AccuRay:
A Physically Based Raytracer

Third Year Project Report

Sam Whelan
MEng Computer Science
School of Computer Science
University of Manchester

Supervised by Dr Steve Pettifer

April 2014

Abstract

This report covers the development of AccuRay: a Physically Based Raytracer. The main objective of the project is to accurately simulate how light interacts with objects in a scene, giving a realistic resultant image. Overall, this objective was generally achieved, having accurately simulated perfectly diffuse interactions, perfectly specular interactions, perfectly transmissive materials, and interactions with materials exhibiting a combination of these qualities. This project was created by Sam Whelan, and supervised by Dr Steve Pettifer.

Acknowledgements

I would like to take this opportunity to thank all of the support staff at the University of Manchester for providing a nourishing and enriching learning environment.

I would like to thank the teaching staff for all they've taught me, and for always being approachable and willing to help.

I would mostly like to thank my project supervisor, Dr Steve Pettifer for all his help and support throughout the year and always being reachable for any questions or advice despite his workload.

Thanks to all those who participated in the survey to evaluate the validity of this project.

I would also like to thank those who assisted in the rendering of the scenes used in this report: Tim Brier, Jordan Kirk, David Lack, Dragos Mihai, Samuel Ordoñez, Liam Pringle, Callum Richardson, Jenni Ward, Miso Zmiric.

Thanks to my sister, Beth Crisp for her assistance in proofreading this report.

Last, but by no means least, I would like to thank my parents for supporting me throughout my educational career.

Contents

Chapter 1

Introduction

This chapter gives a brief description of what the project entails, the main objectives and the ideal results. A list of milestones is included, as well as possible uses for the resultant images. This chapter concludes with an overview of the rest of the report.

1.1 Project Description

The main concept of the project is to create a program which produces realistic looking images from a 3D scene, based on physically based models of light transport. Ideally, this renderer should be able to accurately model a variety of key features including colour bleed, specular reflection and transparent objects. In order to produce realistic looking images, the renderer must include both direct illumination (from lights in the scene) and indirect illumination (from light bounced from other objects in the scene); so standard OpenGL would not suffice. AccuRay intends to provide a solution for these objectives through a process known as Monte Carlo Raytracing.

1.2 Possible Uses

As raytracing produces high quality, realistic looking images, its applications include high quality promotional material and computer generated special effects in TV and movies.

1.3 Milestones

1. Directly illuminated diffuse spheres

2. Soft shadows
3. Specular Reflection
4. Colour bleed
5. Mesh loading from file
6. Transmission
7. Image textures
8. Depth of Field

1.4 Report Overview

This report will include details of all aspects of the development process. Starting with a literature survey to give a more detailed idea of what the project entails, as well as an idea of other work done in the area of realistic computer graphics. Then going on to a discussion of the design of the project, running through the development process and justifying the technologies used. The implementation of each key feature will then be explained with diagrams explaining the overall concepts. In the next chapter, the results of each previously described feature will be demonstrated with images produced by the raytracer. A sample scene (Cornell Box) rendered with AccuRay is then compared to a the original Cornell Box in order to validate its accuracy, and also analysis of a survey comparing images produced with AccuRay and Blender's raytracer, Cycles. To conclude, there is a reflection of what was achieved, and a list of possible features that could be implemented to extend the project.

Chapter 2

Background

This chapter gives background information on the problem of realistic rendering. Expanding on this to describe two major algorithms and the different ways in which they approximate an accurate solution.

2.1 Realistic Rendering

In 1986, Kajiya[1] and Immel et al[2] simultaneously published an equation to accurately describe the lighting at a point on a surface, this equation has come to be known as the Rendering Equation.

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t)(\omega_i \cdot \mathbf{n}) \mathrm{d}\omega_i$$

Figure 2.1: The Rendering Equation

In words, the light at a point is the sum of the light emitted by the material at that point, and the integral over the light received from within a unit hemisphere of the point, multiplied by the cosine of the angle between the surface normal and the direction of incoming light, multiplied by the visibility of the light (so if a light is fully obstructed, it's contribution is not considered).

This equation can be applied for every sampled point in an image, with every light at every relevant point at time. Unfortunately, this equation is a complex multidimensional integral which can not be solved with brute force, only approximated using numerical methods.

All realistic renderers attempt to solve this equation in order to give realistic results.

2.2 Global Illumination Solutions

This section will describe and discuss two major global illumination algorithms, resulting in a comparison table featuring key components of each rendering solution.

There are two common ways in which the Rendering Equation has attempted to be solved using numerical methods, using the Finite Element Method[3] and using Monte Carlo Integration[4]. Both methods are physically based, with Radiosity's original application being to calculate heat energy transfer; and raytracing following the path of individual rays of light through a scene.

2.2.1 Radiosity

Radiosity uses the Finite Element Method in order to solve the Rendering Equation. The Finite Element Method[3] is a numerical technique which produces stable solutions by minimising error functions, solving smaller sub-problems in order to provide an overall solution.

Radiosity[5] considers light as energy that flows throughout a scene until an equilibrium is reached. First the scene is considered as a collection of polygonal 'patches' usually based on the meshes in the scene.

Radiosity[5] is a two step algorithm. After the patches have been defined, the first step involves calculating the 'form factor' between each pair of patches in the scene. The form factor is a measure of how visible two patches are two each other, considering relative sizes, orientations, shapes, and how occluded they are by other patches. In the second stage, the patches with the most light (which would initially be area lights) 'shoot' light into the scene, transferring light to all patches directly visible to them. This first pass should yield results similar to direct lighting setups. Then the second stage is repeated until an equilibrium is reached (a negligible amount of light is being transferred in an iteration), or the level of completion is deemed sufficient by the person running the program.

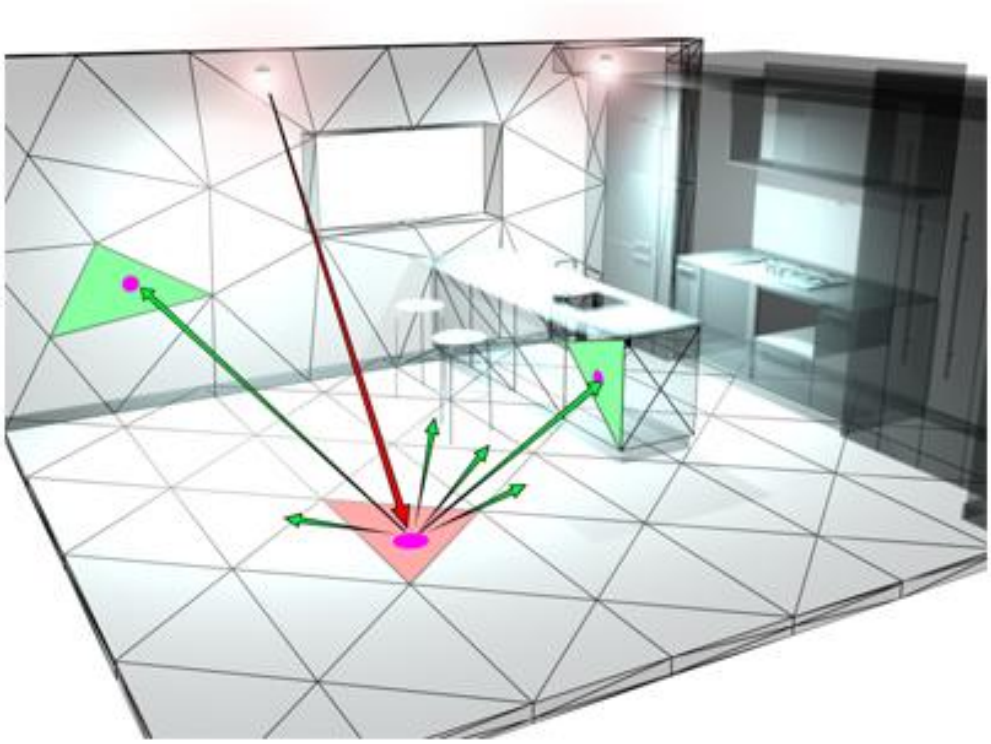


Figure 2.2: An image to show how the basemesh is split up into patches

Radiosity[5] assumes all surfaces are perfectly Lambertian (diffuse). As the algorithm does not consider an eye or camera position, it is known as 'view independent', and therefore cannot handle specular reflections or accurate transmission. However due to this view independent nature, it can be used to bake the lighting in a scene into an image texture known as a lightmap, which can be applied to the mesh and rendered in a raster graphics software using OpenGL in realtime. Due to this, radiosity is used in games, architectural walkthroughs, digital house viewings, and other interactive media.

2.2.2 Raytracing

The main concept of the project is to accurately simulate the interaction of light with a 3D scene. This can be achieved by a process called Raytracing[6] (also known as pathtracing), in which rays of light are traced as they bounce around a scene. In reality, rays of light are emitted from emissive objects, they bounce around a scene and a small portion of these rays intersect with the eye/camera. However, due to the large number of rays which do not intersect the eye/camera and therefore do not contribute to the resultant image, modelling the process in this way is incredibly wasteful of computational resources and therefore slow.

Instead, the process is modelled backwards, with rays originating from a virtual camera, firing through each pixel in an image plane positioned in the 3D world space and into the scene. If a ray intersects with an object, a 'shadow feeler' ray is sent out to each light in the scene to check whether the light is occluded and if the light's contribution is added to the image. The ray then bounces in a direction given by the material properties of the object and carries on the process recursively until a given depth is reached or the ray's contribution is negligible.

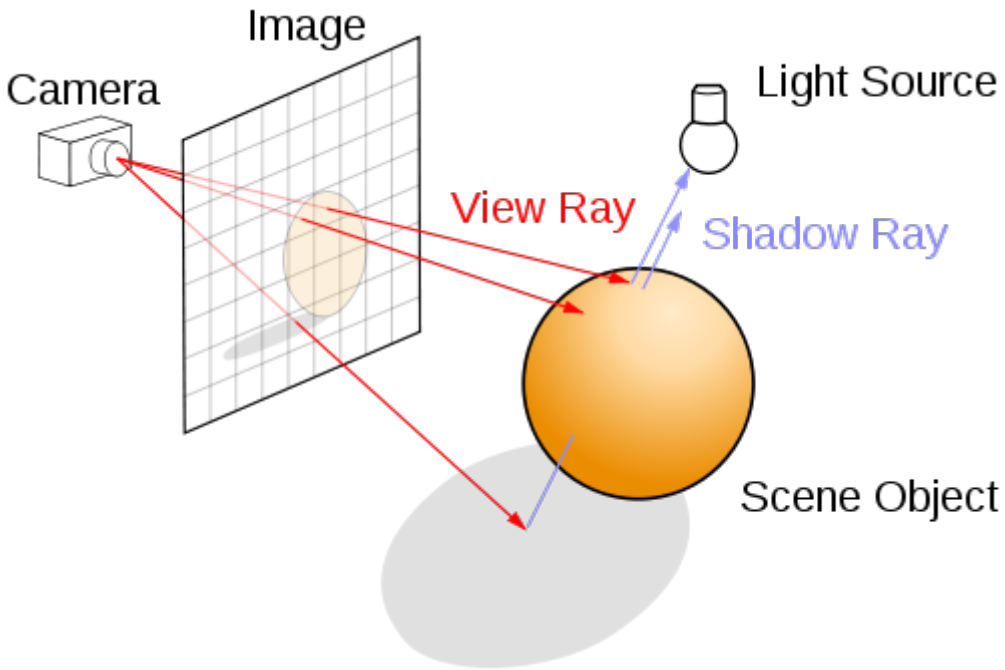


Figure 2.3: Diagram to illustrate the basic concepts of raytracing

Turner Whitted's original raytracing algorithm[7] involved firing one primary ray from the eye point through each pixel in the image grid into the scene. However using the Monte Carlo[8] method of taking multiple random samples within the pixel space converges on a much more accurate solution to the Rendering Equation.

2.2.3 Radiosity vs Ray Tracing

Table 2.1: Comparing Raytracing and Radiosity's suitability for this project

	Raytracing	Radiosity
Colour Bleed	Y	Y
Specular Reflection	Y	N
Soft Shadows	Y	Y
Mixing Diffuse and Specular	Y	N
Transmission	Y	N
Image Mapping	Y	Y
Depth of Field	Y	N
Normal Mapping	Y	Y
Visual Refinement	Y	Y
Works with unenclosed scenes	Y	N
TOTAL	10	5

Raytracing was chosen for this project due to Radiosity's inability to achieve accurate specular reflections and refraction - a requirement of the project.

