geometry, so if more detailed and correct representation of the real scene is desired for the irradiance volume, it is possible to do, simply by adding more geometry in the scene, that symbolises e.g. the bridge.

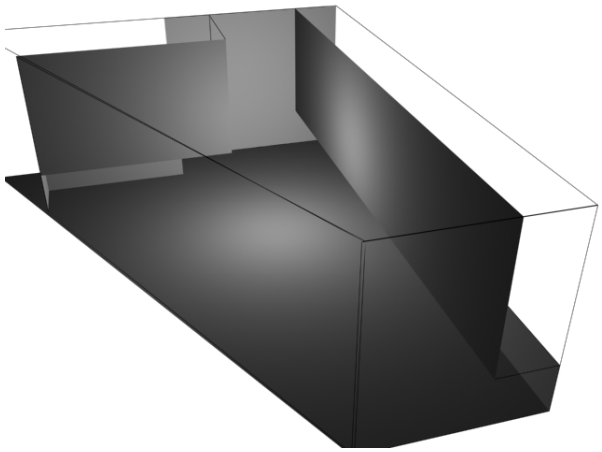


Figure 10.5: The reduced 3D model of the lobby scene.

The Irradiance volumes secondary input, the bounding volume, in this case an axis aligned bounding box (AABB), that controls the areas in which the objects are allowed to move within the scene. This is also the bounds in which the irradiance volume will be set up. The grid of the irradiance volume will start in each corner of the AABB, and subdivide each cell in the volume according to how many sub-levels the user has decided the irradiance volume should contain for the given scene.

The structure of the volume is constructed, as suggested by [Greger et al., 1995], by three entities, grids, cells, and samples. To construct the volume, these entities follows a couple of rules:

1. A grid always contains 8 cells.
2. A cell contains either a new grid or 8 samples.

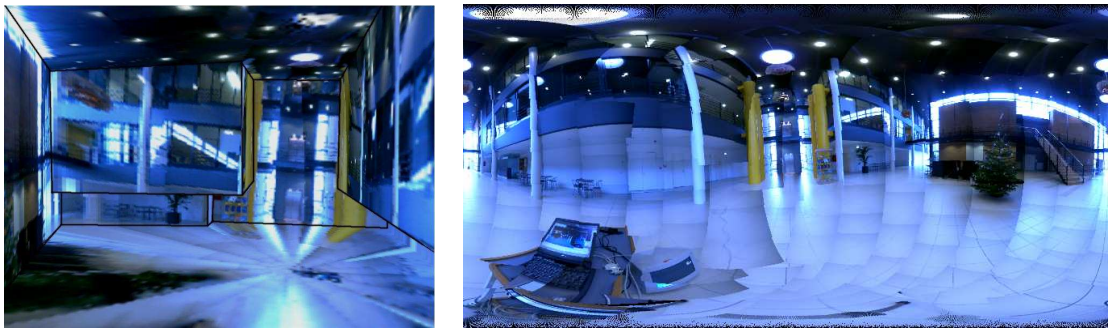
From these rules a irradiance volume grid is created from the bounding boxes given by the user. The

volume is first created by making a grid in the area confined by the bounding box. This first level of the grid will contain 8 cells.

Each cell is evaluated, and if it contains geometry, the cell is set to contain a grid, which again contains 8 new cells. This new grid is the next level of the volume. This process is repeated for as many iterations needed until it reaches a stop criteria. When the stop criteria has been met, all cells not containing grids, is set to contain 8 samples.

When the system creates the radiance samples in the environment, it shoots rays out into the model of the room, and the first surface that is hit, is being used to calculate which longitude-latitude coordinate it is corresponding to on the environment map of the room. The returned value from the environment map is used as the returned radiance for that direction. The irradiance volume in this system handles one environment map, and as such there will be areas of the scene, that is not described by the environment map because of occlusions. Multiple environment maps are currently not handled in the system. When the system shoots rays for each sample, it finds the 3D position of the intersection. If the 3D position from where the environment map for the scene was recorded is subtracted from the intersections 3D position, the result is a vector, that can be converted to a pixel coordinate on the environment map. This process is described in chapter 8: Modeling Real Scene Illumination, and done for each found intersection point. The process does not test if there is geometry between the intersection point and the environment map recording point, in which case the radiance found in the intersection point is not correct. In appendix C on page 133 the various intersection tests used in the system is described.

Multiple environment maps are not handled in the system, because arguably, it has little impact on the final result. For each sample taken in the environment, each direction is integrated over a hemisphere of directions, which mean that flaws in one directional radiance sample has little effect on the final irradiance sample, given most of the sampled directional radiances are correct. A visualisation of how the environment map is represented on the geometry, and thereby what is sampled by the irradiance volume is seen in figure 10.6a, the errors seen in the image where lines on the map does not match up with the lines in the geometry, is because the tracker either was not placed exactly horizontal during recording of the map, or that is has been put off by magnetic fields or nearby metal in the recording process.



(a) An environment map, mapped to the geometry of the lobby scene

(b) The corresponding environment map

Figure 10.6: An environment map, mapped to geometry,

When the radiance has been gathered for each sampling point in the scene, each radiance sample is converted into an irradiance sample. This is a very computational intensive process, if there are many samples in the system. Given that each sample has the same number of directional radiance samples,

and each directional radiance sample is oriented equally from sample to sample, a lookup-table can be created, that handles the $\omega_i \cdot \omega$ calculation once. If all radiance samples has the same mapping dimensions, this part of equation 10.3 is always the same for each sample, and so it can be pre-calculated once, and used by the system in the creation of the irradiance samples. The creation of irradiance samples now only needs to multiply each pre-calculated direction with their corresponding radiance sample, and the irradiance volume is created. In figure 10.7 a the samples are shown for the constructed irradiance volume, and the pink box is set into the scene as geometry, and the irradiance volume creates more samples around the geometry, as it should.

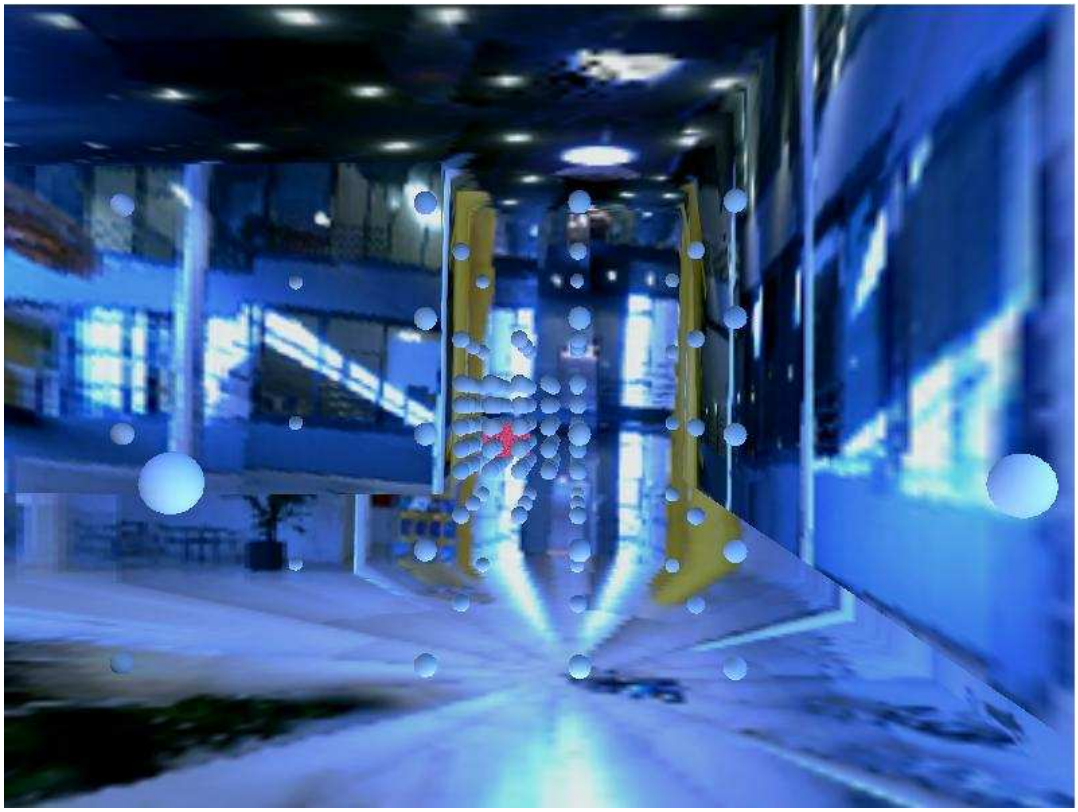


Figure 10.7: The samples for the lobby scene, based on the geometry projected environment map.

The irradiance volume is now ready for use. When an object within the confinements of the irradiance volume is to be shaded, a query is sent to the volume, that searches the structure to find the sample for the objects particular 3D position. The search in the structure is as described by [Greger et al., 1995] to start at the top-level, determine which cell contains the queried 3D position. If this cell contains a grid, the search descends one level and searches the new grid. Once a cell containing the 3D position has been found, the 8 samples of the cell is interpolated in regards to the queried 3D position.

Figure 10.8 on the next page shows a cell, where the 3D point p is to be interpolated from the 8 surrounding samples. To interpolate in 3 dimensions the length of the span of the cell in each direction must be calculated:

$$w_x = |\vec{p_4} - \vec{p_0}| \tag{10.5}$$
$$w_y = |\vec{p_2} - \vec{p_0}| \tag{10.6}$$
$$w_z = |\vec{p_1} - \vec{p_0}| \tag{10.7}$$

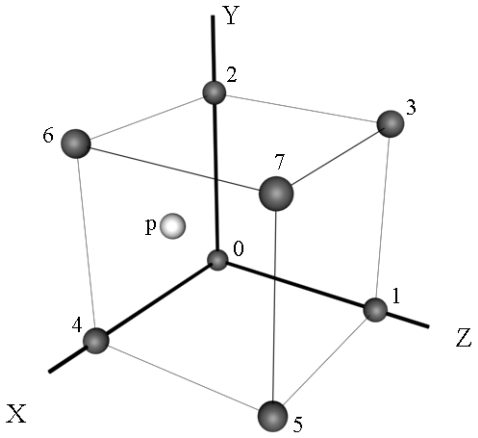


Figure 10.8: A cell, with 8 corner samples, and a point p, that is to be interpolated from the corner samples.

The interpolation is done locally, with p_0 marking the origin of the local coordinate system, and the position p must be re-based to a position p_n within the local coordinate system, to match this new origin:

$$\vec{p_n} = \vec{p} - \vec{p_0} \tag{10.8}$$

With these direction length, the weight per sample can be calculated:

$$w_0 = (\frac{w_x - \vec{p_{n.x}}}{w_x}) \cdot (\frac{w_y - \vec{p_{n.y}}}{w_y}) \cdot (\frac{w_z - \vec{p_{n.z}}}{w_z}) \tag{10.9}$$

$$w_1 = (\frac{w_x - \vec{p_{n.x}}}{w_x}) \cdot (\frac{w_y - \vec{p_{n.y}}}{w_y}) \cdot (\frac{\vec{p_{n.z}}}{w_z}) \tag{10.10}$$

$$w_2 = (\frac{w_x - \vec{p_{n.x}}}{w_x}) \cdot (\frac{\vec{p_{n.y}}}{w_y}) \cdot (\frac{w_z - \vec{p_{n.z}}}{w_z}) \tag{10.11}$$

$$w_3 = (\frac{w_x - \vec{p_{n.x}}}{w_x}) \cdot (\frac{\vec{p_{n.y}}}{w_y}) \cdot (\frac{\vec{p_{n.z}}}{w_z}) \tag{10.12}$$

$$w_4 = (\frac{\vec{p_{n.x}}}{w_x}) \cdot (\frac{w_y - \vec{p_{n.y}}}{w_y}) \cdot (\frac{w_z - \vec{p_{n.z}}}{w_z}) \tag{10.13}$$

$$w_5 = (\frac{\vec{p_{n.x}}}{w_x}) \cdot (\frac{w_y - \vec{p_{n.y}}}{w_y}) \cdot (\frac{\vec{p_{n.z}}}{w_z}) \tag{10.14}$$

$$w_6 = (\frac{\vec{p_{n.x}}}{w_x}) \cdot (\frac{\vec{p_{n.y}}}{w_y}) \cdot (\frac{w_z - \vec{p_{n.z}}}{w_z}) \tag{10.15}$$

$$w_7 = (\frac{\vec{p_{n.x}}}{w_x}) \cdot (\frac{\vec{p_{n.y}}}{w_y}) \cdot (\frac{\vec{p_{n.z}}}{w_z}) \tag{10.16}$$

The sample p can then be created with the following equation:

$$p_{color} = \vec{p_0} \cdot w_0 + \vec{p_1} \cdot w_1 + \cdots + \vec{p_7} \cdot w_7 \tag{10.17}$$

When the interpolation operation has been performed, equation 10.17 can be utilised to find the color in a requested direction, by taking the colors from the same directions from the 8 samples, and weight

them as seen above. In figure 10.9a a sample interpolated from 8 corner samples is seen in the lobby scene.

(a) *(b)*

Figure 10.9: (a):A sample interpolated from 8 corner samples. and (b):An arbitrary object shaded with our irradiance volume.

When the irradiance volume has been created, it can be used to shade arbitrary objects if it is given a position and normal direction for each vertex in the 3D object. This is done by finding the 8 samples, the vertex position is between, and find the weights of the 8 corner samples, to make the interpolation between them. Then the 4 directional samples, that are closest to the requested directional sample, the vertex normal, are found. The sample color is then found by interpolating between the 4 directional samples and the 8 positional samples. An example of this is seen in figure 10.9b. The shading of the vertices of an object using the irradiance sample is currently utilizing the diffuse Lambertian BRDF, where each sample in the hemisphere of directions is scaled by an albedo-value. Since the albedo-value is a constant, the same result can be obtained by scaling irradiance sample with an albedo after the irradiance has been calculated.

The implementation of the irradiance volume has shown, that it is important, to have more than one level before starting to subdivide the cells containing geometry. If the difference between the sizes of two adjacent cells is great the lighting of a sample has a tendency to "pop" when moving from one cell to another. The problem is particularly clear in cells of different size, that are close to a surface. But the problem is possible to fix with more initial levels of samples.

A problem that is clear from the samples in figure 10.7 on page 85 and 10.9, is that the range of the environment map is low. The intensity difference between the bright light sources of the scene is not realistic. The parts of the map where the most light is emitting from is saturated in the environment map. This means that the brightness of those pixels can be many times greater than the data indicates. This means that the light from a surface reflecting light can weigh as much as the light affecting it, if they both are saturated in the environment map. If the environment maps where captured as HDR images this problem would be corrected.

Another issue of the irradiance volume is that if there is a small but very dominant light source in the environment, it is possible that not all irradiance samples "catches" the light source when sampling in the different directions. One way to minimize this possible error, is to have more directional samples per irradiance sample. This however increases the processing time greatly. But the processing time is

mainly used on converting the radiance samples to irradiance samples. With this in mind, the super-sampling method could be applied, where a high amount of directional radiance samples are created for each positional sample, and afterwards averaged to a smaller hemisphere map. This way more points in the environment are hit, and the possibility to find the light sources are increased.

Another possible solution to the problem would be to manually create a point in the scene, that marks the light source, that all samples must take into account during calculations. Practically it could be implemented by finding the absolute direction to these possible light sources and find the four directional samples closest to the direction to the light source, and add the light of the source to these four samples.

CHAPTER 11

Radiosity

Radiosity is a technique that calculates the lighting in a complex diffuse-lighting environment, based on the scene's geometry. Because the radiosity calculations do not include the eye point of the viewer, the geometry and lighting in the environment do not need to be recalculated if the eye point changes. This enables the production of many scenes that are part of the same environment, and a "walk through" the environment in real time.
The amount of light in the scene is constant - light which is emitted is always absorbed somewhere. Every object is subdivided into many, many patches (or polygons). Each patch has a specification telling the system how this patch will affect other patches, that is, how it reflects light.

Source: [Elias, 2003b], [Akenine-Möller and Haines, 2002], [Nettle, 1999a], and [Nettle, 1999b].

11.1 Radiosity matrix

A light source emits energy into the scene, and the energy distribution to the various surfaces are calculated. Each surface has two associated values. A value of how much it is illuminated, and a value of its surplus of energy, i.e. how much energy it can radiate. In the beginning only the light source will have radiative energy, while all others have none.

Next step is to calculate the interaction of energy from each surface to every other surface. If the amount of surfaces is n, then the calculation of the interaction between the surfaces, this is an n^2 problem. The interaction may be calculated based on the surfaces geometrical relationships. This relationship between two surfaces is a single value and is called the "form factor".

All the calculated form factors in a scene may be stored in an $n \times n$ matrix, called the "radiosity matrix". Each element in the matrix contains a form factor for the interaction from the surface indexed by the column and the surface indexed by the row.

The radiosity matrix is set up as equation 11.1:

$$\begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix} = \begin{pmatrix} 1 - r_1 f_{11} & -r_1 f_{12} & \dots & -r_1 f_{1n} \\ r_2 f_{21} & 1 - r_2 f_{22} & \dots & -r_2 f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_n f_{n1} & r_n f_{n2} & \dots & 1 - r_n f_{nn} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{11.1}$$

Where e_i is each sources radiant exitance (i.e. a non-zero value for light sources), and r_i is the reflectance of patch i, f_{ij} is the form factor between two patches i and j. b_i is the radiosities that are to be calculated. To find the radiosity the matrix is solved.

To solve the matrix, each column, or source, is set to emit energy to each row, or destination, in that column. When this is done, the energy from the source has been distributed to all surfaces (destinations). But these surfaces serves as receivers and reflectors at the same time, which means they are going to

reflect some of the absorbed energy back into the scene. So when energy is being sent to other surfaces, the energy is divided into the two values each surface is assigned, the illumination value and the radiative value. The light energy will be sent back into the scene, as the matrix is processed. When the matrix is solved, all light has bounced around and the lighting of the scene has reached an equilibrium.

The classic version of radiosity has the following pipeline:

1. Generate Model
2. Compute Form Factors
3. Solve Radiosity Matrix
4. Render

In radiosity the basic principle is to remove the distinction between objects and light sources. Now, everything can be considered a potential light source.

The basic premises of radiosity are:

1. There is no difference between light sources and objects.
2. A surface in the scene is lit by all parts of the scene that are visible to it.

Anything that is visible is either emitting or reflecting light, i.e. it is a source of light. Everything the surfaces sees around them is a light source. And so, when considering how much light is reaching any part of a scene, care must be taken to add up light from all possible light sources.

11.2 Progressive refinement

[Cohen et al., 1988] published a paper in 1988 called "A Progressive Refinement Approach to Fast Radiosity Image Generation", that introduced a new way to solve radiosity by reordering the way things were done.

The traditional method had illumination being gathered by each destination from its source, hence the method was called "gathering". In progressive refinement this process is reversed and defined this as "shooting".

The basic idea is to start with finding surface with the most energy to contribute to the scene, the surface with the highest amount of radiative energy. This surface will then iterate through the rest of the surfaces in the scene, and distribute its energy as it progresses. The process starts finding the next surface with the most energy, and cycles through the other surfaces. this is repeated until the largest amount of radiative energy in the scene has met a predetermined criteria, that means that its contribution to the scene would be insignificant. When this process is finished an image would be rendered for the user.

11.3 Patch subdivision

The surfaces has only one illumination value assigned, which means there can be no change in illumination across surfaces. In radiosity all surfaces in a given scene may be subdivided into smaller polygons,

called patches, to accommodate this problem. Then all patches are treated as their own surface instead of the original surface. The drawback is that the radiosity matrix will grow to the number of patches in the scene squared.

This means that a very simple scene of 1000 polygons will have a matrix with 1 million elements. If each polygon is subdivided into 8×8 patches the matrix would contain 4 billion elements in the matrix ($(8 \cdot 8 \cdot 1000)^2 = 4,096,000,000$). This illustrates that even simple scenes may produce big problems.

Patches are used to simulate area light sources. Instead of treating a surface as an area light source, it is split up into smaller light sources across the original surface. With a adequate subdivision the result may be quite satisfactory.

The subdivision should be done intelligently. Instead of subdividing every surface into a set of patches one square unit each, the subdivision should be weighed in regards to the contrast in the different areas. Meaning high-contrast areas should have many patches, while similar neighboring areas should have less.

There are various techniques to make an intelligent subdivision. Progressive subdivision is the most commonly used method. Once a surface has emitted its energy, each patch in the dataset is evaluated to decide if two adjoining patches difference in illumination values is above a set subdivision criteria. If there is a big difference, there will be a high contrast between the two, and they should both be subdivided.

The next step is to subdivide the patches themselves, into elements. This is done for both performance- and aesthetic reasons. The subdivision into elements may preset to a specific resolution. Unlike surface subdivision, where the original surface is discarded and replaced by the patches, the patches are maintained when they are subdivided. The reason for this is that the patches may be used for shooting, and the elements may be used for gathering.

A patch is typically subdivided into 8×8 elements. The patch with the largest amount of radiative energy is, during the distribution process, chosen for shooting. From that patch, the energy is distributed to all of the elements in the scene. The elements stores and displays the illumination value they are given, and the radiative energy, that would be reflected from the different elements is transferred to their parent patch. The patch will then handle the shooting instead of the elements. This allows for a high surface geometry resolution with a low resolution for distribution. This may save a lot of processing time.

To exemplify the rendering of a scene, the process is shown from the point of view of a patch:

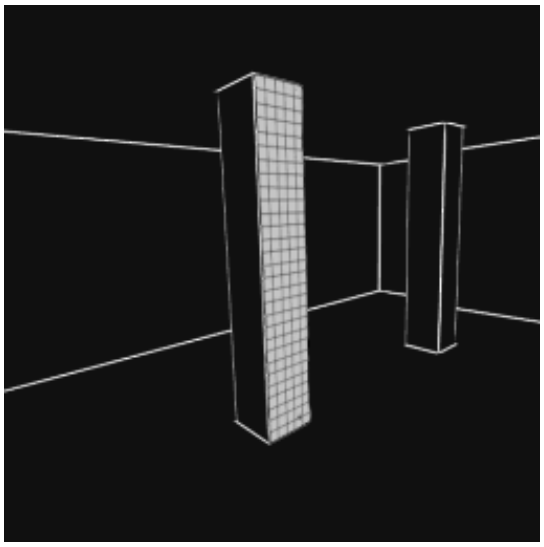
In the scene, all surfaces are subdivided into multiple patches. In figure 11.1a a surface is shown, and a patch on the surface is chosen in figure 11.1b.

If a virtual camera is placed on the patch looking outwards, it is possible to see what it sees, see figure 11.2a. The room is very dark, because no light has entered yet. The edges are drawn afterwards to be able to see the scene. If all the light it sees is added together, the total amount of light from the scene reaching the patch is calculated. This is the total incident light on the patch. This patch can only see the room and the darkness, by adding up the incident light, it is apparent that no light is arriving here. This patch is dark.

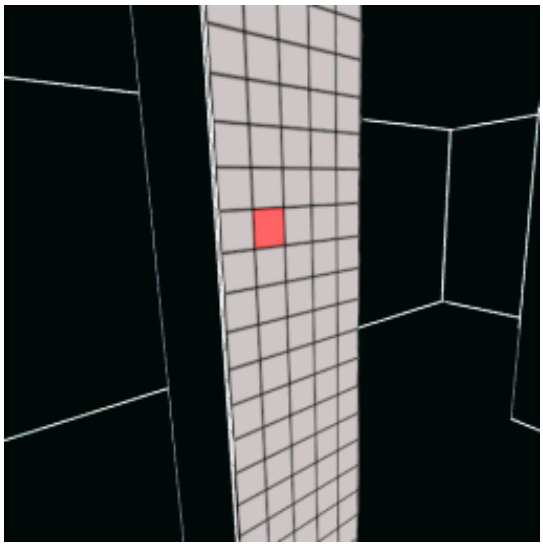
In figure 11.2b a patch placed lower on the surface is rendered, this patch can see the bright sun outside the window. If the incident is added up here, it will show that a lot of light is arriving at this patch (although the sun appears small, it is very bright). This patch is brightly lit.

This process is repeated for all the patches, and the scene will look like figure 11.3.

The patches near at top of the pillar, which is occluded from the sun, are in shadow, and those in eyesight

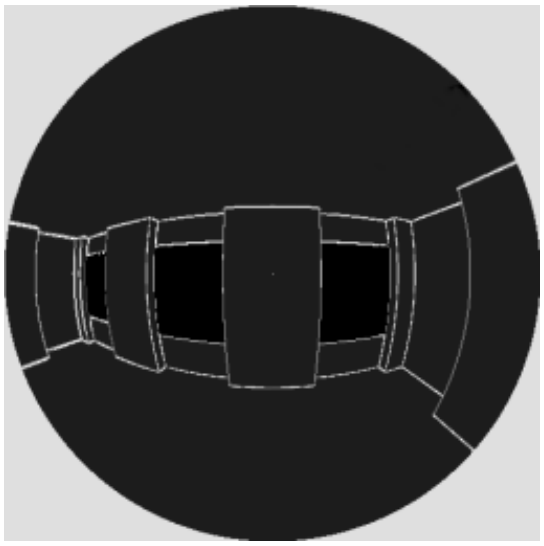


(a) A surface of a given scene is subdivided into patches

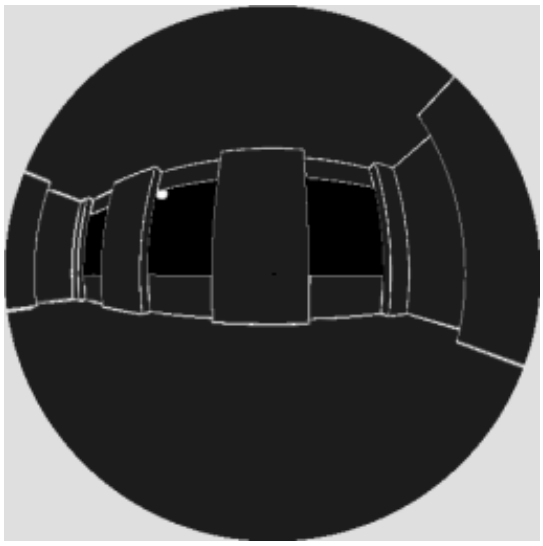


(b) A patch is chosen, and the scene is visualised from its perspective.

Figure 11.1: The surface chosen for the visualisation. [Elias, 2003b]

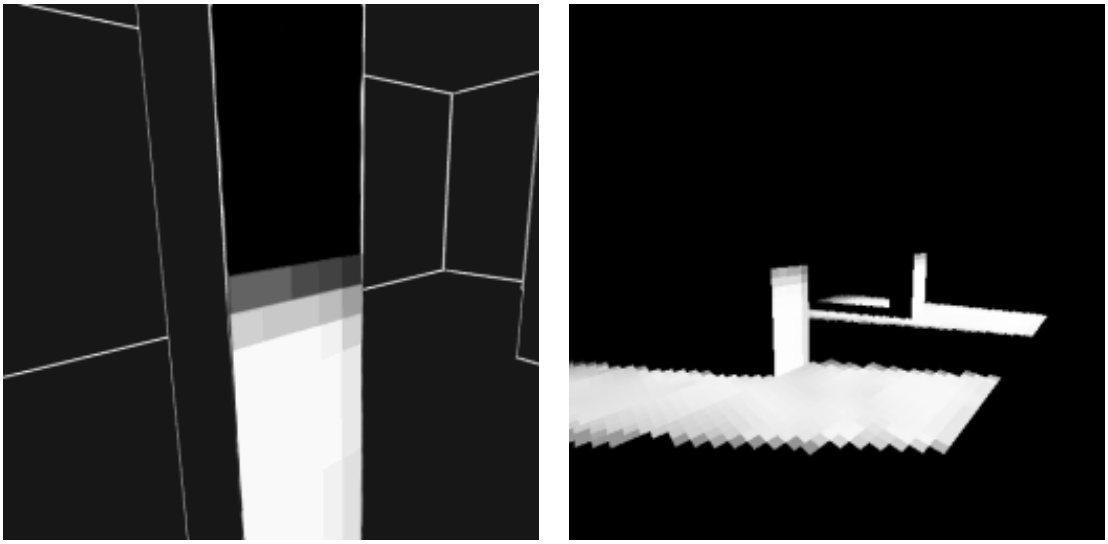


(a) The camera is places in the center of the patch, and its view of the world is seen



(b) The view from a lower patch

Figure 11.2: The view from two different patches, where (b) can see the light source. [Elias, 2003b]



(a) The lighting of the patches after 1st pass

(b) The lighting of the entire scene

Figure 11.3: The lighting of the scene after 1st pass. [Elias, 2003b]

with the sun are brightly lit. Those that partly see the sun, occluded by the edge of the window, are only dimly lit.