Chapter 3

Design

This chapter begins with a list of the requirements of the project, both functional and non-functional. Then continues on to discuss the general attitude towards choosing accuracy over efficiency, the technologies used, and a description of the agile approach taken in the development of AccuRay.

3.1 Requirements

As this project is so specialised, it has a fairly narrow scope with regard to the software engineering aspects of design. In order to illustrate this point, a copy of AccuRay's use case diagram has been included below.

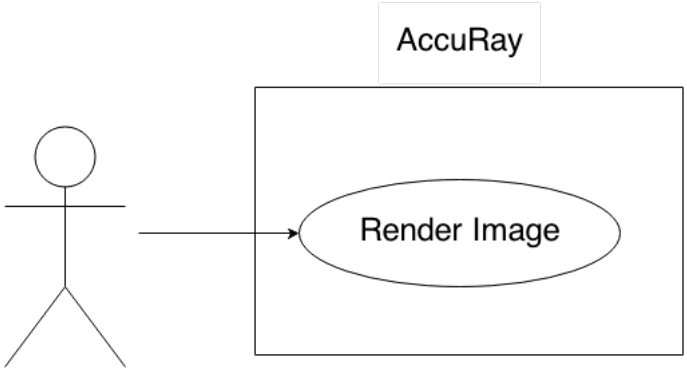


Figure 3.1: Use case diagram for the AccuRay project

Functional Requirements

Functional requirements are necessary features for the project to include. A list of functional requirements can be found below

AccuRay must be able to:

- Load object meshes from file
- Model perfect specular reflection
- Model perfect diffuse reflection
- Model transmission and reflection
- Provide some facility to mix specular and diffuse components

Non-Functional Requirements

As AccuRay is designed to be more of an academic exercise than a marketable product, there are no real non functional requirements. However ideally, a simple render with less than 1000 polygons should take less than a minute to render each sample.

3.2 Optimising the Rendering Equation

Unfortunately, even with Monte Carlo Raytracing, the rendering equation still has multidimensional variability, resulting in massive complexity. AccuRay applies a small number of simplifications to the Rendering Equation in order to make the computation more feasible.

3.2.1 Time

AccuRay only deals with rendering individual frames. This means that time is constant in each render, due to this time can be eliminated from the rendering equation.

3.2.2 Wavelength

While including wavelength would lead to the rendering of several visually interesting phenomena such as chromatic aberration and light dispersion, a wavelength independent equation is massively more computationally efficient. Not only that, but as wavelength is a continuous spectrum Rendering times would be unfeasibly large

3.3 Prototype

As this project is an academic exercise involving a large amount of complex mathematics and physics, a prototype was first made to acquire knowledge of the fundamental principles. In this objective, the prototype succeeded. The prototype was written in Python, chosen for its high level nature and its simplicity. Unfortunately due to the problems inherent to interpreted languages, the render times were too great to consider using Python for the final product.

The Prototype only rendered scenes consisting of spheres. This is because spheres have the simplest ray-object intersection calculation (both to compute and to understand). This simplicity is due to the fact that a sphere is defined parametrically by an origin position (centre of the sphere) and a radius.

3.4 Accuracy vs Efficiency

Many of the major design decisions involved choosing between accuracy and efficiency. One particular example of this is the Fresnel calculation for calculating how much light is reflected and how much is refracted. There is a commonly used approximation, known as Schlick's Approximation[**?**] used to speed up this expensive calculation, however this estimation is less precise and fails entirely when the ray travels from a more dense material to a less dense material. Due to this, the original, physically accurate Fresnel equations were used. Throughout the project, the more accurate method was generally chosen despite additional computational costs, in order to produce a more realistic image.

3.5 Technologies Used

3.5.1 Language

	Python	C	C++
Object Oriented	Y	N	Y
Fast to write	Y	N	N
Fast to execute	N	Y	Y
Support for GPU libraries	N	Y	Y
Good Documentation	Y	N	Y
Explicitly Typed	N	Y	Y

Table 3.1: Table of pros and cons for languages including Python, C and C++

In the end, C++ was chosen for it's object oriented nature and it's speed of execution. It was also considered for the possibility of implementing GPU Acceleration through either OpenCL or Nvidia's CUDA in future.

Technically, raytracing is possible with OpenGL using a Compute shader. However this would have added an additional layer of complexity and would have shifted the focus of the project away from learning about the processes involved in raytracing to tailoring the code to work with OpenGL.

3.6 Approach

The approach taken was a simplification of an agile approach known as Feature Driven Development[9]. It is a 5 stage sequence with the final 3 stages repeated each iteration to develop a single feature. FDD is originally designed for a team, so adjustments were made to streamline the process for a one-man team.

Stage 1, known as 'Develop Overall Model', is a planning phase used to understand the domain of the problem.

Stage 2, known as 'Build Feature List', involves building a list of features the final product should have, these were stated above as functional requirements.

Stage 3, known as 'Plan by Feature', involves using the list of features to form a development plan detailing how to achieve the implementation of the next feature.

Stage 4, known as 'Design by Feature', involves reassessing the design based on the project as it currently stands. Traditionally this involves creating a 'design package' in order to communicate changes to all members of the development team, however this was simplified due to development by a sole programmer.

Stage 5, known as 'Build by Feature', is self explanatory, the code is written to add the feature's functionality. Once validated, this code is committed to the main build.

This approach allowed the focus to be on adding fully functional features and always having a usable solution. The short, focused iterations were also good for morale, as they showed progress was consistently being made.

Chapter 4

Implementation

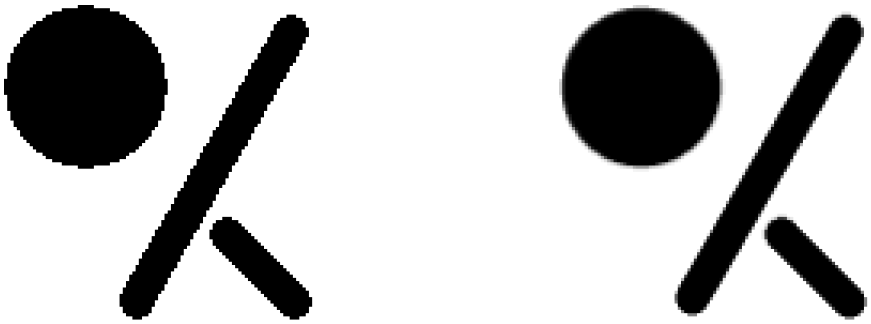
This chapter covers the implementation details of the project, explaining each of the key features using diagrams and describing how they were implemented in AccuRay. This chapter also covers optimisations made by AccuRay in order to reduce rendering times.

4.1 Key Features

This section covers the key features implemented in the raytracer with a detailed explanation of how they were implemented.

4.1.1 Anti Aliasing

Due to the inherent nature of Monte Carlo Raytracing, a number of samples are taken per pixel. This reduces jagged artefacts at object boundaries known as aliasing. An example can be seen below.



(a) 5x zoomed image with aliasing (b) 5x zoomed image with anti-aliasing

Figure 4.1: Example of the visible impact of aliasing

When a scene defined in a continuous environment is discretised, there is always a loss of accuracy, with an insufficient sample space (small image resolution) or when the distance between samples is too large (eg only one sample per pixel), a problem known as aliasing can occur. The effect of aliasing is more apparent on small images, with aliasing being barely visible on higher resolution images. Aliasing is an entirely digital problem and does not occur through natural ocular vision, and so much be minimised in order to create realistic images.
AccuRay achieves this by firing a large number of rays for each pixel, reducing the distance between samples and converging towards an accurate solution.

4.1.2 Phong Shading

All meshes were defined as a list of triangles, thus giving the model a faceted appearance. Phong shading[10] is a way of providing a smooth continuous surface defined by the triangle's vertices. Phong shading interpolates between the vertex normals to give the appearance of a smooth curve between points. As it does not add geometry, the silhouette of the model is unchanged. Phong shading does not work for non-curved meshes with large angles between vertices, eg cubes, where flat shading is best. Other than these problems, phong shading provides a smooth appearance to a mesh, increasing the realism.

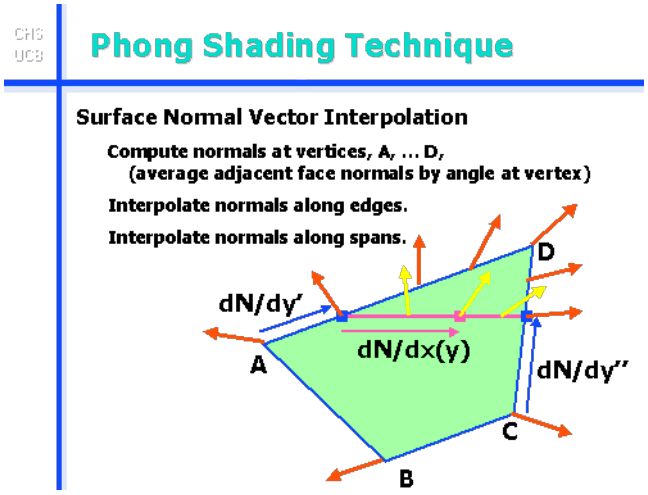
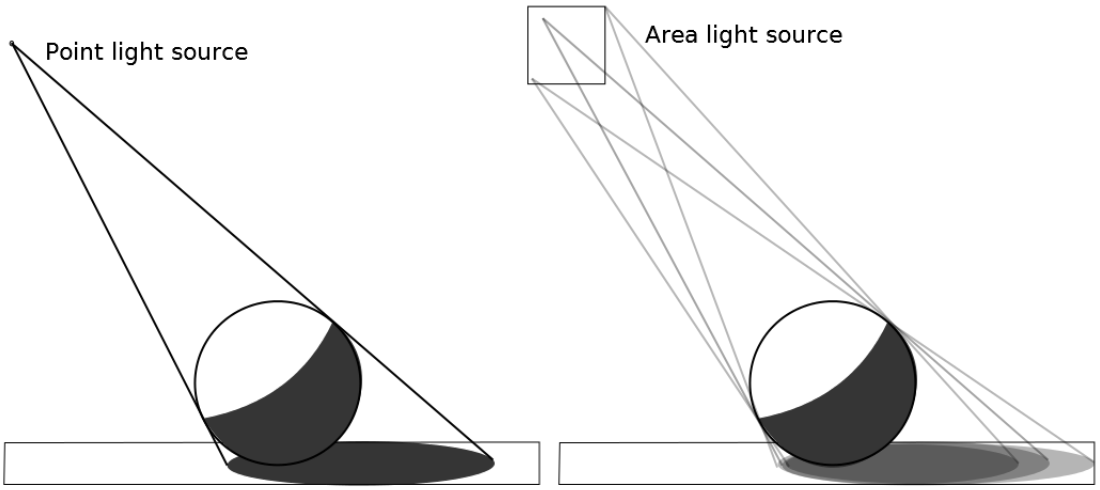


Figure 4.2: How Phong Shading Works

4.1.3 Soft Shadows

Shadows are an inherent feature of raytracing as whenever a ray intersects with an object, it's local colour is only considered if there is an uninterrupted path to

a light. Shadows form naturally in areas where little or no light can reach. Most simple raytracers and other renderers in general use point light sources in which all of the light energy originates from a single point in 3D space. These point lights give stark shadows with hard edges as can be seen in the diagram below. Of course these point lights do not exist in reality and are therefore out of place in a physically accurate simulation. This raytracer uses area lights, physical objects which occupy space in the 3D world and feature emissive material properties.



(a) Shadows formed from a point light (b) Shadows formed from an area light

Figure 4.3: Comparison of shadows formed from point light sources and area light sources

Hard shadow boundaries form as light from a single point is projected against an object, resulting in a single defined silhouette. There are several ways to fake more accurate soft shadowing, for example by just blurring the shadow shape, however the key to modelling realistic shadows is to use realistic lighting. A similar principle of achieving shadows applies to area lights, however instead of the light originating from a single point, the light originates from numerous random points on the light's surface, which over a large number of samples, converge together in order to give accurate soft shadowing.

4.1.4 Diffuse Interreflection

Diffuse Interreflection, also known as colour bleed is the modelling of the light rays interacting between diffuse surfaces.

First the basic diffuse lighting is considered, where the light received is determined by the angle between the surface normal and the incident light ray, this is known as dot product lighting as the dot product of those vectors gives the cosine of the angle between them. A surface parallel to the light will receive full illumination from the light, whereas a surface perpendicular to the light will receive none.