The Radiosity algorithm proceeds like this for more passes, where shadows naturally appear in parts of the scene that cannot see a source of light.

Now that some parts of the room are brightly lit, they themselves have become light sources, and can cast light onto other parts of the scene.

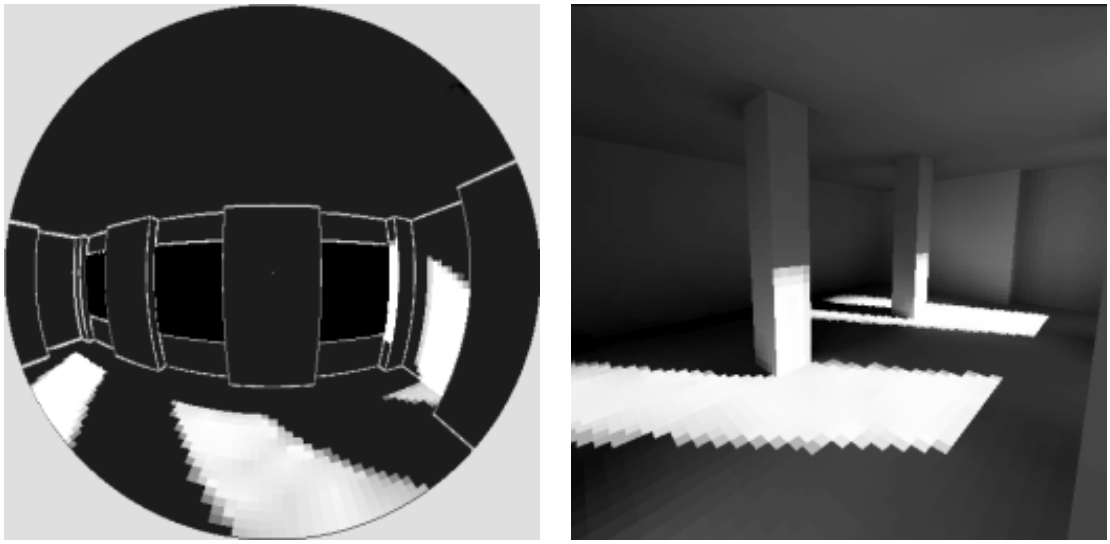
In figure 11.4a the patch that could not see the sun, and so received no light in the 1st pass, can now see the light shining on other surfaces. So in the following pass, this patch will become slightly brighter than the completely black it currently is. All patches are processed once again, and all patches that could see no light before, can now see the other lit up patches and use them as light sources. In figure 11.4b the 2nd pass has been performed, and the lighting of the room is greatly improved.

11.4 Accelerating Radiosity

To create the views from each patch, a hemisphere is placed over the patch, and from that patch's point of view, the scene is rendered on the inside of the hemisphere, and the texture will look just like the scene from the patch's point of view. The hemisphere was introduced by [Cohen and Greenberg, 1985]. There is no difference between a hemisphere and a hemicube from the view of a patch. A hemisphere for the patch in the previous example can be seen in figure 11.5a.

If the fish-eye view could be rendered easily, then the calculations would be to sum up the brightness of every pixel to calculate the total incident light on the patch. It is however not trivial to render a fish-eye view, and so some other ways has been found to calculate the incident light.

To obtain shadows visibility information is needed, to know how much of the various patches in the scene are visible from other patches. A common way to achieve this information is to use a z-buffer.



(a) The view from the patch after 1st pass

(b) The scene lit after 2nd pass

Figure 11.4: The incident light on the patches after 1st pass lets otherwise occluded patches see light, and the rest of the scene is already lit in the 2nd pass. [Elias, 2003b]

The z-buffer must be generated from the patch's point of view. One way to do this, is by using a hemicube. A hemicube equivalent to the hemisphere is shown in figure 11.5b.

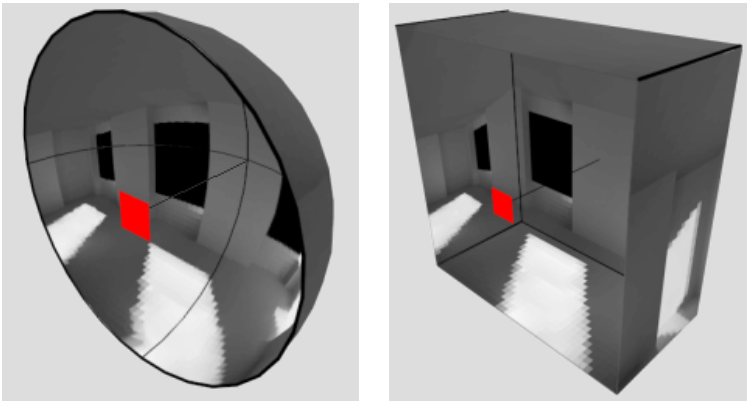
A hemicube is half a cube, split orthogonally along one axis. A pin-hole camera is placed at the base of the hemicube pointing at the top face, with a $90\,\mathrm{deg}$ frustum. The top face may be considered as the rendering surface of the camera. When the scene is rendered from this perspective, the camera sees what the patch sees.

Patches rendered onto the surface of the hemicube will occupy "hemicube pixels". Patches far away and with great angles of relative orientation will occupy fewest pixels. Using the z-buffer it is possible to create shadows by detecting which patches are partially or fully occluded.

These renders needs to be translated into energy transmissions. Normally the frame buffer stores color values and a z-buffer the depth values. Here the z-buffer keeps the depth information, while the frame buffer stores patch ID's instead. A complete render gives a partial form factor information of how much energy gets transmitted between patches. The render is partial because information of the relative angle between patches is missing.

Relative angles are used to decrease the amount of energy transferred from one patch to an other. A greater angle means less energy is transmitted. The render may tell how wide an angle the destination patch is relative to the camera. But the shooters angle also needs to be taken into account. Like Lambert shading, the surface receives less light, the more it turns away from the light. But the light source (the shooting patch) is an area light, which means it may also have an angle, and this angle needs to be taken into consideration before energy is transmitted.

To do this, a delta form factor may be utilised. The delta form factor is a simple table of values, with the same resolution as the surface of the hemicube, and it contains values that are used to scale the amount of energy that each pixel in the hemicube may transmit. Values in this table are highest in the center, and decreases towards the edge, as patches directly in front of the shooter will receive most energy, and



(a) The hemisphere for the patch.

(b) The equivalent hemicube

Figure 11.5: The hemisphere and hemicube, that can be created for a patch. [Elias, 2003b]

those with the most incident angle receive the least.

The shooting is performed through each pixel in the frame buffer, where the patch's ID's were stored. When each pixel, or patch, in the surface is processed, the shooters total radiative energy is scaled by the delta form factor associated with that pixel. And like that, the total energy of the shooter has been transmitted to the other patches in the scene.

11.5 Form factor calculation

The form factor is the calculation of how much energy is transmitted from one patch to another. When performing the form factor calculation, a criterion may be set, to ease the calculations: The area of two interacting patches must be relative small,when compared to the distance between them. This will result in the lines between two patches will be relative uniform to each other, as all points on the receiving patch will have nearly the same relative angle to all points on the transmitting patch. With this criterion met, each patch may be treated as one area, and the interaction between two patches is calculated once. If the relative angle between patches is kept low the errors, as a result of the simplification, are minimized.

To transmit from one area to another, differential areas are applicable. Differential areas are calculated by dividing the transmitting patch's area by the receiving patch's area. The result is a value that scales the amount of transmitted energy that will be received.

The standard form factor equation, according to [Cohen and Greenberg, 1985] and [Wallace et al., 1987], may be seen in eq. 11.2:

$$f_{i-j} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{cos\theta_i \cdot cos\theta_j}{\pi \cdot d^2} \cdot h_{i \cdot j} dA_j dA_i \tag{11.2}$$

i denotes the transmitting surface, and j denotes the receiving. But by utilizing the above mentioned criteria, the two integrals may be avoided, which simplifies equation 11.2 to equation 11.3:

$$f_{i-j} = \frac{cos\theta_i \cdot cos\theta_j}{\pi \cdot d^2} \cdot h_{i \cdot j} dA_j \tag{11.3}$$

In equation 11.3 the d is the distance from patch to patch. $h_{i \cdot j}$ is the relative visibility between two patches. And dA_j is the differential area.

11.6 Use of Radiosity in the system

In the AR system the utilization of radiosity is limited to shadow-casting. Normally radiosity is used to distribute energy into a scene, and thereby lighting it. In a scene where virtual objects must be integrated into an existing real image, some light calculations may be unnecessary, as the scene is already lit. In this case, the lighting of the scene must be estimated, and the virtual objects lit and casting shadows from the estimated light sources.

A radiosity solution, depending on which type, makes several passes to compute the light-distribution, with an environment-map of a given scene, these computations may be skipped, and the light information from the environment map may be utilised to determine from where the light affecting the objects, and thereby casting shadows, are positioned. So the system needs only a 1st pass radiosity calculation. This saves a lot of processing time for the system.

The radiosity algorithm designed for this project has an online and an offline process, the pseudocode for the offline process is:

Pseudo code 11.1 The offline part of the Radiosity utilization in the system.

1. Subdivide all Emitting objects.
2. Subdivide all Receiving objects.
3. Map Thresholded environment map to emission patches.
4. Emit energy to all receiving patches.
5. Find most significant energy additions to all receiving patches.
6. Omit least significant energy additions for all receiving patches.
7. Normalize remaining energy for all receiving patches.
8. Subdivide movement space.
9. Create lines between receiving and emitting patches.

The radiosity algorithm designed for the lobby scene project takes as input the same scene model as the irradiance volume, and seen in figure 11.6. This model is used for intersection testing during the radiosity generation, and can also be used as the model for generating receiving and emitting patches.

There are 3 classes of geometry that is utilised in the radiosity calculations:

1. Full scene model used for intersection test
2. Shadow receiving geometry
3. Light emitting geometry

The user has the possibility to input a specific version of the scene model, where only the surfaces that are likely to receive shadows from the objects are contained. This step is done to prevent the

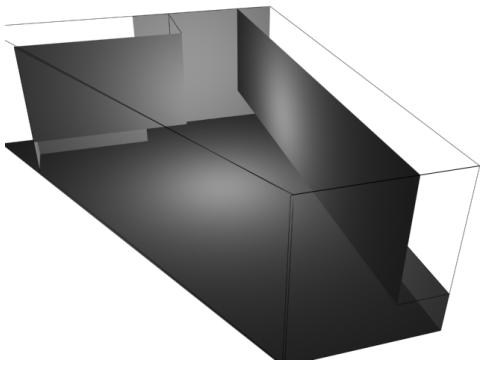
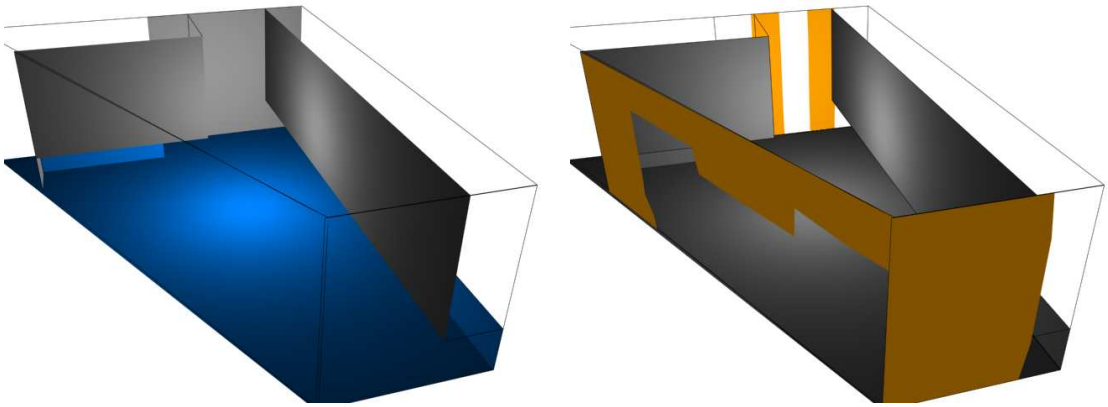


Figure 11.6: The low-polygon version of the lobby scene

algorithm to create a shadow on top of a window, and at the same time the speed of the system is greatly improved if no unnecessary patches are present within the scene, that will demand processing time during the online process. The user can, if decided set a model of which surfaces are to be used as possible emitting patches. The algorithm can as standard take the full scene model and use the contained surfaces as emitting surfaces, depending on the intensity their area has on the environment map. Figure 11.7 shows an example of surfaces marked as shadow area and light area respectively.



(a) The area marked by a user to be shadow receiving area

(b) The area marked by a user to be light emitting area

Figure 11.7: The user can mark areas that the algorithm is confined to find its emitting and receiving patches in.