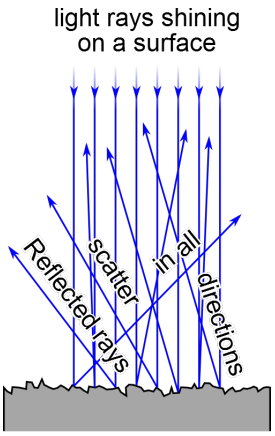


Figure 4.4: How Diffuse Interreflection works

In order to model light interactions with other surfaces and therefore provide global illumination, a ray is fired with a random direction within a unit hemisphere of the surface normal. As the algorithm progresses and more samples are taken, this converges to give the appearance of a perfectly matte surface.

The following sections will involve a large amount of vector maths, so a diagram has been provided below to aid understanding.

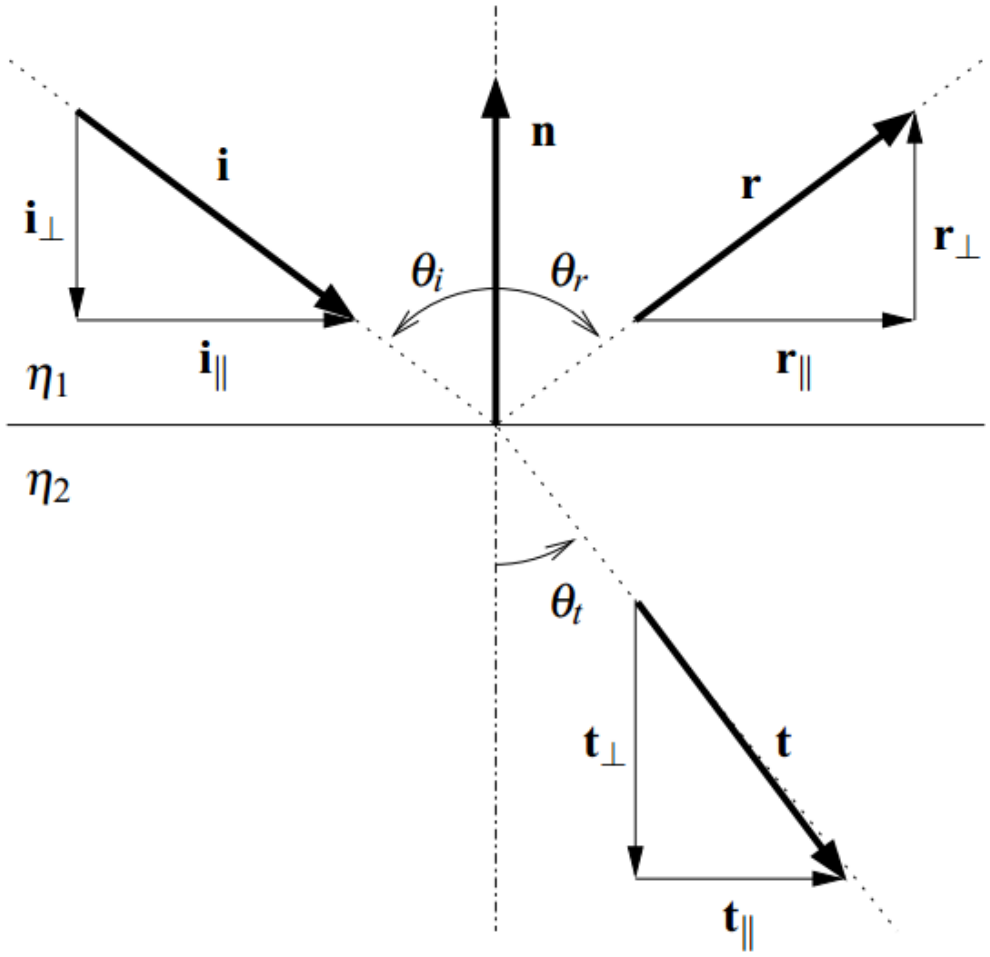


Figure 4.5: A diagram of vectors for reflection and transmission

4.1.5 Specular Reflection

Specular reflection is a property inherit to raytracing, as when a ray intersects with a specular material, it bounces off and continues to trace the ray (assuming the maximum depth is not reached). The simplest form of specular reflection occurs in purely specular surfaces e.g. mirrors. When a ray intersects with such a surface the direction of the ray is reflected in the surface normal. The vector direction of the reflected ray can be calculated using the following formula

$$\mathbf{r} = \mathbf{i}_{\parallel} - \mathbf{i}_{\perp} \tag{4.1}$$
$$= [\mathbf{i} - (\mathbf{i} \cdot \mathbf{n})\mathbf{n}] - (\mathbf{i} \cdot \mathbf{n})\mathbf{n}) \tag{4.2}$$
$$= \mathbf{i} - 2(\mathbf{i} \cdot \mathbf{n})\mathbf{n} \tag{4.3}$$

Figure 4.6: Calculating Reflection Direction

4.1.6 Transmission

One major feature of the raytracer is transmission rays which allow transparent effects.
A generic vector $\mathbf{v}$ (which in this case would be wither $\mathbf{i}$, $\mathbf{r}$, or $\mathbf{t}$); can be split up into parallel and vertical components through the following formulae[11]:

$$\mathbf{v}_{\perp} = \frac{\mathbf{v} \cdot \mathbf{n}}{|\mathbf{n}|^2}\mathbf{n} = (\mathbf{v} \cdot \mathbf{n})\mathbf{n}$$

Figure 4.7: Perpendicular Component

$$\mathbf{v}_{\parallel} = \mathbf{v} - \mathbf{v}_{\perp}$$

Figure 4.8: Parallel Component

Calculating Transmission Direction

According to Snell's Law, amount of divergence on the incident ray direction is dependant on the ratio of the material's index of refraction. However if the incident angle is greater than the inverse of the ratio of the refractive indices this results in $\sin\theta_t$ with a value greater than 1, which is impossible. At this point, Total Internal Reflection (TIR) occurs, which is discussed later in the report. To avoid TIR, the following check is put in place:

$$sin\theta_t = \frac{\eta_1}{\eta_2}\sin\theta_i \Leftrightarrow \sin\theta_i \leq \frac{\eta_2}{\eta_1}$$

In order to calculate t, it must first be split up into its component parts

$$\mathbf{t} = \mathbf{t}_{\parallel} + \mathbf{t}_{\perp}$$

$\mathbf{t}_{\parallel}$ is calculated as:

$$\mathbf{t}_{\parallel} = \frac{\eta_1}{\eta_2}\mathbf{i}_{\parallel} = \frac{\eta_1}{\eta_2}[\mathbf{i} + (\mathbf{i} \cdot \mathbf{n})\mathbf{n}]$$

$\mathbf{t}_{\perp}$ is calculated as:

$$\mathbf{t}_{\perp} = -\sqrt{1 - |\mathbf{t}_{\parallel}|^2}\mathbf{n}$$

Combining and expanding the two component vectors which makes up $\mathbf{t}$ gives the following formula:

$$\mathbf{t} = \frac{\eta_1}{\eta_2}\mathbf{i} + \left(\frac{\eta_1}{\eta_2}\cos\theta_i - \sqrt{1 - \sin^2\theta_t}\right)\mathbf{n}$$

where $\sin^2\theta_t$ is given by Snell's Law as:

$$\sin^2\theta_t = \left(\frac{\eta_1}{\eta_2}\right)^2 \sin^2\theta_i = \left(\frac{\eta_1}{\eta_2}\right)^2 (1 - (\mathbf{i} \cdot \mathbf{n})^2)$$

Balancing Reflection and Transmission

In order to maintain principles of conservation of energy, the sum of the co-efficients of transmission and reflections must equal 1.
As light can be considered as an electromagnetic wave with an electric field, it's behaviour can be described with Fresnel Equations.
The Reflectance coefficient for s-polarised light is given by:

$$R_{\perp}(\theta_i) = \left(\frac{\eta_1\cos\theta_i - \eta_2\cos\theta_t}{\eta_1\cos\theta_i + \eta_2\cos\theta_t}\right)^2$$

The Reflectance coefficient for p-polarised light is given by:

$$R_{\parallel}(\theta_i) = \left(\frac{\eta_2\cos\theta_i - \eta_1\cos\theta_t}{\eta_2\cos\theta_i + \eta_1\cos\theta_t}\right)^2$$

These equations are physically based and call for polarised light. However in reality, light is generally unpolarised, so this can be approximated by averaging the s-polarised and p-polarised light reflected.

$$R(\theta_i) = \frac{R_{\perp}(\theta_i) + R_{\parallel}(\theta_i)}{2} \Leftrightarrow \neg TIR$$

If Total Internal Reflection occurs, there is full reflection and no transmission.

$$R(\theta_i) = 1 \Leftrightarrow TIR$$

In order to conserve energy, the transmission co-efficient is given by:

$$T(\theta_i) = 1 - R(\theta_i)$$

Total Internal Reflection

If the angle of the incident ray with the surface normal is greater than a critical angle dependent on the material's index of refraction, then a phenomena known as total internal reflection occurs. Total Internal Reflection involves the light ray reflecting inside the object rather than transmitting back outside the object. This is the property that gives different cuts of diamonds their different visual properties.

4.1.7 Roughness

In reality, not all materials are completely diffuse, specular or transmissive. To enable a mixture of these components, a roughness material property was implemented. A rough material has it's normal perturbed by a given amount within a unit hemisphere, with a rougher material appearing more matte (and therefore diffuse). A rough transmissive material has the appearance of frosted glass, while a rough specular material looks like slightly distressed metal or plastic.

4.1.8 Image Textures

One way to add realistic colour to an object is to map an image texture onto it. These textures can be taken from photographs in order to give photo-realistic results. For this, the .obj file for the given object mesh must include texture co-ordinates for each vertex. If a model did not come supplied with these, they were generated automatically using Blender.

Texture Space

Each vertex of the object mesh also includes texture co-ordinates (also known as UV co-ordinates in order to distinguish texture space from world space). These map the vertices of the model to a pixel co-ordinate on the texture image. UV co-ordinates are in the range $\{0 \ldots 1\}$ so there is no dependence on a specific texture size. As previously stated, these texture co-ordinates give the pixel values for the vertices in the model, but in raytracing, an intersection can occur at any point on any face of a model, so a way of interpolating between the three vertices on the face is required.

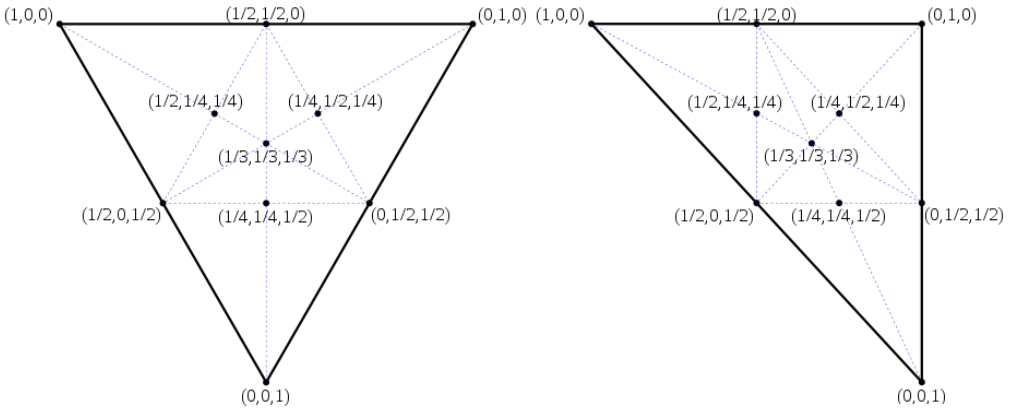


Figure 4.9: Triangles defined by Barycentric Co-ordinates

This was done by using barycentric co-ordinates, which can be see illustrated above. The midpoint is defined as the barycenter, with the vertices being defined by the distance from the barycenter and any point on the surface being described as a weighted sum of the vertex co-ordinates. This weighted sum method is used to calculate the UV co-ordinates for the intersection point.

Sampling Strategies

Once the UV co-ordinates for an intersection have been calculated, they must next be mapped to a pixel value from the texture. However the UV co-ordinate values are rarely integers, and so an image sampling strategy must be implemented to give an estimated colour value from the fin image.
After testing numerous sampling strategies, starting with nearest neighbour, bilinear filtering was chosen.
Bilinear filtering interpolates the texture co-ordinate of the intersection point based on its distance to each vertex in the triangle and their texture co-ordinates. If the co-ordinates are not integers, the texture colour is averaged between adjacent pixels to give a smooth blending effect.

4.1.9 Normal Mapping

With the texture co-ordinates already calculated and sampling strategies implemented, normal mapping was a simple addition to the raytracer. Normal mapping is a way to simulate a high level of detail on a low polygon mesh. Normal mapping does not change the geometry of the mesh, but instead perturbs the surface normals, changing the way the surface interacts with light, giving the impression of a higher detailed mesh.