The models for receiving geometry and emitting geometry is subdivided into several patches. The resolution of the patches can be set by the user, by stating the smallest size a patch may have, the subdivision creates the patches so their size is close to this set value. The receiving and emitting patches can be set to have different sizes, as it may be important that the receiving patches have a certain size, to look realistic. If the receiving patches are too big, the shadow may look blurry in regions that are supposed to have a sharp edge on the shadow. At the same time it may be desirable to have the emitting patches being bigger in regards to the performance of the system.

The size of both the emitting and receiving patches depends highly upon the size of the objects that are moving in the scene. The smaller the objects, the smaller the patches needs to be.

Many radiosity patch subdivision algorithms can adaptively subdivide the surfaces, so the areas with

the most contrast has the most patches. This approach is not applicable in the system for the lobby, as the virtual objects are moving dynamically in the scene. Therefore the subdivision must be uniform throughout the scene, to assure the same level of shadow detail wherever the shadow falls.

A patch in standard radiosity has emission, reflectance, incoming energy, and outgoing energy as variables. In the radiosity based on an environment map, only incoming energy and outgoing energy are preserved, as the emission and reflectance of a patch is not interesting in regards to finding the energy transmitted from the emitting to the receiving patches.

To use the environment map with radiosity shadows, each emitting patch is evaluated, for which area on the environment map it represents. For each emitting patch, the algorithm makes a sampling across its surface, and the average intensity of these samples is saved as the patches emitting or outgoing energy.

The next step is to calculate the energy transfers between the emitting and the receiving patches. To do this, equation 11.3 on page 95 is utilised, and calculated between all receiving and emitting patches.

Equation 11.3 looks like:

$$f_{i-j} = \frac{cos\theta_i \cdot cos\theta_j}{\pi \cdot d^2} \cdot h_{i \cdot j} dA_j$$

From the environment mapped to scene geometry, light will be transmitted to the receiving patches in the scene, the luminous intensity from the emitting patches off course given by the intensity in the environment map. During the radiosity calculation it will be noted for each receiving patch how much light it receives from each of the emitting patches. When all receiving patches has received light from all emitting patches, each receiving patch will be checked for which light sources adds to it's lighting, in this process each patch will have it's energy addition normalized, so the total received energy amounts to 1. If there are patches that only adds below a certain percentage of the total energy to each patch, that is, adding below what will be possible to see in the rendering of the patches, it will be omitted from the receiving patch's domain, this is mainly done to speed up the online process. In the process each line is intersection tested with the full scene model, to make sure the light does not pass through a wall and still lights a patch.

For the virtual objects to be able to cast shadow in the scene, lines will be set up from each receiving patch to each of the emitting patches, that adds to the patch. This is the reason why each receiving patch is checked for additions from each emitting patch, to omit the unnecessary links between patches. The line is drawn from the center of each patch.

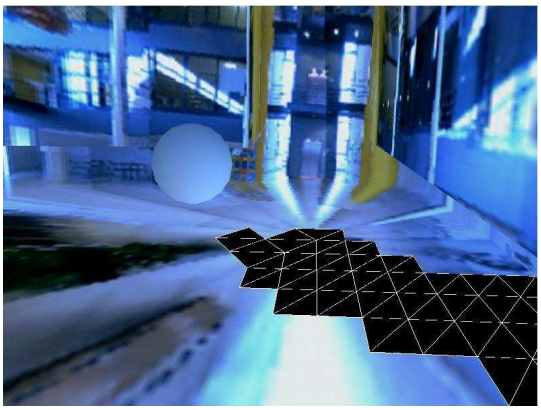
The Radiosity setup is now ready to cast shadows in the scene. The pseudocode for the online rendering of the radiosity is:

Pseudo code 11.2 The online part of the Radiosity utilization in the system.

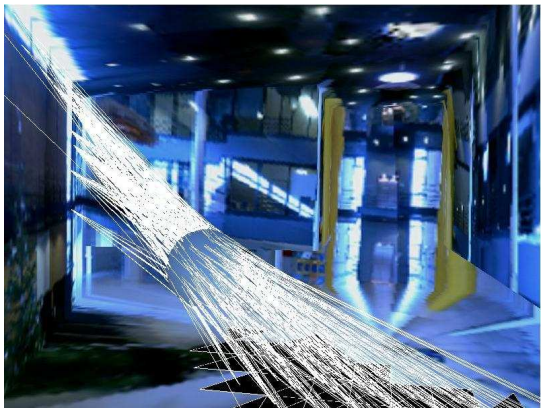
1. For each dynamic scene object, test if a line is intersected, and write which patch are affected in buffer.
2. Render patches in buffer as shadow.

If an object intersects any of the lines that has been set up, the intersection means that the receiving patch can no longer "see" the emitting patch that it shares the line with, and therefore the energy that was received from the occluded emitting patch is subtracted from the receiving patch. This will dampen the area the patch represents in the scene, and thus the object will cast shadows in the scene. In figure 11.8 an example is seen where all patches that are affected by the intersection of the lines associated with them is marked, as well as the lines that intersects the object.

To give each patch the look of a shadow, the alpha channel is affected at their vertices. The patches has



(a) The receiving patches that are affected by the position of the sphere is marked on the floor of the scene with black faces



(b) The lines the sphere is intersecting is marked

Figure 11.8: The patches affected by the sphere position and the lines that marks the occlusion of the various patches.

3 vertices, and has for this reason each vertex belongs to 6 patches. The shadow affecting each of these 6 faces affects their shared vertex. When a line leading to a receiving patch is intersected, the energy from the emitting patch the line was spanned from is subtracted from the total value (1) of the face. The energy from each emitting patch was normalised, therefore a subtraction of the various adding energy emitting patches will always result in a value between 1 and 0.

When the values for all receiving patches, that loses energy because of the occlusion of the object has been found, each affected receiving patch adds their $\frac{1}{6}$ of their energy to each of their 3 vertices, when all neighboring faces has added their energy, the vertex has the value of how much energy is added at that point in space. This process is not carried out on vertices that lies on the edge of the patch-network, and therefore does not have 6 neighboring faces. When this process is done the patches are rendered with the energy added to each vertex as the alpha value. In figure 11.9 an example is seen how some large patches are rendered with this technique.

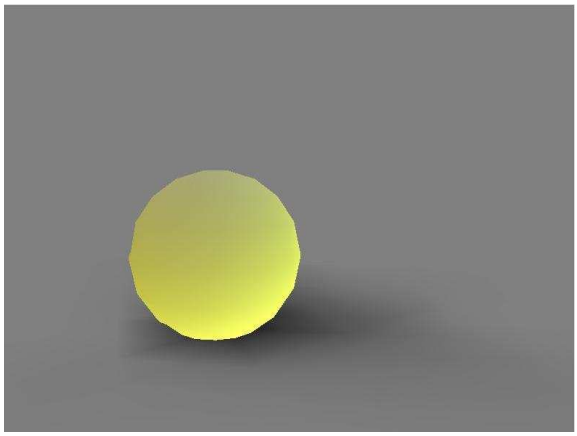
Through the implementation of the radiosity, it has been experienced, that though it is important to have a high resolution for the receiving patches, it is important as well, that the resolution for the emitting patches is not too low, as the objects may have a tendency to avoid the lines if they are too far apart. Again the resolution of the patches depends largely upon the size of the occluding object.

The current implementation of the radiosity tests for intersection with all lines every time it draws shadows. The algorithm could be greatly optimised if the it could detect in advance which lines that are likely to intersect the object. [Gibson et al.,] developed an algorithm that use a shaft hierarchy, with a coarse representation, that determines where the object is placed, and afterwards only tests the lines that possibly intersects the object.

When an object intersects a line, the current assumption of the algorithm is that the entire emitting face is invisible to the receiving face. If the representation should be correct, it should be tested if from some of the points on the receiving patch the emitting patch is visible, which would give a better precision in the shadow casting.

The current implementation of the radiosity shadowing algorithm only casts shadows based on the

(a) Shadows cast in the lobby scene, by coloring each vertex of the patches.



(b) Shadows cast on an invisible floor, ready to be superimposed on a video feed.

Figure 11.9: A sphere casting shadow onto patches, that are colored according to how much energy is subtracted from them.

objects bounding sphere. To yield the full benefit of the radiosity algorithm implemented, the ray intersection testing should include a low-polygon version of the objects that are to cast a shadow, when the bounding volume test has passed. This will create more realistic shadows of the object.

CHAPTER 12

The Final system

This chapter describes how the various methods described in the previous chapters are merged into one system capable of running a real-time augmented reality system

The system developed during the project is very similar the intended system described in chapter 3 on page 25. There are minor parts that was intended to be in the system, that were not finished at project deadline.

The system is divided into offline and online processes.

12.1 Offline process

In the offline process, an environment map is created, by recording the surrounding with a camera placed in the scene. The recorded environment map is then used by both the irradiance volume creation and radiosity calculations.

The irradiance volume generates samples throughout the scene, that indicates how the irradiance conditions are in the various sample positions. The samples are based on a 3D model of the scene, that has the environment map assigned.

The radiosity part of the system is calculating how light is distributed in the scene, based on the environment map. The surfaces that are taken into account by the radiosity calculations are reduced to be the surfaces most likely to receive and generate shadows. Rays are spanned between these surfaces, and are used to generate the shadows in the online part of the system. The offline rendering pipeline has been evaluated to fit the final system, and can be seen in figure 12.1.

12.2 Online process

In the online part, the camera is positioned in the scene. The camera data is captured by the system together with tracking data, indicating the orientation of the physical camera. The tracking data is used to transform the virtual objects in the scene, to correspond to the camera movement. There is a delay between the time the tracking data is received until the corresponding video data is received. The result is that during camera movement, the virtual objects are ahead of the rest of the scene. This problem is solvable by the use of a buffer on the tracking data, where the tracking data is time-stamped. The delay is measured, and the buffer is used to delay the tracking data by the delay time. This is currently not supported by the system.

Furthermore the isotrak 2 tracker used in the system has low precision, and when the image is supposed to be at a fixed position, the virtual objects, that are dependent on the tracking data, has a tendency to jitter in its position, which quickly reveals to the viewer, that the object is virtual.

When the system is started, the virtual model of the scene may not be aligned to the real scene. This is handled through a calibration routine, where the user points at a point on the virtual model, and

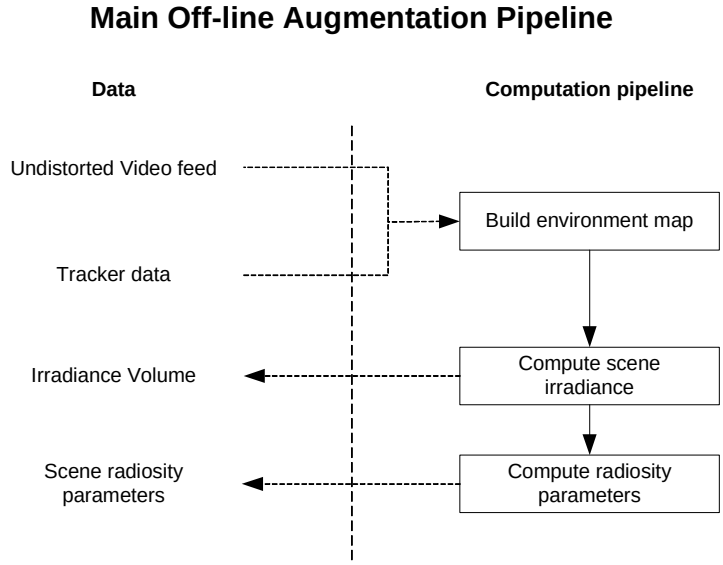


Figure 12.1: The offline rendering pipeline for the final system.

afterwards its corresponding point in the real scene. From this information, the system is able to realign the scene so the two points match.

The image data from the camera that is captured by the system is in the online process rendered as background to the virtual graphics. Next step is to render the occlusion data to make sure that if a virtual object is positioned behind a real object, that has been modeled in the virtual 3D model, the real object is displayed in front of the virtual object.

When the virtual objects are rendered, the irradiance volume is used for shading. At each vertex the irradiance volume is queried for the irradiance in the vertex normal direction. This value is used to color the each vertex on the object according to the value stated by the irradiance volume. The color value is scaled by the albedo for the object.

The rays that were set up in the offline process are used to cast shadows for each object. The rays are tested for intersection with each virtual object in the scene. The test is performed on the bounding sphere of the object, and if an intersection is detected, a shadow is casted on the area that was supposed to be lit by the ray. During the offline process the light addition that each ray represents to the patch is calculated. When a ray is occluded from the receiving surface, the receiving surface is dampened by the light addition the ray represented.

When the virtual objects are rendered with irradiance volume as shader, and radiosity as shadow generator, the image can be post-processed to further amplify the illusion that the image with the virtual object is recorded by the camera.

Three post-processing methods has been examined, Anti-aliasing, Motion blur, and video noise. None of these methods has been implemented by the deadline of the project.

Furthermore a method of updating the environment map used in the system for light calculations has been examined, but has due to the project deadline not been pursued.

The final system has the ability to record an environment map for a scene, and use it to create lighting for virtual objects it places within the scene, on top of the video-data from the connected camera.The online rendering pipeline has been evaluated to fit the final system, and can be seen in figure 12.2.

The class diagram for the final system is seen in appendix D on page 138.