All of the lighting calculations depend on the surface normal, particularly the angle between the surface normal and the vector direction of the incident ray. For example, direct diffuse illumination gives brightest results when the angle is 0 (the surface is facing the light), and darker results as the surface faces away. Flat shaded surfaces have uniform normals across each triangle, whereas Phong shaded surfaces have interpolated normals to give a smooth effect. A large normal map can alter the normals across multiple points within a triangle, giving the impression of extra geometry at little computational expense.

4.1.10 Depth of Field

Depth of Field was implementing by accurately simulating how a camera or an eye sees the world. Without this feature the raytracer uses a simple pinhole camera model in which rays originate from a single point and travel from that point through each pixel into the scene. A more complicated camera model is required to simulate depth of field, requiring two key components: the focal length of the camera and the aperture size. The focal length gives the distance to the focal plane and the aperture size gives the strength of the focus blur. This feature is key in adding a sense of realism; in reality, not everything can be in focus, so images without depth of field appear artificial. Unfortunately, it isn't possible to trace rays from every single point on the aperture, so using Monte Carlo methods, a random point on the aperture is chosen for each sample. Over a large number of samples, the average of these samples converges towards total coverage of the aperture.

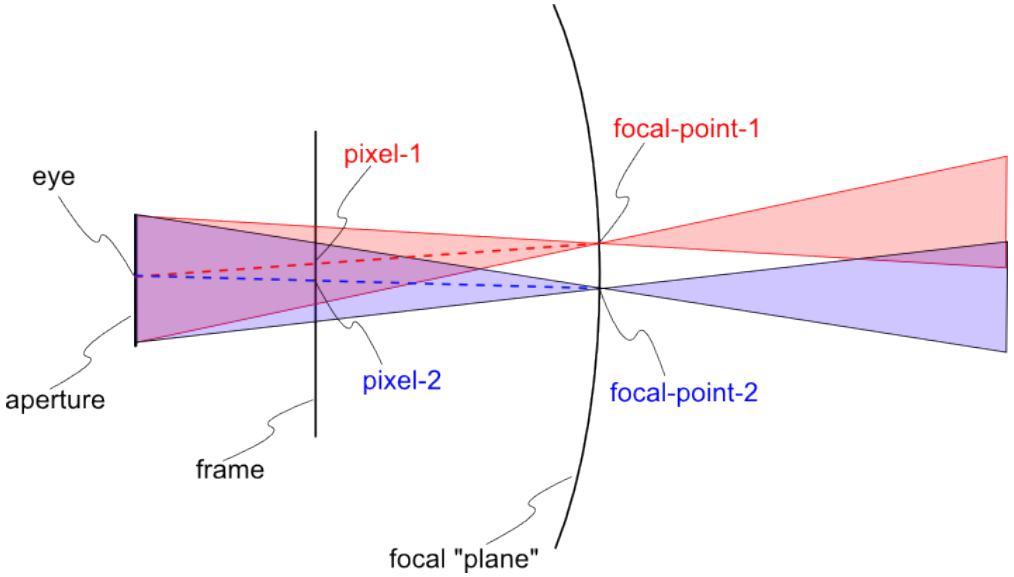


Figure 4.10: An aperture camera exhibiting depth of field

The Depth of Field effect is achieved by firing a ray from the original eye-point through the pixel until the focal distance is reached. This co-ordinate is then recorded as the focal point for this pixel, and a ray is sent from a random point on the camera surface through that focal point into the scene, continuing the raytracing process as before.

4.1.11 Arbitrary Model Loading

AccuRay supports loading arbitrary models from files in the '.obj' file format. Models are defined by a triangular mesh which is stored in AccuRay as a list of primitive Triangle objects. Models must also include UV texture co-ordinates for each vertex and vertex normals. If one of these is missing, AccuRay throws an error describing the problem and exits.

Ray-Triangle Intersection

In order to render this mesh, AccuRay must know how to interact a ray with a triangle. AccuRay uses the Möller–Trumbore Intersection Algorithm[12], a fast ray-triangle intersection algorithm developed in 1997, which can be found described below.

Pseudocode 4.1: Möller–Trumbore Intersection Algorithm

```
#define EPSILON 0.000001

int triangle_intersection( const Vec3   V1,  // Triangle vertices
                           const Vec3   V2,
                           const Vec3   V3,
                           const Vec3    O,  //Ray origin
                           const Vec3    D,  //Ray direction
                                 float* out )
{
  Vec3 e1, e2;  //Edge1, Edge2
  Vec3 P, Q, T;
  float det, inv_det, u, v;
  float t;

  //Find vectors for two edges sharing V1
  SUB(e1, V2, V1);
  SUB(e2, V3, V1);
  //Begin calculating determinant - also used to calculate u parameter
  CROSS(P, D, e2);
  //if determinant is near zero, ray lies in plane of triangle
  det = DOT(e1, P);
  //NOT CULLING
  if(det > -EPSILON && det < EPSILON) return 0;
  inv_det = 1.f / det;

  //calculate distance from V1 to ray origin
  SUB(T, O, V1);

  //Calculate u parameter and test bound
  u = DOT(T, P) * inv_det;
  //The intersection lies outside of the triangle
  if(u < 0.f || u > 1.f) return 0;

  //Prepare to test v parameter
  CROSS(Q, T, e1);

  //Calculate V parameter and test bound
  v = DOT(D, Q) * inv_det;
  //The intersection lies outside of the triangle
  if(v < 0.f || u + v  > 1.f) return 0;

  t = DOT(e2, Q) * inv_det;

  if(t > EPSILON) { //ray intersection
    *out = t;
    return 1;
  }

  // No hit, no win
  return 0;
}
```

4.1.12 Object-World Space Mapping

All loaded objects are considered to have their own objects space, however when inserting these objects into world space, they will all be centred at the origin and therefore may occlude each other, unless they are mapped to world space. World Space mapping was implemented by adding simple affine transformations so the objects, changing the position and scale of the objects.

4.2 Optimisations

4.2.1 Code Optimisations

There are several code specific optimisations used to speed up AccuRay. All objects and lists are passed by pointers rather than value to save time copying large amounts of memory every time a function is called. Simple functions are inlined to reduce function call overheads. Also, the code was compiled with with [-03] flag to optimise fully for speed, giving up to a 5x speed increase at no additional cost.

4.2.2 Adaptive Ray Depth

Different material properties require different number of rays. Transparent materials require the most due to total internal reflection (depending on the model, a ray of light could bounce around an object infinitely). Diffuse inter-reflection (colour bleed) requires the least, and specular materials require something in between. Rather than recursing each material to the full recursion depth, a primary ray starts with a maximum depth eg 20; a transmissive material reduces the depth by one, a specular material reduces the depth by two, and a diffuse material reduces the depth by 4. If the ray depth is less than zero, that ray terminates and its contribution is added to the image.

4.2.3 Bounding Volume

The most computationally expensive part of raytracing is ray-object intersection calculations. Reducing the number of unnecessary ray-intersection calculations gives a massive boost to rendering speed. In order to achieve this, bounding volumes have been introduced. These bounding volumes can be imported as object meshes or hand-coded as a list of triangles defined by their vertices. In general terms, a bounding volume is a low polygon mesh which envelops a high poly object in the scene. Ray intersections are only calculated with the high poly mesh if the ray intersects with the bounding volume. This massively speeds up

the rendering of areas outside of the bounding volume as they are not being tested for intersections with each triangle in the high poly mesh. The exact speed-up depends entirely on the resolution of the high poly mesh, the proportion of the image taken up by the high poly mesh, how well the bounding volume fits the high poly mesh and the maximum ray depth.

Chapter 5

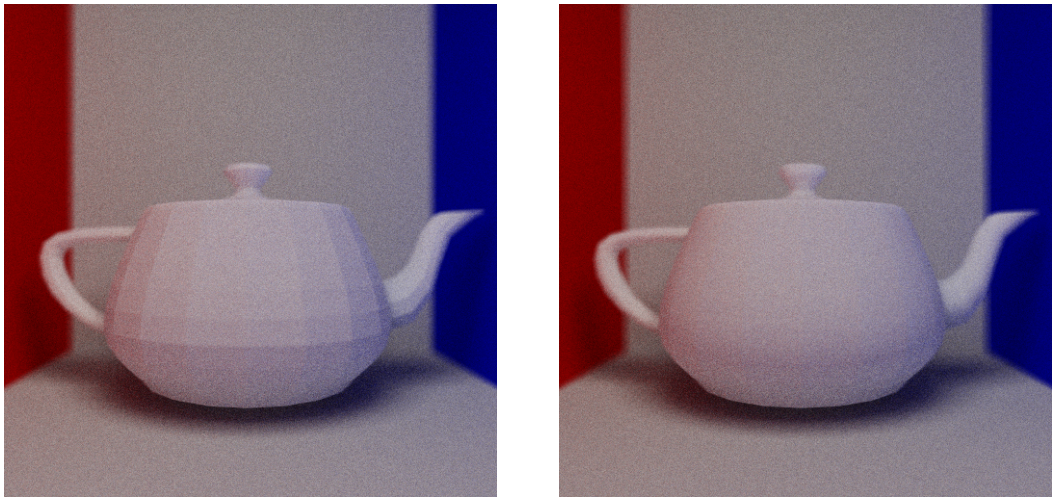
Results

This chapter illustrates the results of the features implemented in AccuRay with descriptions of the effects achieved. This chapter also discusses problems with AccuRay as well as methods to improve them in future.

5.1 Achievements

The following section shows the results achieved with AccuRay testing the main features previously outlined.

5.1.1 Phong Shading



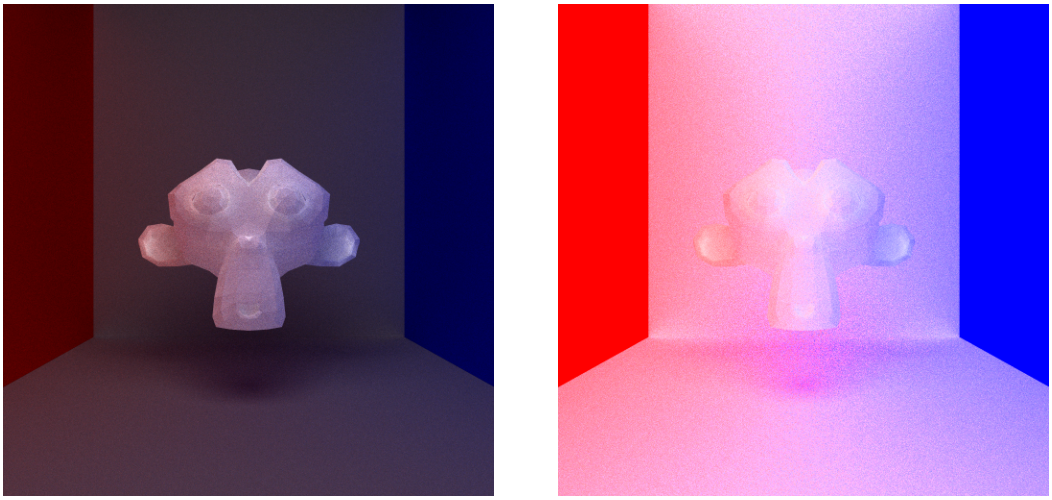
(a) Flat Shaded Teapot (b) Phong Shaded Teapot

Figure 5.1: Comparison of Flat Shading and Phong Shading

Image (b) appears substantially smoother and more realistic, even though both use the same geometry, the only difference being the normals used in the lighting calculations. Note that there is still a jagged silhouette on the spout and at the base of the teapot.

5.1.2 Diffuse Interreflection

A colour bleeding effect can be observed on the floor near the white wall. Also note how the shadows are not black, but are influenced by the light bounced from nearby diffuse surfaces.



(a) AccuRay render showing colour bleeding

(b) Colour Isolated Render to show colour bleeding

Figure 5.2: Demonstration of Colour Bleeding (Diffuse-Interreflection)

To clarify, the back wall and floor are grey diffuse surfaces, and the monkey model is a white diffuse model and the light is a white emissive surface. All colour on these objects is a result of the light bouncing around the diffuse surfaces in the scene. This effect is particularly noticeable in shadows as there is less direct light in these areas.