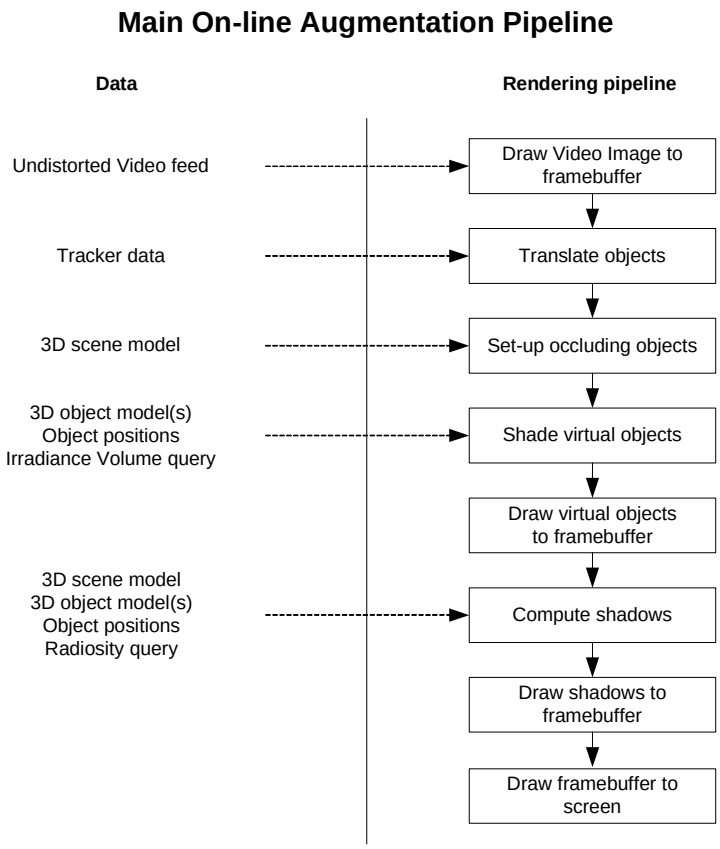


Figure 12.2: The online rendering pipeline for the final system.

Part III

Results and Discussion

CHAPTER 13

Test

This chapter contains the description of the various tests performed on the system, to evaluate the performance of the final system.

13.1 Irradiance sampling

13.1.1 Purpose

The purpose of this test is to visually verify that the system is able to sample irradiance in the defined bounding area.

13.1.2 Set-up

The test is carried out in the online part of the system, displaying the environment map on top of the video stream as virtual objects. All samples of the irradiance volume are displayed.

An environment map of the surroundings is created and is seen in figure 13.1

Figure 13.1: The environment map of the scene used in the test.

To verify how the environment map influences the irradiance samples within the volume, the irradiance volume is tested with a modified version as seen in figure 13.2 on the following page

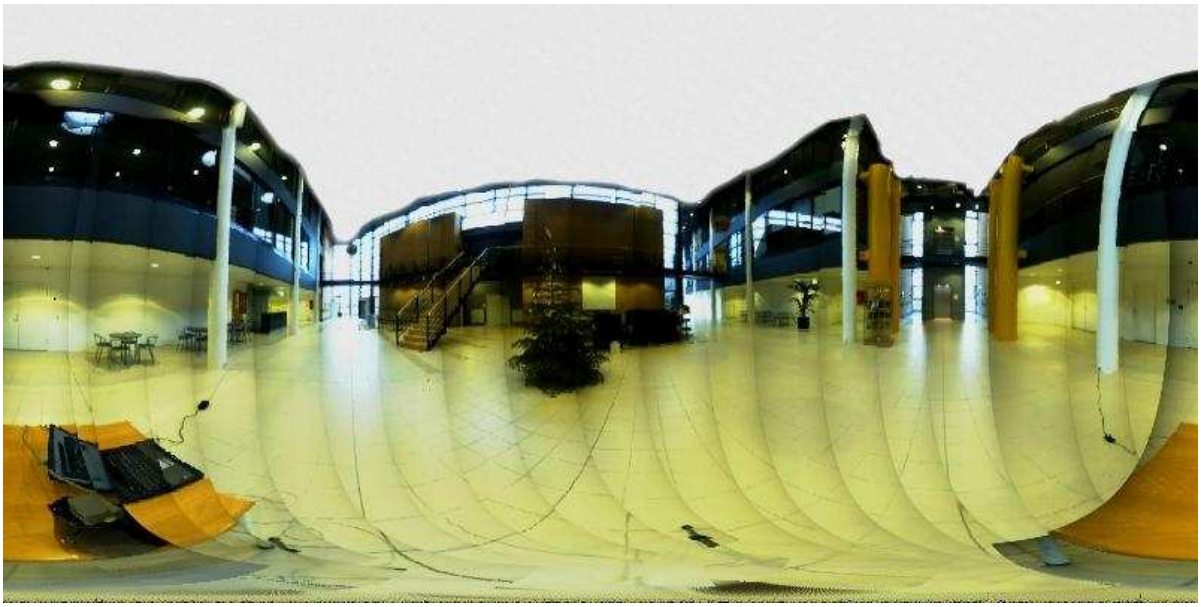


Figure 13.2: The modified environment map

13.1.3 Results and Discussion

In figure 13.3 on the next page the samples created from the recorded environment map is seen. The dark ceiling in the lobby is causing the samples to appear as if the dominant light source of the lobby is the floor. To see how the volume would react if the lamps in the ceiling has a bigger surface, and thereby has an increased chance to be sampled by each irradiance sample directional samples, the ceiling has been modified to be all light, and the result is seen in figure 13.4 on the facing page.

Through visual inspection, between the samples and their corresponding environment map, the samples are evaluated to match the environment maps. The samples furthermore matches the scene in which they are placed. The samples derived from the original environment map, seems to match the scene the best color-wise when the two sample images are compared. But taking the lights in the scene into account, which are difficult to see in the images, the modified samples matches the lighting of the scene best, though they are slightly overexposed.

13.2 Irradiance shading

13.2.1 Purpose

The purpose of this test is to determine how well the shading of arbitrary objects are utilizing the irradiance volume. To showcase the results of the project a dragon figurine has been selected to perform as test object.

13.2.2 Set-up

The environment maps used in the previous test are used to shade the objects in this test. Furthermore an additional environment map has been recorded for this test, with a bright sun illuminating the entire

Figure 13.3: The samples created in the irradiance volume is displayed in the lobby scene.

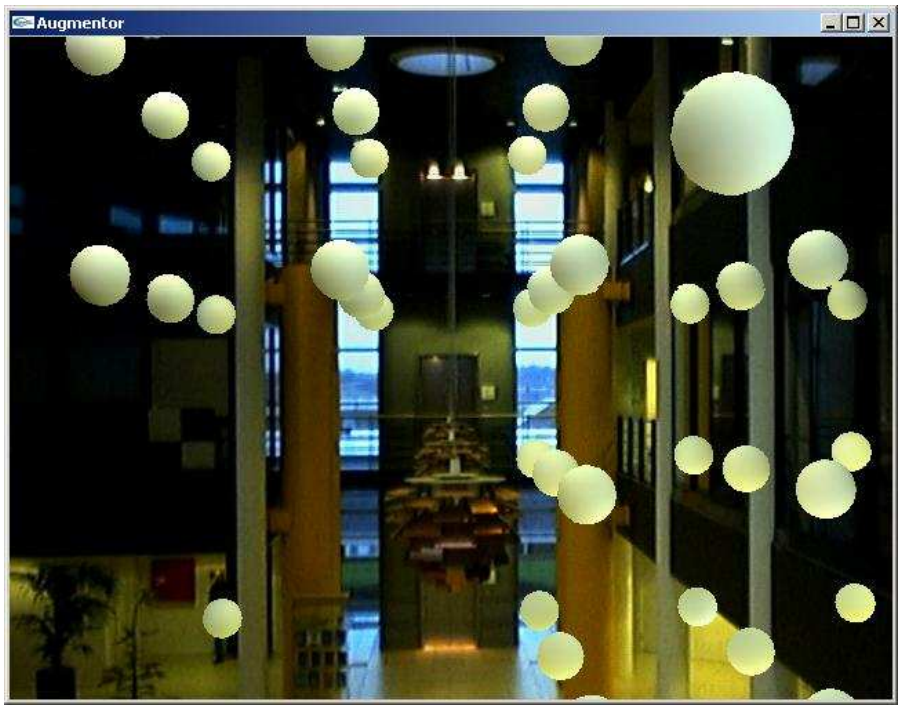


Figure 13.4: The samples created using the modified environment map.

scene through a window. The new environment map is seen in figure 13.5.

Figure 13.5: An environment map with a bright sun.

13.2.3 Results and Discussion

In figure 13.6 on the facing page the dragon figurine is shaded with the originally recorded environment map, and figure 13.7 on the next page shows the figurine shaded with the modified environment map. The figurine shaded with the original map looks more realistic than the figurine with the modified skylight.

In figure 13.8 on page 112 the figurine has been shaded with the environment map with the bright sun, and the brightness of the shading on the figure is matching up with the brightness on the surrounding environment. Other effects like shadowing and video noise could possibly be applied to the image to attain a more realistic looking image. The image furthermore shows that the system is flexible in regards to camera placement, the camera view is not locked to any fixed point, as long as the system is told what position the camera is placed in.

In figure 13.9 on page 112 and 13.10 on page 113 the figurine has been placed in two different positions of the scene, to visually compare if the shading has changed depending on the objects position. The shading of the object has changed from one image to the next, meaning the irradiance volume has found different samples for each position. Furthermore figure 13.10 shows that the occlusion handling has placed the object behind the metal rafter it is positioned behind.

13.3 Radiosity Shadows

13.3.1 Purpose

The purpose of this test is to evaluate the performance of the radiosity part of the system, that generates the shadows for the objects.

Figure 13.6: The dragon figurine is shaded with the originally recorded environment map.

Figure 13.7: The figurine shaded with the modified environment map.

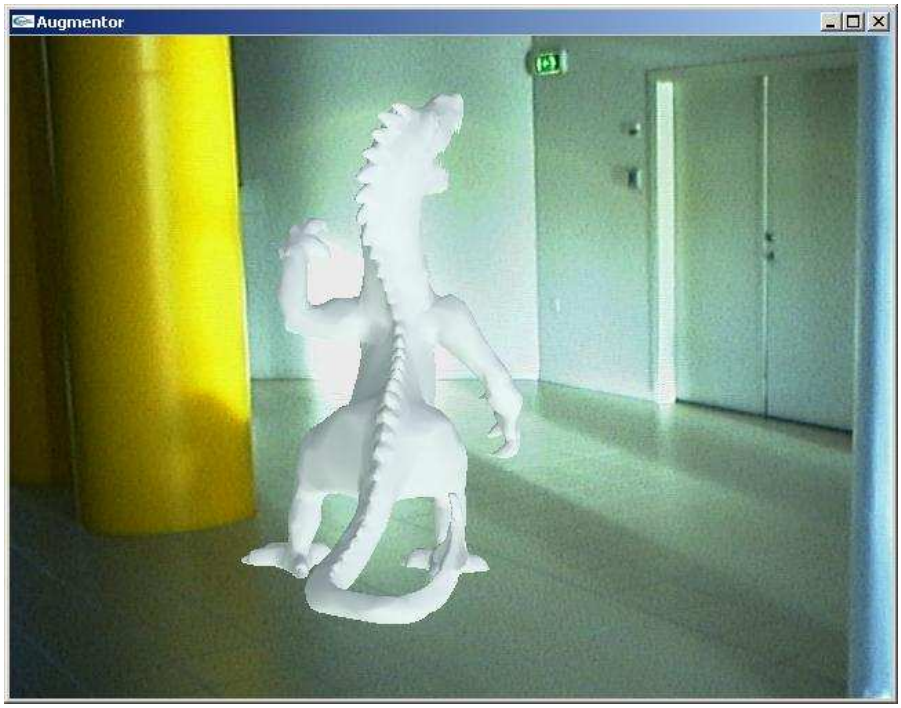


Figure 13.8: The figurine shaded with the sunny environment map.

Figure 13.9: The dragon is placed at two different positions in the lobby scene selected for the project, for shading comparison. Shadows are disabled for the shot.

Figure 13.10: The figurine has been moved to a new position, to show shading changes, and occlusion handling.

13.3.2 Set-up

The test is carried out in the full implementation of the system. A single virtual object, in this case the dragon, has a bounding sphere as its shadow casting object. The originally recorded environment map is used by the radiosity part of the system in this test.

13.3.3 Results and Discussion

Figures 13.11 on the next page and 13.12 on the following page shows the dragon figurine with a shadow projected on the floor underneath it.

The sense of placement for the figurine is better, when the shadow is added to the image. Furthermore the shadows enhances the illusion that the virtual object is placed within the scene. The shadow projected for the object is not a precise shadow for the geometry of the object. An improvement on the shadow casting can further enhance the merging of the two images.

13.4 Flexibility

13.4.1 Purpose

This test is carried out to test the systems ability to perform in other scenarios than the one it was designed for. The environment for the system is changed to the environment seen in figure 13.13 on page 115.

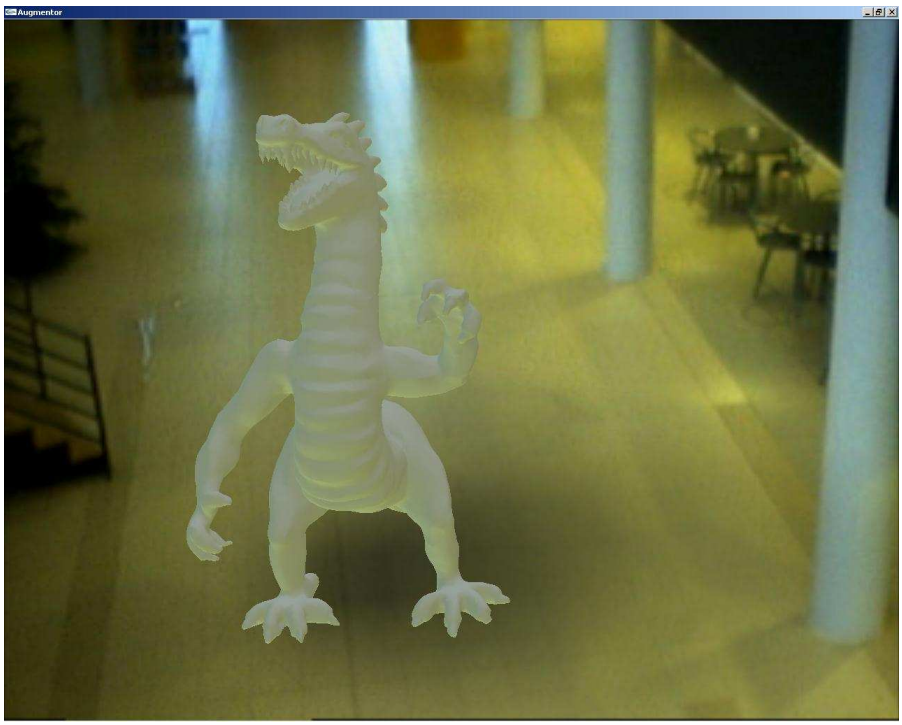


Figure 13.11: The object is shaded as if light is coming from above at the same time as the floor is shading from below, with an added shadow cast by the objects bounding sphere.