5.1.3 Specular Reflection

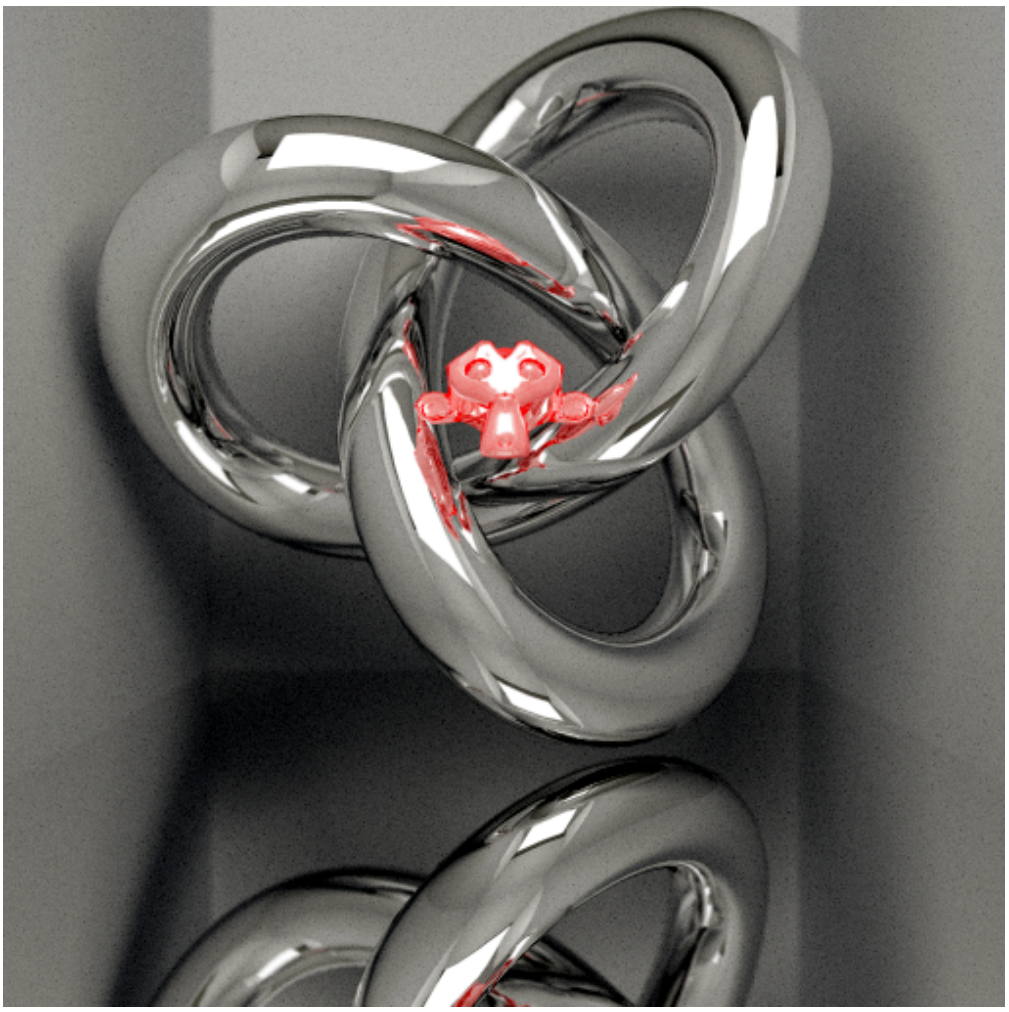


Figure 5.3: Perfect Specular Reflection including Area Light

The image above shows perfect specular reflection rendered with AccuRay. In order to make the reflections more distinguishable, a glossy red object was reflected in a mirror material. Note the multiple reflections achieved from a large ray depth which are distorted realistically based on the angle between the surface normal and the origin of the reflection. The area light is also reflected, giving a sharp highlight. Despite the torus knot being a fairly low poly mesh, the application of Phong shading gives a very smooth appearance.

5.1.4 Rough Specular

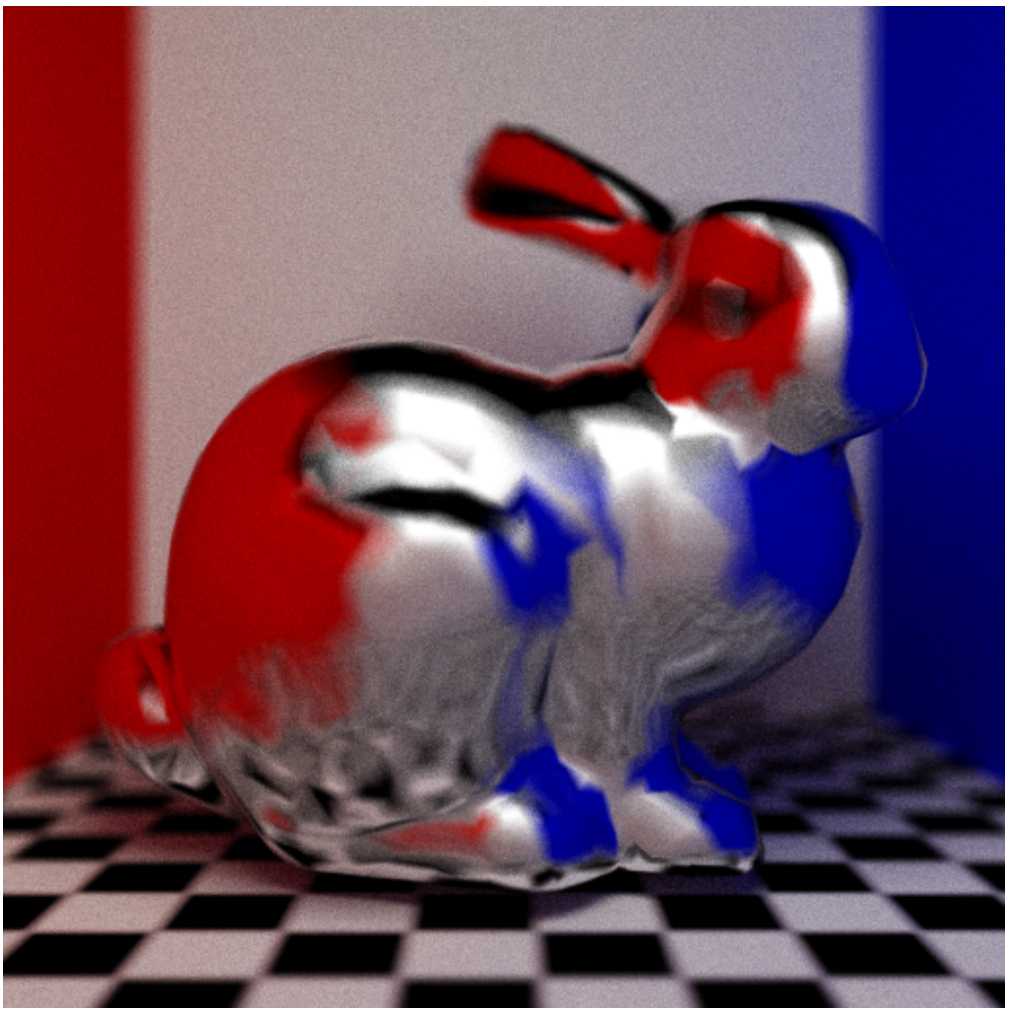
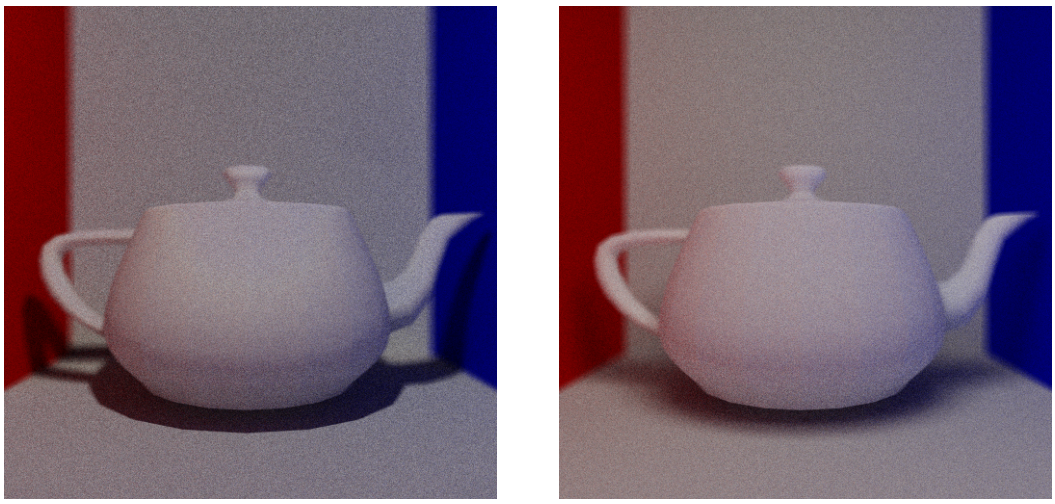


Figure 5.4: Rough Specular Stanford Bunny

Note the lack of sharpness in highlights (any sharpness is a result of using a low resolution mesh). Rough specular materials still reflect images textures, although they also result in blurry reflections. The degree of blurriness, or rather the radius of the blur depends on the roughness of the material, with 0 being perfectly specular and 1 being half specular, half diffuse. An increased ratio of diffused shading can be achieved by setting the specularity to a low value with a high roughness value.

5.1.5 Area lights

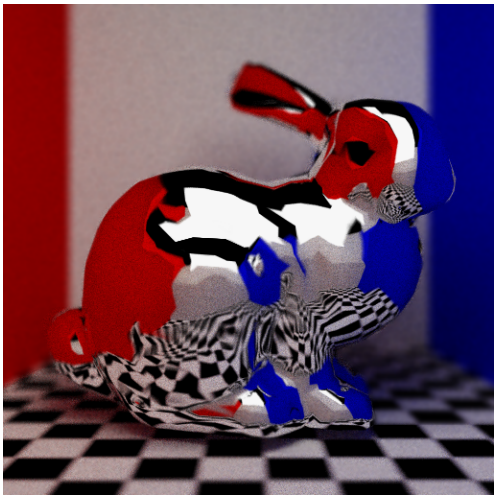


(a) Teapot lit by point light (b) Teapot lit by area light

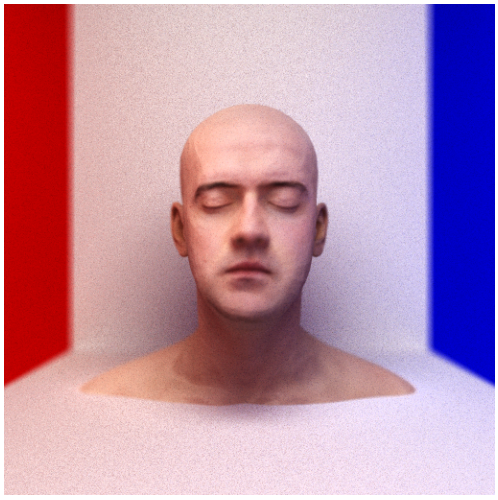
Figure 5.5: Comparison of shadows formed from point lights and area lights

It must be noted that the blurring of the shadow in the first image is only due to the effect of depth of field. Also note that despite the use of Phong shading on the teapot, the sharp edges of the shadow formed from the point light are still apparent.

5.1.6 Image Textures



(a) Specular bunny with textured floor



(b) Diffuse image textured head model

Figure 5.6: Demonstration of image textures in AccuRay

Image textures are just mapped as diffuse colours, and so still exhibit diffuse properties seen elsewhere in AccuRay. Image (a) shows that reflections of an image texture can be warped by the shape of the specular object. Image (b) shows that objects with image textures can still have diffuse inter-reflection and can therefore exhibit colour bleeding.