Figure 13.12: The virtual dragon figurine has attracted a worshipper.

Figure 13.13: The environment map for the group room scene.

13.4.2 Set-up

The test is performed on an implementation of the system without radiosity shadows.

13.4.3 Results and Discussion

In figure 13.14 on the following page the result of placing the figurine in another and smaller confined scene is displayed. The shading of the object is dominated by the lights in the ceiling of the room.

The test shows the systems ability to generically emulate various environment illuminations.

13.5 Comparison with rendered image

13.5.1 Purpose

The purpose of this test is to determine the quality of the shading solution implemented for the system.

13.5.2 Set-up

The test object is rendered in 3D Studio MAX 7's advanced light tracer renderer using the recorded environment map. The same object, the dragon figurine, is rendered in the system framework using the irradiance volume to shade the object. The two images are compared.

The test machine has the following data: AMD Athlon 1800+ processor, 256 mb RAM, GeForce 3 Ti500, Windows XP SP1.

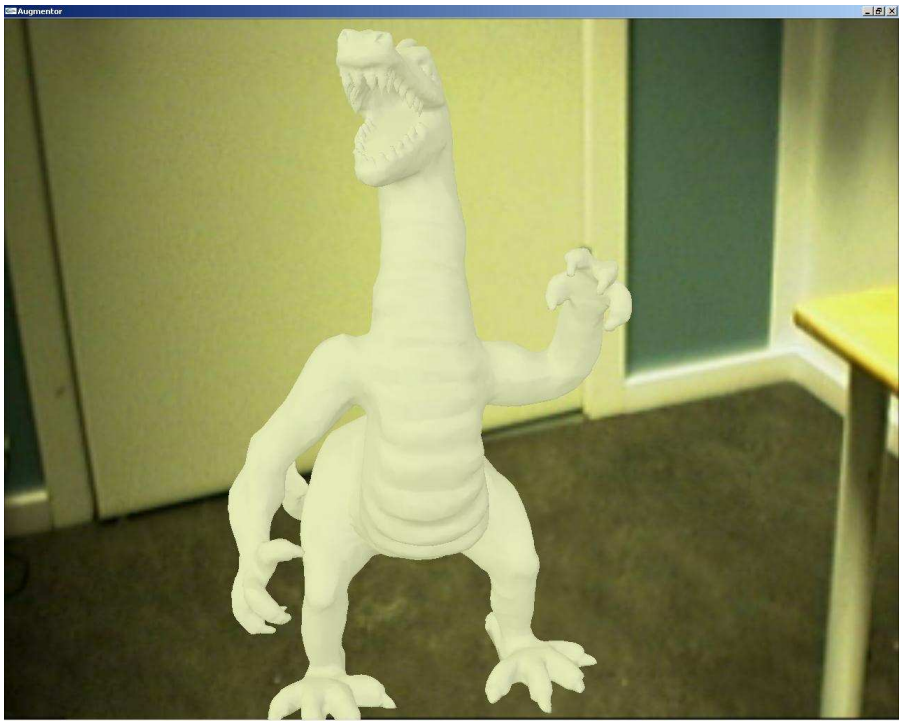


Figure 13.14: The object is placed in a smaller confine, close to walls. The scene is a group room.

13.5.3 Results and Discussion

In figure 13.15a the rendering performed by 3D Studio MAX is seen, and the corresponding shading performed by the irradiance renderer used in the system is seen in figure 13.15b.

The two renderings are performed with an albedo of 0.5. The biggest difference between the two is that the rendering performed by 3D Studio MAX has self-shadowing in crevice-areas of the figurine. This is not supported by the irradiance volume. The rendering time for the 3D Studio MAX image is 4:47 minutes. The irradiance volume is calculated and the rendering is displayed 2.2 seconds after startup on the same machine.

The rendering comparison shows that the irradiance volume calculations are performed correctly by the system.

13.6 Performance

13.6.1 Purpose

The purpose of this test is to evaluate the performance of the system, measured in frames per second.

13.6.2 Set-up

The test is carried out in the full implementation of the system. Virtual objects, the dragon figurine, and a mannequin mesh, is inserted into separate lobby scenes. The frame rate is observed with shading only and with both shading and shadowing. The object is moved to different locations in the scene.

(a) *(b)*

Figure 13.15: The figurine is rendered in (a) 3D studio MAX 7 advanced lighting and (b) using the irradiance volume shader.

The test machine has the following data: AMD Athlon 1800+ processor, 256 mb RAM, GeForce 3 Ti500, Windows XP SP1.

13.6.3 Results and Discussion

The dragon figurine consists of 17477 vertices, and 34951 faces. The irradiance volume algorithm currently performs a lookup per vertex per face. Each face consists of 3 vertices, so in order to shade the dragon figurine the irradiance volume must perform 104853 lookups per frame.

The mannequin is defined by 1594 vertices, and 3166 faces. This yields a lookup rate per frame on 9498 lookups.

The radiosity implementation used in the test has 3200 receiving patches, and 50 emitting patches, which means that the radiosity must perform 160000 intersection tests per frame rendered to the screen.

The performance data can be seen in table 13.1 and 13.2.

Mannequin	**Shading only**	**Shading and Shadows**
Standing still	37 fps	37 fps
Movement	10 fps	0.9 fps

Table 13.1: The performance data for the mannequin model.

Dragon	**Shading only**	**Shading and Shadows**
Standing still	37 fps	37 fps
Movement	4 fps	0.8 fps

Table 13.2: The performance data for the dragon model.

The irradiance performs $4 \cdot 104853 = 419412$ lookups per second on the dragon figurine, and $10 \cdot 9498 =$

94980 lookups per second on the mannequin model. This shows that the frame rate is partly dependant on the lookup in the Irradiance Volume.

The initialisation of the radiosity has been measured to take 34 seconds, while the irradiance volume is initialised in the given scene in 2 seconds.

The data shows that the shadow calculations is the part of the augmentation pipeline which is the most computational expensive.

CHAPTER 14

Conclusion

This chapter concludes the Real-time Augmented Reality project made by group 922 on the Computer Vision and Graphics specialisation from the Laboratory of Computer Vision and Media Technology, Aalborg University.

14.1 Purpose of the project

The goal of this project is to create a real-time Augmented Reality system. The objective of the system will be to place virtual objects within a real scene, enabling these to interact with existing light, shadow, and occlusion. The user will be able to interact with both the viewing direction and the virtual objects.

14.2 Methods

In this project, the implementation of a real-time Augmented Reality system, has been explored. The primary aim has been how to move towards photo-realistic rendering, enabling seamless blending of the real world and virtual objects, in a real-time system. Two primary areas have received an in-depth analysis, these are global illumination of virtual objects, and shadow casting, both image based methods.

The system developed for the project combines information obtained from a standard webcam with orientation information from a hardware tracking unit, to enable virtual 3D objects to be placed into the video image. The tracking data enables transformation of the virtual objects with respect to the orientation of the camera.

The system handles occlusions between the real world and the virtual objects by the means of a prebuilt 3D model of the environment or scene the application is augmenting.

14.2.1 Modelling scene illumination

The process of rendering realistic shaded augmented objects has been the emphasis of the project. The project has explored how global illumination can be extracted from images of the surrounding environment. The process of extracting illumination information and modelling scene illumination is done in an offline process.

To capture the light information of the scene to be augmented, an environment map is generated using a stand-alone software tool developed in the project. The tool is able to construct spherical environment maps of user defined sizes, covering all view angles, automatically. This is done based on images from a camera with a tracker, registering orientation, fixed to it.

This environment map is in conjunction with a 3D model of the surrounding environment utilised to generate an Irradiance Volume.

The Irradiance Volume is a volumetric representation of the global illumination within a space based on the radiometric quantity irradiance. The irradiance volume provides a realistic approximation without the amount of calculations needed with traditional global illumination algorithms. This approximated irradiance is utilised to shade the virtual objects within the scene. Because the irradiance volume generation is an image based method, using an environment map, the objects are shaded based on the actual radiance from the surroundings.

14.2.2 Shadow casting

To generate the shadows in the image, a reduced radiosity solution is employed. Radiosity calculates the light energy received by every surface in a scene, taking into account light bounces from all surfaces. The solution used in the system creates light emitting surfaces in the areas generating light, based on the environment map, and light receiving surfaces in the areas of the scene that are likely to receive light. The light energy for a given emitting surface in the 3D model is based on an intensity measurement in the matching area of the environment map. The light emitted and received by the various surfaces is spanning rays throughout the scene. When the virtual objects bounding volume intersects these rays, the surfaces that where supposed to receive the light, are darkened by the amount the occluded light ray added to the specific point, thus creating a shadow.

14.2.3 Post processing

When objects have been shaded and shadows have been cast, a post-processing step can be added to further enhance the illusion that a virtual object is placed within the real scene. Various post-processing methods have been explored. Anti-aliasing of the virtual object removes the jagged and sharp edges between an object and its surroundings, while motion blur emulates the motion blur effect of the camera. Video noise can be matched by adding additional noise to the image, improving the illusion that the virtual objects are in fact a part of the real image.

14.2.4 Dynamically changing environment

The lighting conditions in the scene chosen for this project dynamically changes during the course of a day. During daytime the large windows sections provides illumination from sunlight, in the dark hours incandescent lights are the main illumination sources. Clearly a static environment map will cause the shaded objects to stand out from the real image, if the scene illumination has changed significantly from the time the environment map was created. To handle this effects the system must be able to adjust the environment map to match the current scene illumination. Methods to assimilate these dynamic changes have been explored, and there are several solutions. A preliminary solution of adjusting the brightness of the virtual objects by the difference between the offline-recorded environment map and the online recordable environment have been suggested.

14.3 Results

The irradiance Volume creates samples throughout the scene augmented by the system, and uses the calculated samples to correctly shade the virtual objects that are augmented into the real scene. The

Irradiance Volume bases its resulting shading of objects on an environment map recorded of the environment. If the environment changes, the irradiance used to shade the objects does not.

The shading of the objects is updated depending on where in the scene they are placed, and if an object moves behind a real object present in the virtual model of the scene, the object will appear as if it is placed behind the real object.

The shadows generated by the radiosity part of the system, is currently not able to project an exact shadow, but adds a sense of placement to the objects, and enhances the merging of the real and virtual part of the image.

The system is not in any way locked to one scene or one camera placement, but can be used in other environments than the lobby scene it has been developed for.

The resulting shading of the system is comparable to that of a global illumination rendering performed by a commercial light tracer program.

The creation of the Irradiance Volume is fast in the system, the test has measured the initialisation time to be 2 seconds. The shading of objects in the system is performed by lookups in the Irradiance Volume, and observations while performing lookups shows that the frame rate is partially dependant on the lookups in the volume.

The shadows generated in the system are based on radiosity calculations that are performed in the initialisation phase of the system. The initialisation is measured to take 34 seconds with 3250 patches. The online part tests 160000 rays for intersection on the objects per frame, which in the current implementation has a poor performance, below 1 frame per second, when an object is in motion.

14.4 Future possibilities

This section will explore some of the interesting future possibilities of improvement in the existing system. The following sections describe both performance enhancing improvements and some new functionalities for the system.

14.4.1 Tracking and frame grabbing

The magnetic tracker used in this project is limited by both a low resolution and a limited range of operation. Some interesting improvements to the system includes more accurate tracking and a tracker with a larger range of operation. A more accurate tracker will reduce the jitter augmented objects are prone to, resulting in a more realistic appearance. Using a tracker with an extended range of operation, taking into account not only the orientation of the camera, but also the position could enable the viewer to move the camera around augmented objects. This has not been an objective in the current project, but could provide the system with more flexibility.

14.4.2 Environment maps

The limited dynamic range of an environment map created in a standard image format is problematic when shading objects. To correctly reflect the dynamic range in real scene illumination it is necessary to create the environment map using High Dynamic Range Images (HDRI). This entails grabbing two or more images in the same direction at different exposure times, combining these to a HDR image. It is expected that HDR images will greatly improve the realism of the shading.

14.4.3 Illumination and shadow casting

When generating an irradiance volume the resolution of sampling directions is a tradeoff between rendering speed and details captured into the irradiance distribution functions. A low resolution of sampling directions often causes light sources with a small surface area to be completely missed by the sampler, even though they may represent a powerful light source. This problem may be remedied by super-sampling during the building of the irradiance volume. Super-sampling increases the possibility that any given light source is captured, by increasing the sample directions during sampling. When sampling is finished, the super-sampled data is down-sampled to a lower resolution. This maintains all the super-sampled data in a lower resolution.

The performance of the current radiosity implementation can be increased by partitioning the entire scene into a series of bounding boxes. Currently intersection testing is performed in the entire scene on a per-frame basis. This reduces performance considerably. Building a hierarchy of bounding boxes with known ray intersections, can reduce the number of intersection tests considerably, thereby increasing performance.

Another limitation in the radiosity implementation is in the object types that can generate shadows. Currently the system is only able to generate shadows from spheres. The reason for this is to minimise the complexity of the intersection tests. To enable more accurate shadowing, a low polygon version of the object to be shaded could be used. This low polygon object can either be created manually or a polygon reduction algorithm could be implemented as to automatically generate the shadow casting object.

Bibliography

[iso, 2001] (2001). *ISOTRAK 2 - User's manual*. Polhemus Incorporated, Colchester, Vermont USA, rev. a edition.

[Akenine-Möller and Haines, 2002] Akenine-Möller, T. and Haines, E. (2002). *Real-Time Rendering*. A K Peters Ltd, 2 edition.

[Andersson and Akenine-Möller, 2003] Andersson, U. and Akenine-Möller, T. (2003). A geometry based shadow volume algorithm using graphics hardware. `http://graphics.cs.lth.se/research/shadows/`.