5.1.7 Normal Mapping

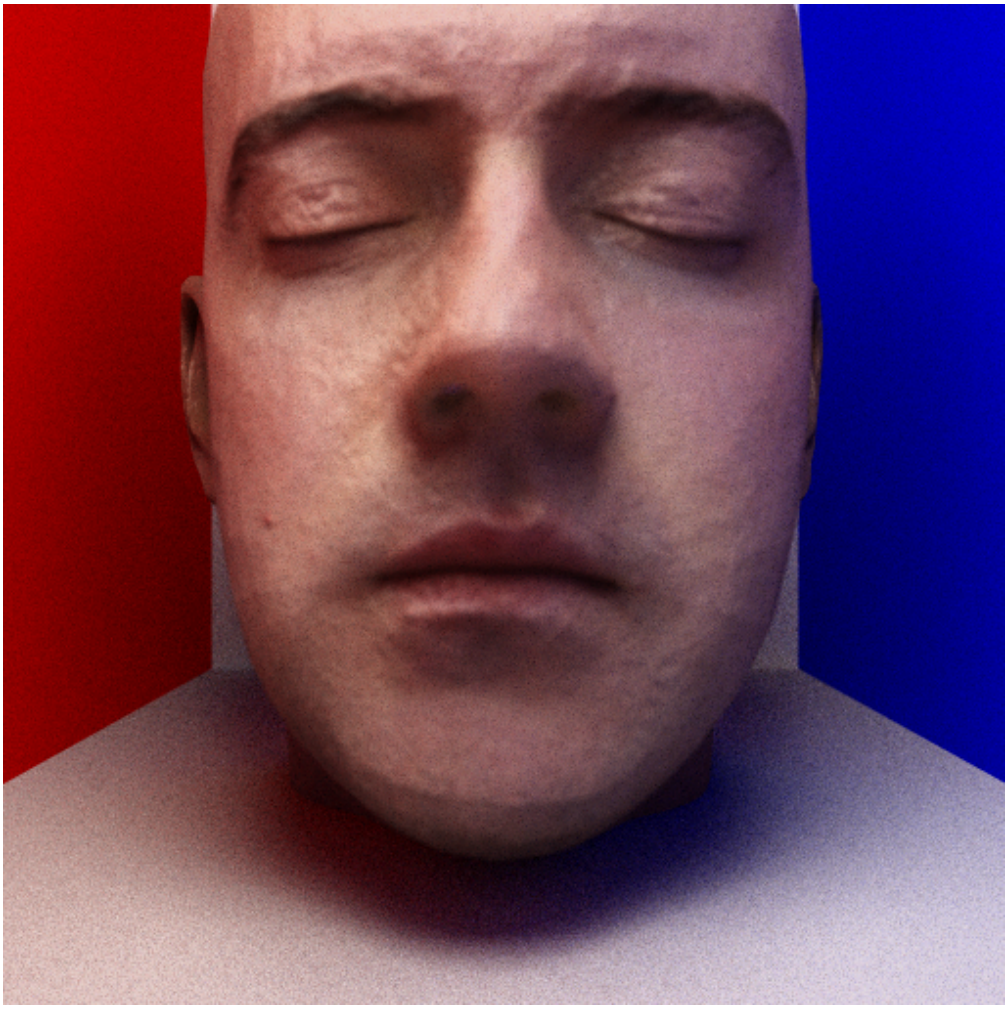


Figure 5.7: Close up of face showing the high level of detail.

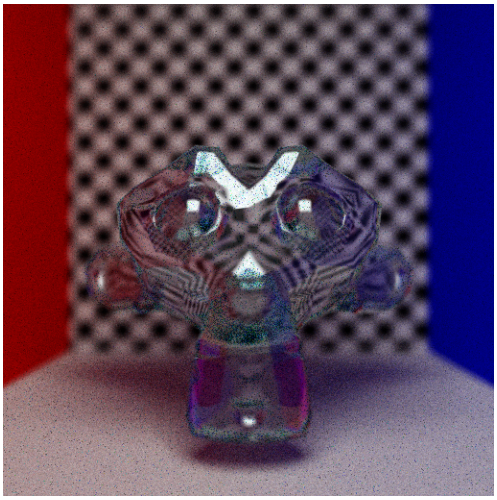
This image shows a very close up head model with normal mapping. The head appears to have realistic pores and wrinkles, even though the base head mesh is relatively low resolution(¡15K triangles). The normal mapping provides a much higher level of detail at very little computational expense.

5.1.8 Depth of Field

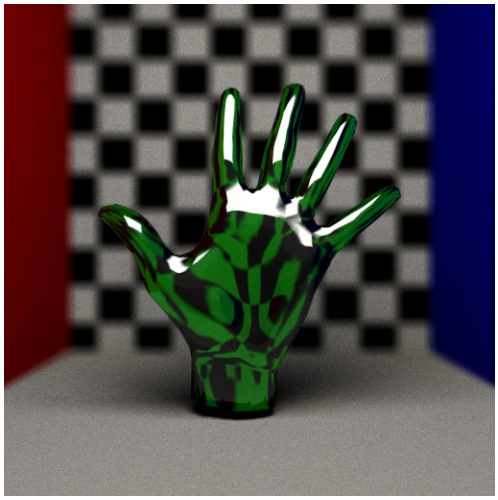
Figure 5.8: An exaggerated example of Depth of Field.

The above image shows an extreme demonstration of focus blurring. The middle cuboid is clearly situated at the focal point, while cuboids in front, and those behind the centre cube are blurred based on their distance from the focal point. This effect is most noticeable on the cuboid to the right of the centre, where the face closest to the camera is heavily blurred, with the blur effect becoming weaker towards the focal point.

5.1.9 Transmission



(a) Glass Suzanne



(b) Coloured Glass Hand

Figure 5.9: Comparison of Flat Shading and Phong Shading

The above images show that the modelling of glass accurately balances reflection and transmission. The image textures behind the glass are also warped realistically, based on the object shape. Note that the image details are more 'in focus' when viewed through the glass.

5.2 Visual Refinement

AccuRay updates the image every few samples, to allow a user to monitor its progress. Once the image is sufficiently rendered (as deemed by the user), the program can be stopped and the image can be used as is. In order to provide reliability, AccuRay writes to two images alternatively every 5 samples, in case the program is closed in the middle of a sample, so there is always a backup.

5.3 Problems

5.3.1 Caustics

One feature detracting from the realism of the rendering of transmissive materials is the lack of caustics. Caustics are the light patterns that occur when light passing through a transmissive substance converges. Unfortunately these

are not an inherent feature of raytracing, however through additional algorithms discussed in the 'Future Work' section in the final chapter, accurate caustics can be modelled.

5.3.2 Rendering Speed

One of the major problems with AccuRay is the extensive time it takes to render complex models, especially with transmissive materials. Due to this, assistance was required in order to produce the high quality images in this report. As explained previously, raytracing is a relatively computational expensive process, however it is easily parallelisable and distributable. As no ray is dependent on another, every ray can be run in parallel; and as no sample is dependent on another sample, every sample can be run in parallel. The latter allows the rendering work to be easily distributed across multiple machines, which is how many of the images in this report were rendered. The number of samples in each render was written as a comment in the image file, allowing simple and accurate automated averaging of results. Still, there are many other ways of speeding up the rendering process, which will be covered in the final chapter of this report, Conclusions.

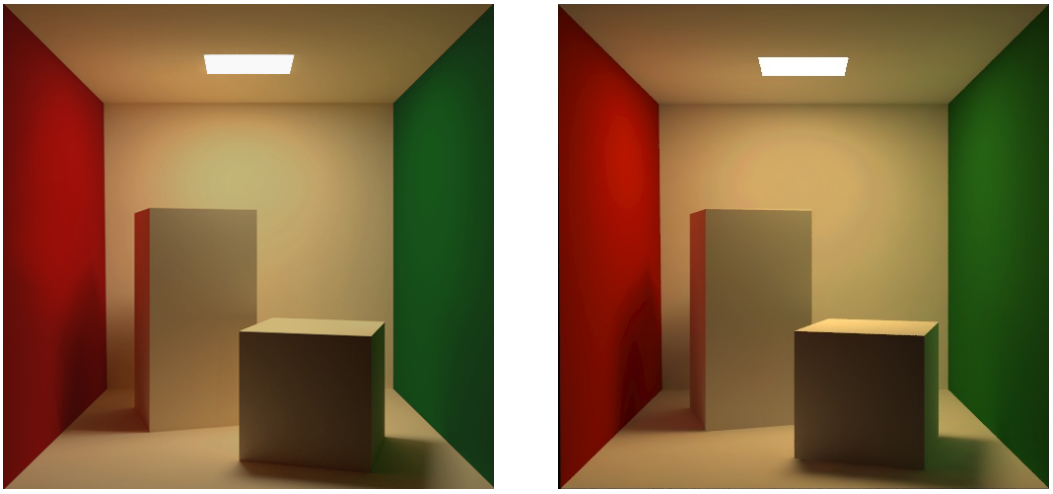
Chapter 6

Evaluation

This chapter evaluates the validity of AccuRays renders, by comparing it both to a sample scene (Cornell Box), and also comparing to the results of other raytracers (namely Blender's Cycles renderer). This chapter covers both quantative evaluation in the form of calculating error from difference images, and qualitative evaluation in the form of a survey. The evaluation also includes a discussion of possible sources of error and inaccuracies.

6.1 Cornell Box Comparisons

In order to confirm the validity of the AccuRay, a well known test scene, the Cornell Box was rendered.



(a) AccuRay Render (b) Original Cornell Box

Figure 6.1: Comparison of the original Cornell Box and an AccuRay Render

In order to more quantitatively analyse this comparison, a difference image has been created. This was done by taking the absolute value of subtracting each pixel in the AccuRay render from the corresponding pixel in the original Cornell Box image.

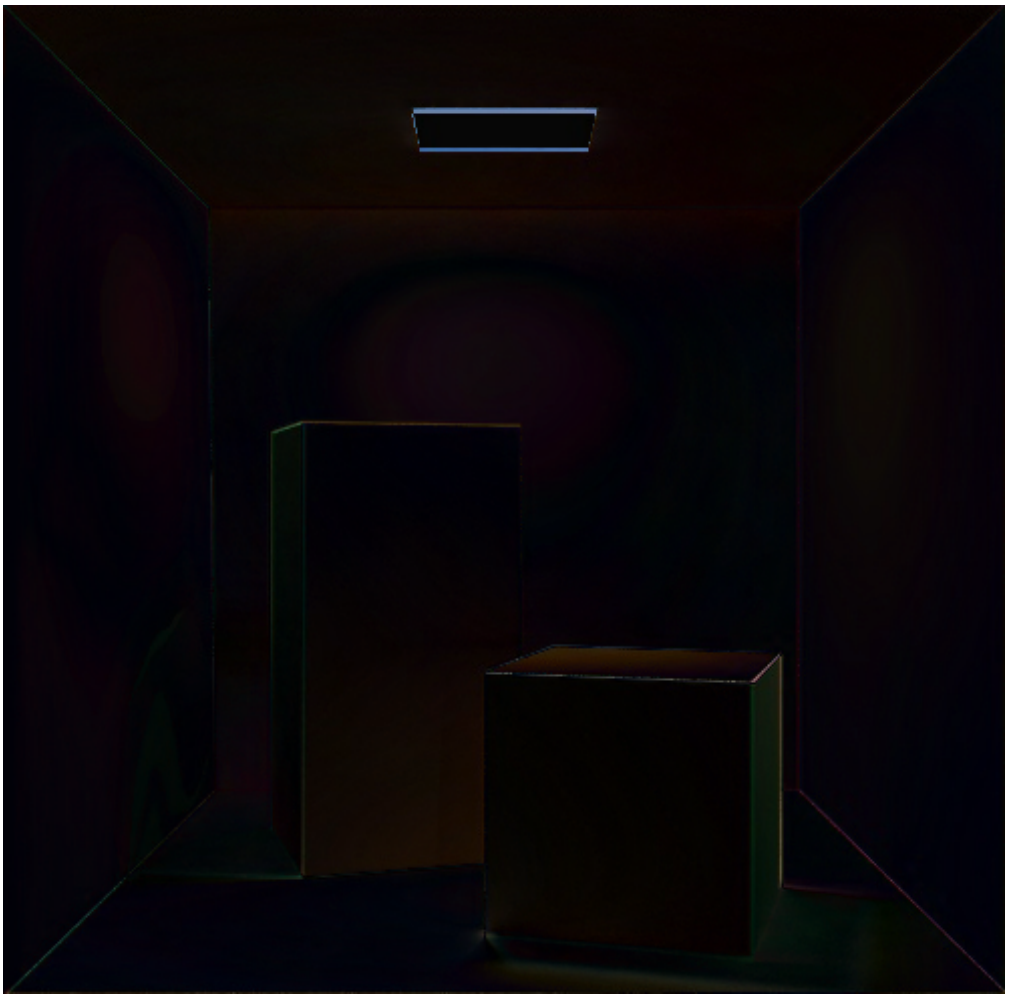


Figure 6.2: Difference image produced from the original Cornell Box and an AccuRay Render

In difference, a darker value means that the images are more similar, with black meaning the pixel values are identical and white meaning the pixel values are completely different. As the image is generally dark, this means the images are quite similar. In order to provide a more precise measure of the similarity, the sum of all the pixel values was taken. Together with the number of pixels in the image, the average pixel value was computed ({0.033, 0.018, 0.029} on a scale of

0 to 1). Averaging these gives a value of 0.027, as a value of 0 would mean the image was black, meaning the images were identical, 2.7% can be given as the error value of these images, giving 97.3% as the accuracy.

6.1.1 Sources of Error

In difference images, edges are from inaccuracies in mapping the object positions to digital co-ordinates, this is most noticeable by the white lines around the light. As this is beyond the scope of AccuRay, and this test concerns only the illumination, these can be ignored.
Unfortunately, the original Cornell data does not give RGB values for the materials in the scene. Likewise, the light is described by its emission spectra. As AccuRay currently only supports RGB materials, so these were estimated based on RGB values provided by the University of Iowa[13]. The bright spots on the walls can be attributed to the light being too bright in the AccuRay render. The excessive colour bleed at the base of the tall box can be attributed to the approximated material colour of the box being too saturated in order to compensate for Accuray's lack of full emission spectra support, and the light's luminosity being too strong.

6.2 Survey

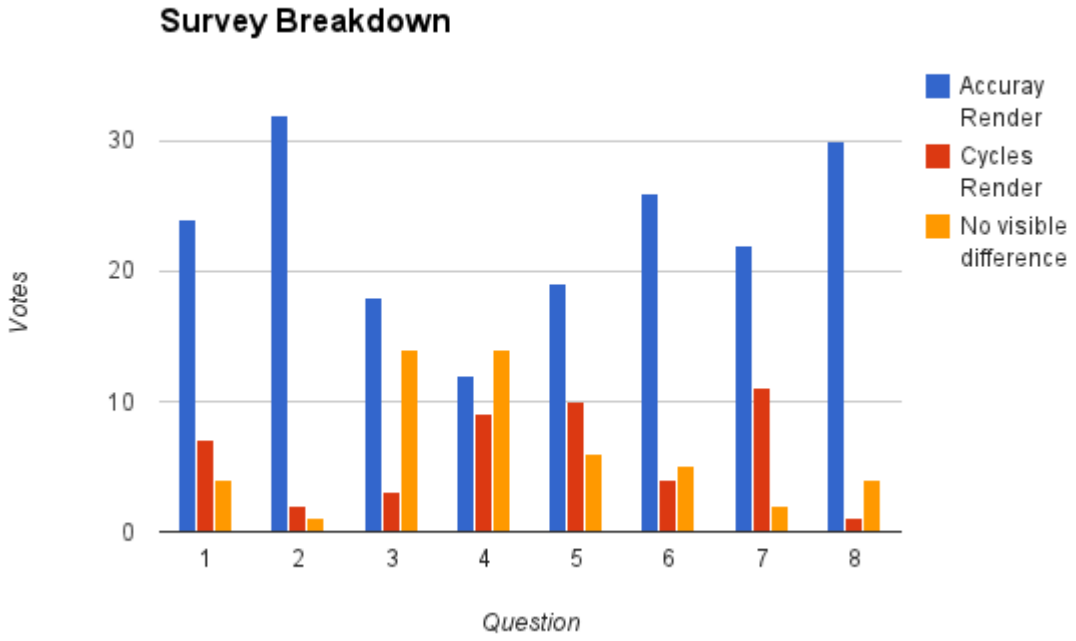
6.2.1 Description

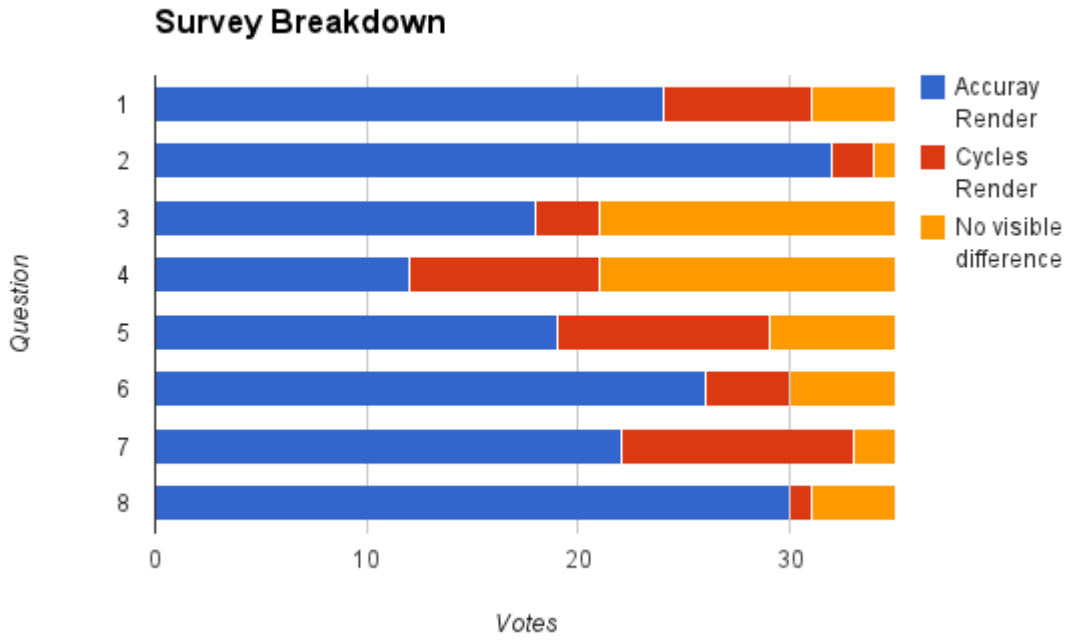
The survey consisted of a series of pairs of images, one rendered with Blender's Cycles raytracer, the other with AccuRay. The 35 participants were asked which image has the most realistic illumination, with the option of choosing 'No visible difference'. If an image was chosen, they were then asked to briefly explain why they chose that image. Participants were also asked to ignore noise, and small changes in position as Blender's co-ordinate system works differently.
The images used in the survey can be found below:

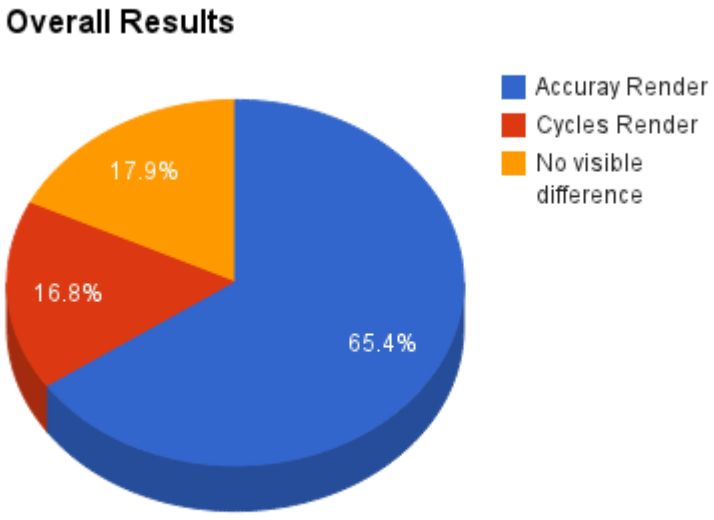
6.2.2 Results

Results of the survey can be found in the table below.

Property Tested	Question	Accuray	Cycles	No visible difference
Diffuse Inter-reflection	1	24	7	4
Specular Reflection	2	32	2	1
Specular Reflection + Image Textures	3	18	3	14
Rough Specular + Image Textures	4	12	9	14
Transmission + Reflection	5	19	10	6
Rough Transmission + Reflection	6	26	4	5
Image Textures + Normal Mapping	7	22	11	2
Coloured Transmission + Reflection	8	30	1	4
	TOTAL	183	47	50



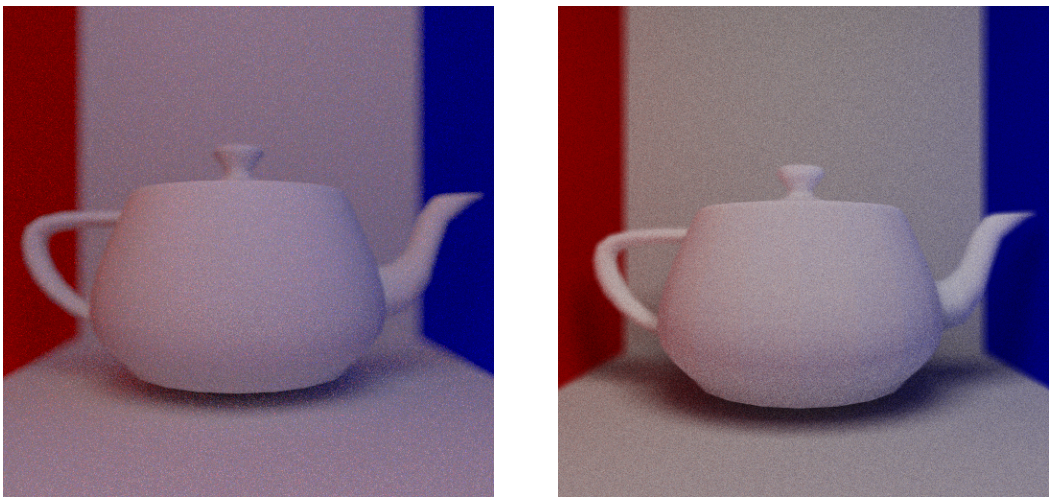

Survey Breakdown
Question
1
2
3
4
5
6
7
8
0
10
20
30
Votes
Accuray Render
Cycles Render
No visible difference


Overall Results
17.9%
16.8%
65.4%
Accuray Render
Cycles Render
No visible difference

6.2.3 Analysis of results

This section features a side by side comparison of the images in each question, with a graph showing the results of the poll, a summary of the participants' justifications for their choices concluding with an evaluation

Question 1 Summary



(a) Cycles Render (b) AccuRay Render

Figure 6.3: Comparison of rendering diffuse inter-reflection

This question was designed to compare how AccuRay's rendering of diffuse inter-reflection (colour bleed) compares to Cycles. Most participants agreed that the effect was overpowering in Cycles, giving a dull appearance, and that AccuRay looked more realistic. Another common comment was that the shadows looked more substantial and "more defined", firmly placing the object in the scene. The main difference between the two images is that the light in the AccuRay render better shows the form of the teapot, participants described this as the second images being "sharper" or having "more contrast". Those who chose the Cycles render stated that the AccuRay render appeared as if it was floating due to the teapot being further away from the floor, however this was not what the objective of the question.

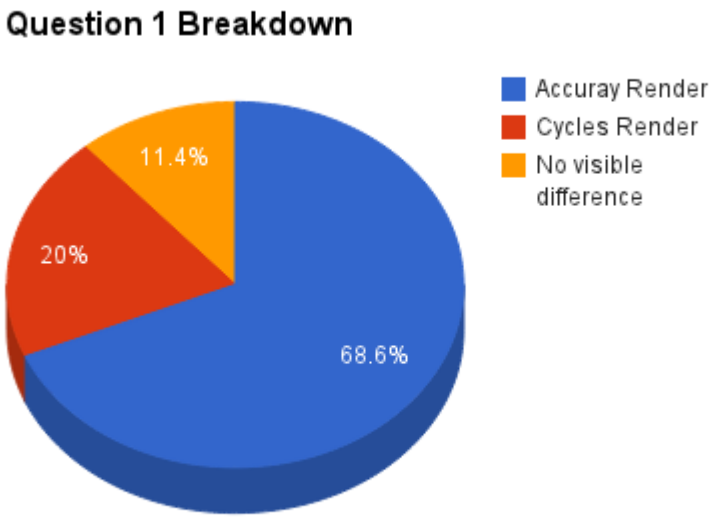
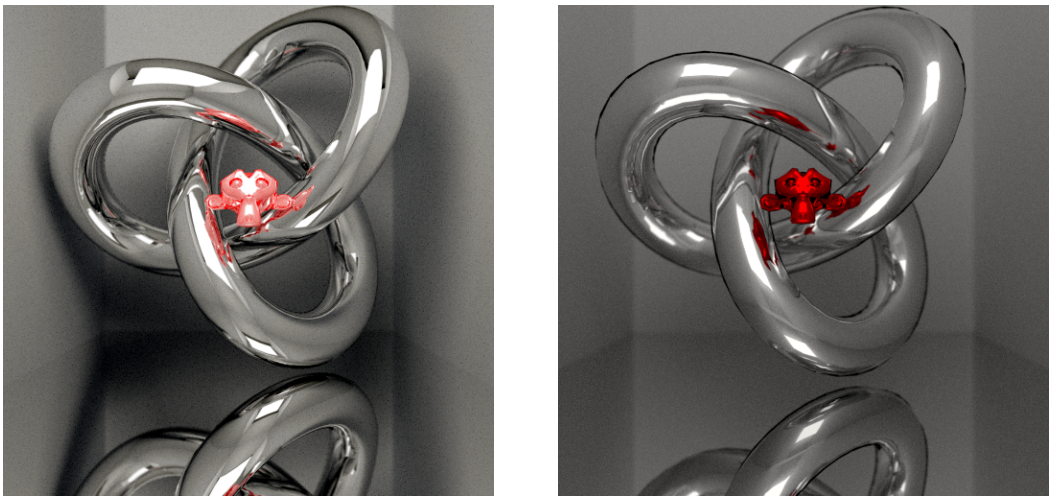


Figure 6.4: Question 1 breakdown

Question 2 Summary



(a) AccuRay Render (b) Cycles Render

Figure 6.5: Comparison of rendering perfect specular reflection

This question was designed to compare how AccuRay's rendering of specular reflection compares to Cycles. Almost all (over 90%) participants agreed that the AccuRay render looked more realistic. This was cited as being due to many reasons, the most notable being the jagged black artefacts along the edge of the trefoil knot. The AccuRay render has "crisper" highlights than the Cycles giving it a more mirror-like appearance (which is the intention), the Cycles render looks like a duller metal. One other major difference is the lack of shadows in the Cycles render, with many participants claiming the Cycles render to appear "photoshopped" due to this.