[ARTHUR, 2004] ARTHUR (2004). Augmented round table for architecture and urban planning. `http://www.fit.fraunhofer.de/projekte/arthur/index_en.xml`.

[Behrens, 1995] Behrens, U. (1995). Opengl reference page. `http://www.mevis.de/~uwe/opengl/glPushMatrix.html`.

[Bergen County Academies, 2004] Bergen County Academies, H. (2004). Anti-aliasing. `http://www.bergen.org/AAST/ComputerAnimation/Graph_AntiAlias.html`.

[Bouguet, 2004] Bouguet, J.-Y. (2004). Camera calibration toolbox for matlab. `http://www.vision.caltech.edu/brouguetj/calib\_doc/`.

[Cohen et al., 1988] Cohen, M. F., Chen, S. E., Wallace, J. R., and Greenberg, D. P. (1988). A progressive refinement approach to fast radiosity image generation. *Computer Graphics*, 22(4):75–84.

[Cohen and Greenberg, 1985] Cohen, M. F. and Greenberg, D. P. (1985). The hemi-cube: A radiosity solution for complex environments. *Computer Graphics*, 19(3):31–40.

[Debevec, 1998] Debevec, P. (1998). Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography.

[Elias, 2003a] Elias, H. (2003a). Motion blur. `http://freespace.virgin.net/hugo.elias/graphics/x_motion.htm`.

[Elias, 2003b] Elias, H. (2003b). Radiosity. `http://freespace.virgin.net/hugo.elias/radiosity/radiosity.htm`.

[Gibson et al.,] Gibson, S., Cook, J., Howard, T., and Hubbold, R. Rapid shadow generation in real-world lighting environments.

[Gonzalez and Woods, 2002] Gonzalez, R. C. and Woods, R. E. (2002). *Digital Image Processing*. Prentice-Hall Inc, 2 edition.

[Greger et al., 1995] Greger, G., Shirley, P., Hubbard, P. M., and Greenberg, D. P. (1995). The irradiance volume.

[Kilgard, 1997] Kilgard, M. (1997). Opengl-based real-time shadows. `http://www.opengl.org/resources/code/rendering/mjktips/rts/index.html`.

[LightWorks-User.com, 2004] LightWorks-User.com (2004). Lightworks-user.com. `http://www.lightworks-user.com/hdri.htm`.

[Madsen et al., 2003] Madsen, C. B., Sørensen, M. K. D., and Vittrup, M. (2003). Estimating position and radiances of a small number of light sources for real-time image-based lighting.

[Milgram and Kishino, 1994] Milgram, P. and Kishino, F. (1994). Taxonomy of mixed reality visual displays. *IEICE Transactions on Information and Systems*, Vol.E77-D Issue.12(1321-1329):9.

[Nettle, 1999a] Nettle, P. (1999a). Radiosity in english. `http://www.fluidstudios.com`.

[Nettle, 1999b] Nettle, P. (1999b). Radiosity in english ii: Form factor calculations. `http://www.fluidstudios.com`.

[openGL.org, 2004] openGL.org (2004). Opengl.org. `http://www.opengl.org/resources/faq/technical/transformations.htm`.

[Ramamoorthi and Hanrahan,] Ramamoorthi, R. and Hanrahan, P. An efficient representation for irradiance environment.

[Ruether, 2002] Ruether, D. (2002). On video image characteristics and faults. `http://www.ferrario.com/ruether/vid_pict_characts.htm`.

[Sato et al.,] Sato, I., Sati, Y., and Ikeuchi, K. *Illumination distribution from shadows.*

[Shirley, 1998] Shirley, P. (1998). Monte carlo methods in rendering. `http://www.siggraph.org/education/materials/siggraph_courses/S98/05/c05%.pdf`.

[Shirley and Chiu, 1994] Shirley, P. and Chiu, K. (1994). Motes on adaptive quadrature on the hemisphere.

[SigGraph, 2004] SigGraph (2004). Aliasing problems and anti-aliasing techniques. `http://www.siggraph.org/education/materials/HyperGraph/aliasing/alias0.%htm`.

[Slater et al., 2001] Slater, Steed, and Chrysanthou (2001). *Computer Graphics And Virtual Environments: From Realism To Real-Time*. ADDISON-WESLEY, 1 edition.

[Vignaud, 2001] Vignaud, S. (2001). Real time soft shadows on geforce class hardware. `http://tfpsly.planet-d.net/english/3d/SoftShadows.html`.

[Wallace et al., 1987] Wallace, J. R., Cohen, M. F., and Greenberg, D. P. (1987). A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. *Computer Graphics*, 21(4):311–320.

[Weisstein, 2004] Weisstein, E. W. (2004). Euler angles. `http://mathworld.wolfram.com/EulerAngles.html`.

[Wellner, 1993] Wellner, P. (1993). Interacting with paper on the DigitalDesk. *Communications of the ACM*, 36(7):86–97.

Part IV

Appendix

APPENDIX A

Euler Transform

The Euler transform is the multiplication of three rotation matrices, denoted $\mathbf{A}$, given by Equation A.1.

$$\mathbf{A} = \mathbf{BCD} \tag{A.1}$$

Where $\mathbf{A}$ is written as

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{A.2}$$

The three rotational matrices are given by Equations A.3, A.4, and A.5. B, C, and D represent the rotations in figure 6.1 a, b, and c.

$$\mathbf{B} = \begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & sin(\theta) \\ 0 & -sin(\theta) & cos(\theta) \end{bmatrix} \tag{A.4}$$

$$\mathbf{D} = \begin{bmatrix} cos(\phi) & sin(\phi) & 0 \\ -sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.5}$$

APPENDIX A. EULER TRANSFORM

APPENDIX B

Lambertian Shading

Source: [Slater et al., 2001]
This appendix contains a description of the Lambert model, that describes how light affects 3D surfaces.

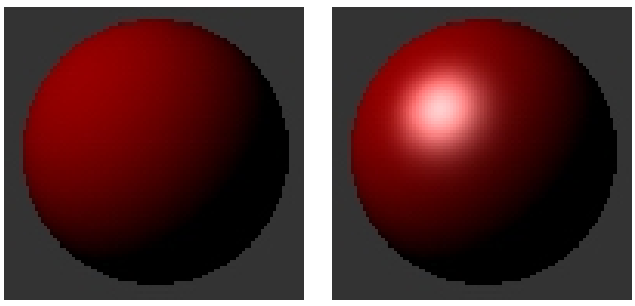
In computer graphics the ideal shading of surfaces is by utilising a global illumination model, to calculate all lighting in a scene. Global illumination calculates influences from all light sources in all directions at all points of a 3D scene.

Local illumination: Lambert's Reflection Model

A way to simplify illumination calculations is by only calculating the most significant influencing lights, as the peripheral lights has little effect on the illumination. This is called local illumination.

The Lambert Reflection Model is based on local illumination. The model has no base in reality, but is merely a simple way to obtain shading of 3D objects.

The shading of each 3D surface, may be divided into three categories. Diffuse reflection, which is the direct illumination of a surface, which makes the surface visible and specular reflection which controls how the surface shines on surfaces with direct light, see figures B.1a and B.1b. Ambient reflection controls the illumination of surfaces not affected by light.



(a) *(b)*

Figure B.1: (a)Diffuse reflection is the direct illumination of a surface, which makes the surface visible. (b) Specular reflection controls how the surface shines on surfaces with direct light.

To calculate the total light reflection in a scene the three reflections must be calculated for each point.

Diffuse reflection

Diffuse reflection may be calculated with the following equation. The factors are illustrated in figure B.2:

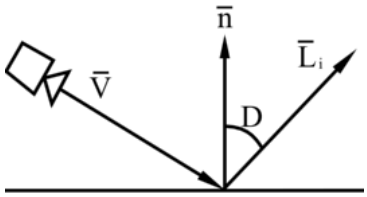


Figure B.2: The diffuse reflection of the Lambert Reflection Model.

$$I_{r,d} = k_d \cdot I_i \cdot \cos D \tag{B.1}$$

where:
$I_{r,d}$: The resulting intensity value for each edge voxel.
k_d: Diffuse reflection coefficient. One for each colour channel.
I_i: Intensity of the incoming light.
D: The angle between the normal vector and the incoming light vector.
$\vec{n}$: The normal vector to the surface.
$\vec{L_i}$: The vector for the incoming light.
$\vec{V}$: The viewing vector.

The diffuse calculations are performed for each colour channel denoted by k_d. The calculations are performed with normalised values, ranging between 0 and 1. With normalised vectors, $\cos D$ may be calculated with the following equation:

$$\cos D = \vec{n} \cdot \vec{L_i} \tag{B.2}$$

Specular reflection

Specular reflection may be calculated with the following equation. The factors are illustrated in figure B.3:

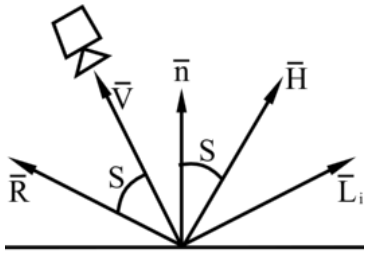


Figure B.3: The specular reflection of the Lambert Reflection Model.

$$I_{r,s} = k_s \cdot Ii \cdot (\cos S)^m \tag{B.3}$$

with:
k_s: Specular reflection coefficient.
I_i: Intensity of the incoming light.
S: The angle between the perfect reflection angle and the viewing angle.
m: Shininess-factor. Ranging from 1 and to values above. The higher the value, the more concentrated the specular reflection will be.
$\vec{L_i}$: The angle for the incoming light.
$\vec{H}$: The angle between the viewing angle and the incoming light angle.
$\vec{R}$: The perfect light reflection angle.

These are the same calculations for each colour channels.
The $\cos S$ may be replaced by:

$$\cos S = \vec{V} \bullet \vec{R} = \vec{n} \bullet \vec{H}$$

Where

$$\vec{H} = \vec{V} + \vec{L_i}$$

By applying the latter replacement the calculations may be simplified, as all factors $\vec{n}$, $\vec{V}$ and $\vec{L_i}$ are known.

Ambient reflection

The ambient reflection is the simplest of the three, as it does not incorporate angle calculations. Ambient is the light that affects all points.

The ambient reflection I_a may be calculated with:

$$I_{r,a} = k_a \cdot I_a \tag{B.4}$$

where:
k_a: Ambient reflection coefficient
I_b: Ambient light intensity

B.0.4 Total reflection

All factors adds to the total reflection like:

$$I_r = I_{r,a} + I_{r,d} + I_{r,s} \tag{B.5}$$

If there are several light sources within the scene, they will have to be calculated one at a time, and the partial results summed at the end. Light sources has no effect on the ambient reflection, hence this can be omitted of the illumination calculations.

$$I_r = k_a + I_a + \sum_{j=1}^{N}(k_d \cdot I_{i,j} \cdot (\vec{n} \bullet L_{i,j})) + \sum_{j=1}^{N}(k_s \cdot I_{i,j} \cdot (\vec{n} \bullet (\vec{V} + \vec{L_{i,j}}))^m) \tag{B.6}$$

This can be reduced to:

$$I_r = k_a + I_a + \sum_{j=1}^{N}(I_{i,j} \cdot (k_d \cdot (\vec{n} \bullet \vec{L_{i,j}}) + k_s \cdot (\vec{n} \bullet (\vec{V} + \vec{L_{i,j}}))^m)) \tag{B.7}$$

The summations ranges from 1 to N, with N denoting the number of light sources.

APPENDIX C

Intersection testing

Source: [Akenine-Möller and Haines, 2002]
When dealing with computer graphics, a basic technique to master is intersection testing. User-controlled picking, ray-tracing and shadow casting are but a few of the applications where intersection testing may be necessary. This section examines various methods for intersection testing in three dimensions. Throughout the various tests described in this section there will be a numerical imprecision, which is why a very small number will be used in the different tests. This number is denoted ϵ, and it's value varies from test to test.

Bounding Volume

When a scene full of objects needs to be tested for intersection, it will be very costly test for all faces on all objects for each ray sent into the scene. To this end a bounding volume can be applied to minimize intersection costs. The ray will then test the scene for intersection with the bounding volume, and the objects will only be tested if their bounding volume is intersected. The chance an arbitrary ray will hit an object is proportional to that object's surface area. If this area is minimized, the efficiency of an intersection algorithm is increased.

Sphere Bounding Volume

The simplest form of bounding volume, is to use a sphere bounding volume(SBV). To create a SBV the center of the object is found and the point in the object furthest away from the center is used as the radius for the SBV, which is on the usual Sphere formulae:

$$r^2 = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \tag{C.1}$$

Axis Aligned Bounding Box

Another simple type of bounding volume to create is an Axis Aligned Bounding Box(AABB). To form an AABB, the minimum and maximum extents of the set of polygons along each axis is taken, and used as the bounds for the AABB.

Ray/sphere Intersection

To test for an intersection between a ray and a sphere, both needs to be presented parametrically. The sphere is represented by equation C.2, and the line by equation C.3:

$$f(p) = ||\tilde{\mathbf{p}} - \tilde{\mathbf{c}}|| - r = 0 \tag{C.2}$$

With $\tilde{\mathbf{p}}$ being any point on the spheres surface, $\tilde{\mathbf{c}}$ the center point and r the radius.

$$\tilde{\mathbf{r}}(t) = \tilde{\mathbf{p_0}} + t \cdot \tilde{\mathbf{d}} \tag{C.3}$$

Where $\tilde{\mathbf{p_0}}$ is the lines starting point and $\tilde{\mathbf{d}}$ the direction vector. To solve the intersection problem these equations are combined, where $\tilde{\mathbf{r}}(t)$ replaces $\tilde{\mathbf{p}}$:

$$f(\tilde{\mathbf{r}}(t)) = ||\tilde{\mathbf{r}}(t) - \tilde{\mathbf{c}}|| - r = 0 \tag{C.4}$$

This may be rewritten to:

$$t^2 + 2t(\tilde{\mathbf{d}} \cdot (\tilde{\mathbf{p_0}} - \tilde{\mathbf{c}})) + (\tilde{\mathbf{p_0}} - \tilde{\mathbf{c}})^2 - r^2 = 0 \tag{C.5}$$

This is a second order equation, and may be solved with:

$$t^2 + 2tb + c = 0 \rightarrow t = -b \pm \sqrt{b^2 - c} \tag{C.6}$$

Where $b = \tilde{\mathbf{d}} \cdot (\tilde{\mathbf{p_0}} - \tilde{\mathbf{c}})$ and $c = (\tilde{\mathbf{p_0}} - \tilde{\mathbf{c}})^2 - r^2$

The test is: If $b^2 - c < 0$ then the ray misses the sphere, and test is rejected, and further calculations not needed.
solve the rest of equation C.6, and the smallest solution to t is the first intersection. An example is seen in figure C.1.

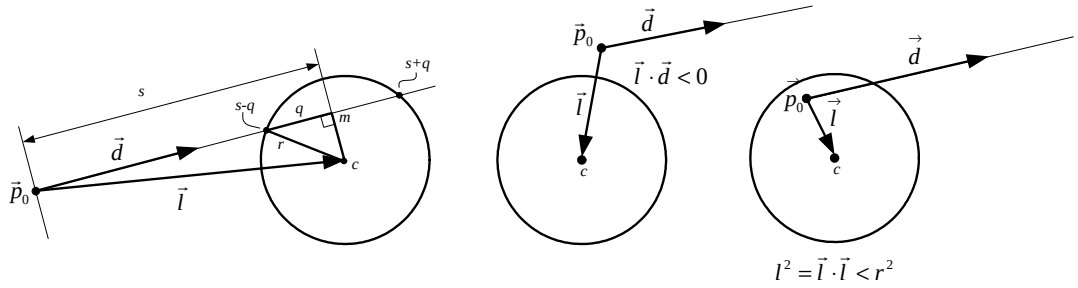


Figure C.1: Notation of the optimized ray/sphere intersection. Left figure: ray intersects sphere in two points with the distance is $t = s \pm q$ along the ray. Middle case is a rejection made when sphere is behind ray origin. Right case: Origin is inside sphere, ray always hits the sphere.

The pseudocode for an optimized ray/sphere intersection test is:

Ray/box Intersection

To intersect-test an AABB Woo's method can be applied. The idea is that given a ray and an AABB, denoted B, the three candidate planes out of the six composing the AABB must be identified. For each pair of parallel planes, the one backfacing the ray may be omitted from the further calculations. when the three planes are identified, the intersection distances, t-values, between the ray and the planes are found. The largest of these calculated distances is a possible hit, which can be seen in figure C.2. The distance is tested if it lies on the border of the box, if a hit is found the actual hit-point is calculated.

Pseudo code C.1 The intersection test between a ray and a triangle

RaySphereIntersect($\tilde{\mathbf{p_0}}, \tilde{\mathbf{d}}, \tilde{\mathbf{c}}, r$**)** :
$Returns(REJECT/INTERSECT)$
1: $\tilde{\mathbf{l}} = \tilde{\mathbf{c}} - \tilde{\mathbf{p_0}}$
3: $s = \tilde{\mathbf{l}} \cdot \tilde{\mathbf{d}}$
3: $l^2 = \tilde{\mathbf{l}} \cdot \tilde{\mathbf{l}}$
4: if($s < 0$ and $l^2 > r^2$) return ($REJECT$);
5: $m^2 = l^2 - s^2$
6: if($m^2 > r^2$) return ($REJECT$);
7: else return ($INTERSECT$)//

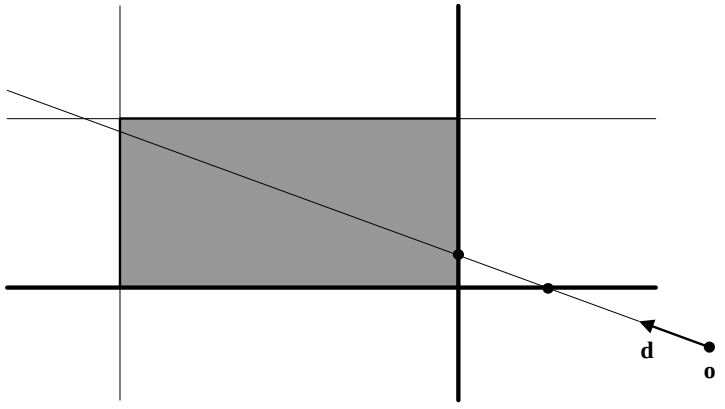


Figure C.2: Woo's method for calculating intersection between a ray and an AABB. The candidate planes are the front-facing marked with bold line, intersects the ray, and the intersect points are marked. The point farthest away from the origin, or the point with the biggest scalar on the direction vector is a possible intersection

Ray/triangle Intersection

One of the most applicable intersection tests is the ray/triangle, as all geometric objects consists of triangles. There exists many different ray/triangle intersection tests. The one described here reduces the calculations to 2D by transforming the triangle and ray into a system where the axis are aligned to the orientation of the triangle and the ray using Barycentric coordinates.

With the corners of a given triangle defined by $\tilde{\mathbf{v_0}}$, $\tilde{\mathbf{v_1}}$ and $\tilde{\mathbf{v_2}}$, a point $\mathbf{t}(u, v)$ on the triangle is given by the formula:

$$\mathbf{t}(u, v) = (1 - u - v)\tilde{\mathbf{v_0}} + u\tilde{\mathbf{v_1}} + v\tilde{\mathbf{v_2}} \tag{C.7}$$

(u, v) are the barycentric coordinates, which must fulfill the rule $u \geq 0$, $v \geq 0$ and $u + v \leq 1$. (u, v) can be used for texture mapping, normal interpolation, color interpolation etc. u and v represents the amount by which to weight each vertex's contribution to a particular location, where $w = (1 - u - v)$ is the third weight. To calculate the intersection between the ray $\tilde{\mathbf{r}}(t)$, and the triangle $\mathbf{t}(u, v)$, the two are set to equal each other: $\tilde{\mathbf{r}}(t) = \mathbf{t}(u, v)$, which is:

$$\tilde{\mathbf{p_0}} + t \cdot \tilde{\mathbf{d}} = (1 - u - v)\tilde{\mathbf{v_0}} + u\tilde{\mathbf{v_1}} + v\tilde{\mathbf{v_2}} \tag{C.8}$$

This term is rearranged to:

$$(-\tilde{\mathbf{d}} \quad \tilde{\mathbf{v_1}} - \tilde{\mathbf{v_0}} \quad \tilde{\mathbf{v_2}} - \tilde{\mathbf{v_0}}) \begin{pmatrix} t \\ u \\ v \end{pmatrix} = \tilde{\mathbf{p_0}} - \tilde{\mathbf{v_0}} \tag{C.9}$$

To find the barycentric coordinates (u, v) and the distance t from the ray origin to intersection point equation C.8 is solved. This solution may be viewed geometrically as translating the triangle to the origin, and transforming it to a unit triangle in y and z with the ray direction aligned with x. An example of this is seen in figure C.3.

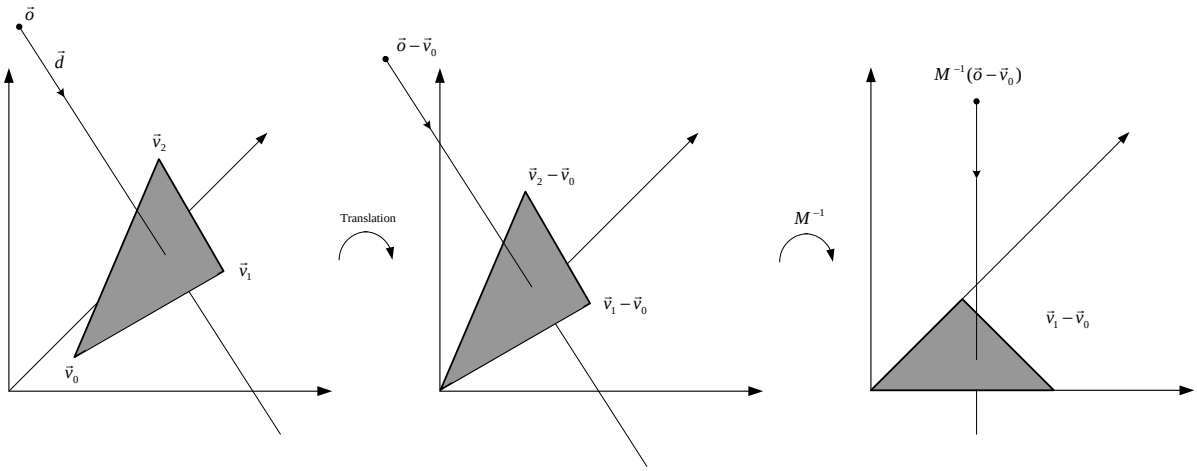


Figure C.3: Translation and change of base of the ray origin

$\mathbf{M} = (-\tilde{\mathbf{d}} \quad \tilde{\mathbf{v_1}} - \tilde{\mathbf{v_0}} \quad \tilde{\mathbf{v_2}} - \tilde{\mathbf{v_0}})$ is the matrix in equation C.8, and the solution is found by multiplying

the equation with $\mathbf{M^{-1}}$. Utilizing Cramer's rule, the following solution is obtained:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{det(-\tilde{\mathbf{d}}, \tilde{\mathbf{e_1}}, \tilde{\mathbf{e_2}})} \begin{pmatrix} det(\tilde{\mathbf{s}}, \tilde{\mathbf{e_1}}, \tilde{\mathbf{e_2}}) \\ det(-\tilde{\mathbf{d}}, \tilde{\mathbf{s}}, \tilde{\mathbf{e_2}}) \\ det(-\tilde{\mathbf{d}}, \tilde{\mathbf{e_1}}, \tilde{\mathbf{s}}) \end{pmatrix} \tag{C.10}$$

Where $\tilde{\mathbf{e_1}} = \tilde{\mathbf{v_1}} - \tilde{\mathbf{v_0}}$, $\tilde{\mathbf{e_2}} = \tilde{\mathbf{v_2}} - \tilde{\mathbf{v_0}}$ and $\tilde{\mathbf{s}} = \tilde{\mathbf{p_0}} - \tilde{\mathbf{v_0}}$.

Linear algebra yields that $det(\tilde{\mathbf{a}}, \tilde{\mathbf{b}}, \tilde{\mathbf{c}}) = |\tilde{\mathbf{a}} \quad \tilde{\mathbf{b}} \quad \tilde{\mathbf{c}}| = -(\tilde{\mathbf{a}} \times \tilde{\mathbf{c}}) \cdot \tilde{\mathbf{b}} = -(\tilde{\mathbf{c}} \times \tilde{\mathbf{b}}) \cdot \tilde{\mathbf{a}}$, and from this the solution may be rewritten as:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{(\tilde{\mathbf{d}} \times \tilde{\mathbf{e_2}}) \cdot \tilde{\mathbf{e_1}}} \begin{pmatrix} (\tilde{\mathbf{s}} \times \tilde{\mathbf{e_1}}) \cdot \tilde{\mathbf{e_2}} \\ (\tilde{\mathbf{d}} \times \tilde{\mathbf{e_2}}) \cdot \tilde{\mathbf{s}} \\ (\tilde{\mathbf{s}} \times \tilde{\mathbf{e_1}}) \cdot \tilde{\mathbf{d}} \end{pmatrix} = \frac{1}{\tilde{\mathbf{p}} \cdot \tilde{\mathbf{e_1}}} \begin{pmatrix} \tilde{\mathbf{q}} \cdot \tilde{\mathbf{e_2}} \\ \tilde{\mathbf{p}} \cdot \tilde{\mathbf{s}} \\ \tilde{\mathbf{q}} \cdot \tilde{\mathbf{d}} \end{pmatrix} \tag{C.11}$$

With $\tilde{\mathbf{p}} = \tilde{\mathbf{d}} \times \tilde{\mathbf{e_2}}$ and $\tilde{\mathbf{q}} = \tilde{\mathbf{s}} \times \tilde{\mathbf{e_1}}$, which are factors used to speed up the calculations.

The pseudo-code for such an operation would be:

Pseudo code C.2 The intersection test between a ray and a triangle

RayTriIntersect($\tilde{\mathbf{p_0}}, \tilde{\mathbf{d}}, \tilde{\mathbf{v_0}}, \tilde{\mathbf{v_1}}, \tilde{\mathbf{v_2}}$) : $Returns(REJECT/INTERSECT, u, v, t)$
1: $\tilde{\mathbf{e_1}} = \tilde{\mathbf{v_1}} - \tilde{\mathbf{v_0}}$
2: $\tilde{\mathbf{e_2}} = \tilde{\mathbf{v_2}} - \tilde{\mathbf{v_0}}$
3: $\tilde{\mathbf{p}} = \tilde{\mathbf{d}} \times \tilde{\mathbf{e_2}}$
4: $a = \tilde{\mathbf{e_1}} \cdot \tilde{\mathbf{p}}$
5: if($a > -\epsilon$ and $a < \epsilon$) return ($REJECT, 0, 0, 0$);
6: $f = \frac{1}{a}$
7: $\tilde{\mathbf{s}} = \tilde{\mathbf{p_0}} - \tilde{\mathbf{v_0}}$
8: $u = f(\tilde{\mathbf{s}} \cdot \tilde{\mathbf{p}})$
9: if($u < 0.0$ or $u > 1.0$) return ($REJECT, 0, 0, 0$);
10: $\tilde{\mathbf{q}} = \tilde{\mathbf{s}} \times \tilde{\mathbf{e_1}}$
11: $v = f(\tilde{\mathbf{d}} \cdot \tilde{\mathbf{q}})$
12: if($v < 0.0$ or $u + v > 1.0$) return ($REJECT, 0, 0, 0$);
13: $t = f(\tilde{\mathbf{e_2}} \cdot \tilde{\mathbf{q}})$
14: return ($INTERSECT, u, v, t$)

APPENDIX D

System Class Diagram