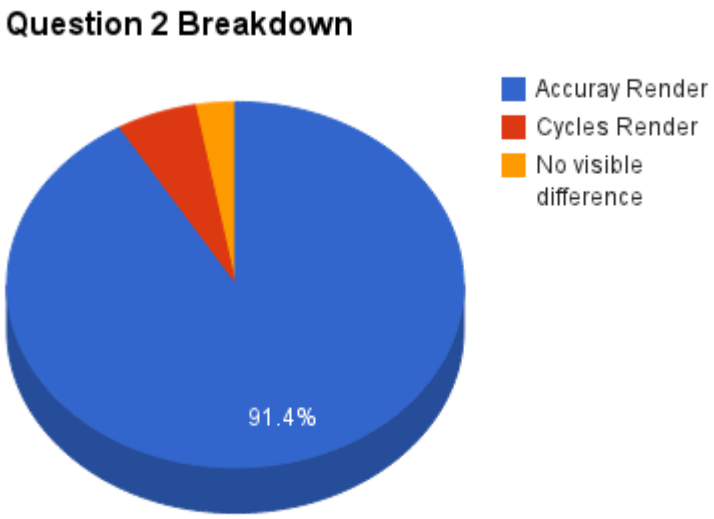
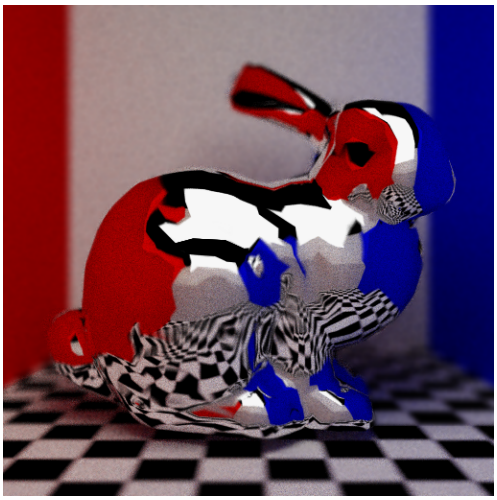
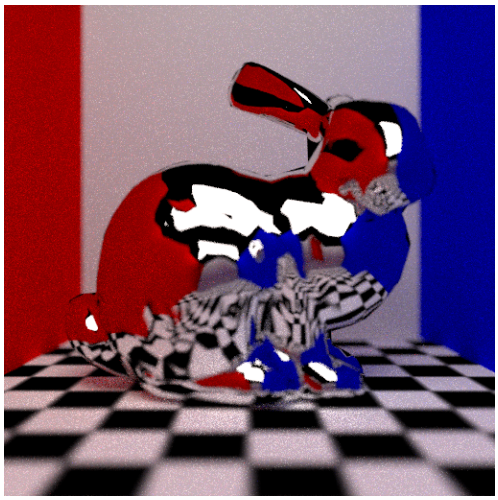


Figure 6.6: Question 2 breakdown

Question 3 Summary



(a) AccuRay Render



(b) Cycles Render

Figure 6.7: Comparison of rendering perfect specular reflection and image textures

This question was designed to test the rendering of images textures and their interaction with specular materials in both AccuRay and Cycles. Over half of the participants were of the opinion that the AccuRay render was more realistic, with another 40% choosing 'No visible difference'. The main reason participants chose the Cycles render was due to the slightly different depth of field, which was not the objective this question and it caused by Cycles. Once again the main reason participants chose the AccuRay render was due to the sharpness of the highlights. The blurred highlights in the Cycles render could be to do with how Blender treats lights or just inaccuracies (optimisations) in calculating the vector direction of the reflected ray.

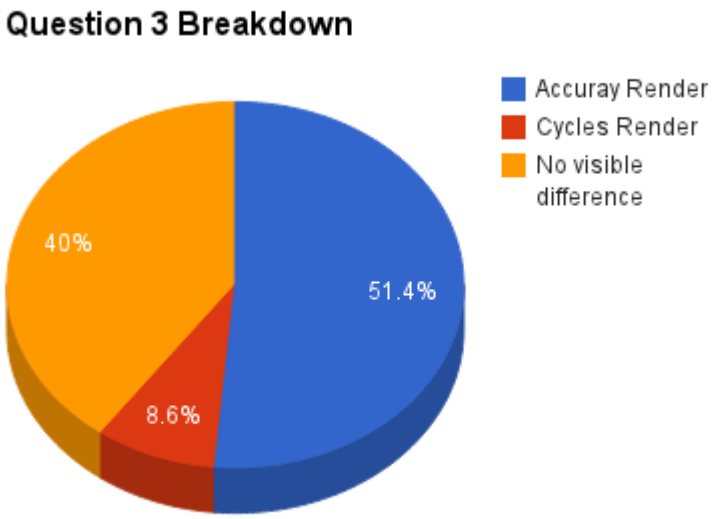
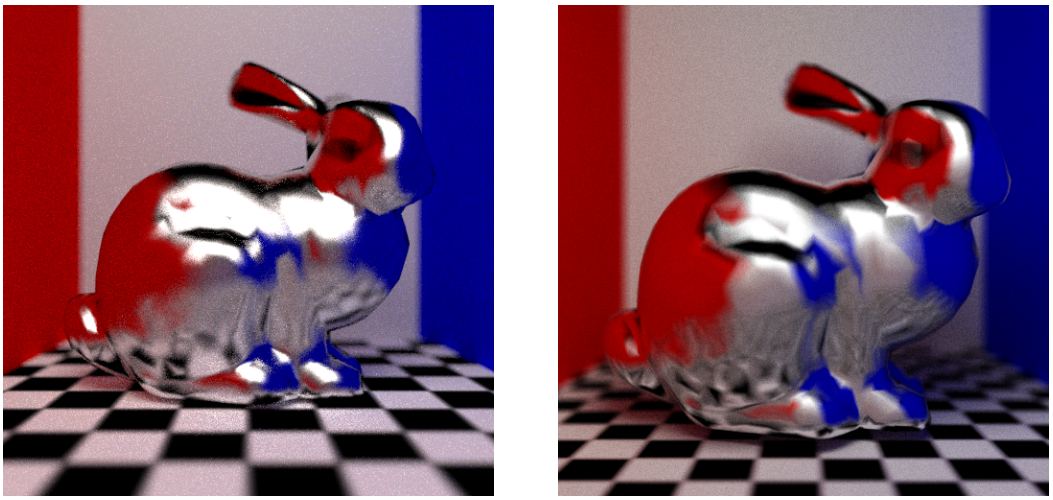


Figure 6.8: Question 3 breakdown

Question 4 Summary



(a) Cycles Render (b) AccuRay Render

Figure 6.9: Comparison of rendering rough specular reflection and image textures

This question was designed to test the rendering of image textures and their interaction with rough specular materials in both AccuRay and Cycles. Unlike any other question, 'No visible choice' was the most popular option with votes from 40% of participants. The majority of those who noticed a difference still chose AccuRay over Cycles. That being said, 'No visible difference' can still be considered a victory as it means that AccuRay competes with Cycles.

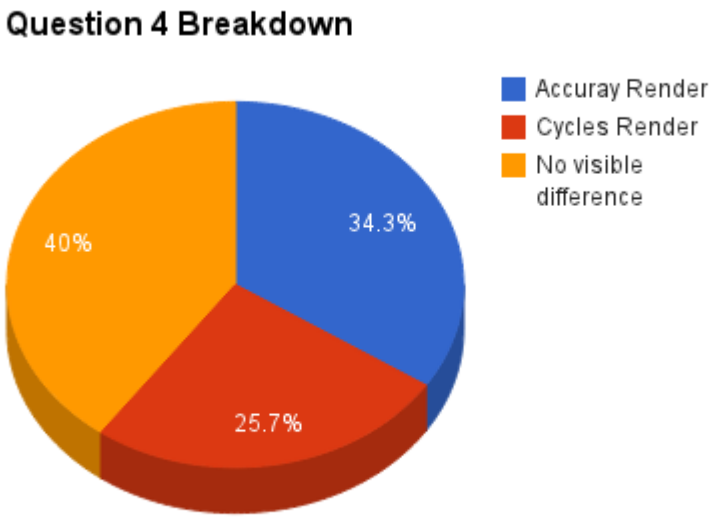
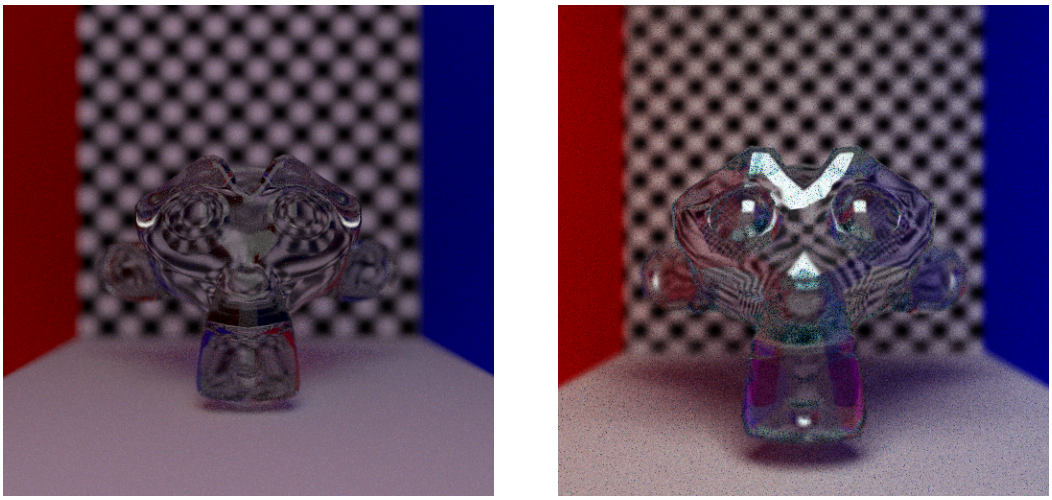


Figure 6.10: Question 4 breakdown

Question 5 Summary



(a) Cycles Render (b) AccuRay Render

Figure 6.11: Comparison of rendering transmissive materials and image textures

This question was designed to test the rendering of transmissive surfaces and their interaction with image textures in both AccuRay and Cycles. Once again, AccuRay's renders was voted to be the most accurate with over half of the votes (54%). The main reason for this was due to the lack of reflections in the Cycles Render. The Cycles Render also has a black artefact at the base of Suzanne's skull caused by a lack of light rays reaching that area, which is unrealistic in this scenario. AccuRay's physically accurate simulation of both reflection and transmission results in an overall more realistic image.

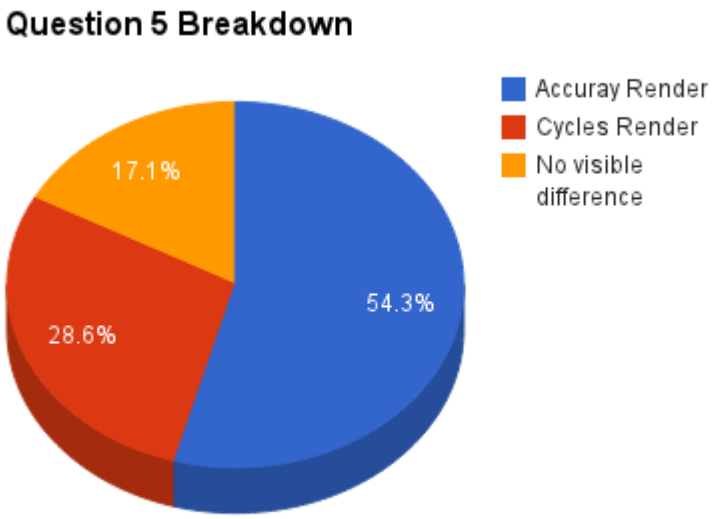
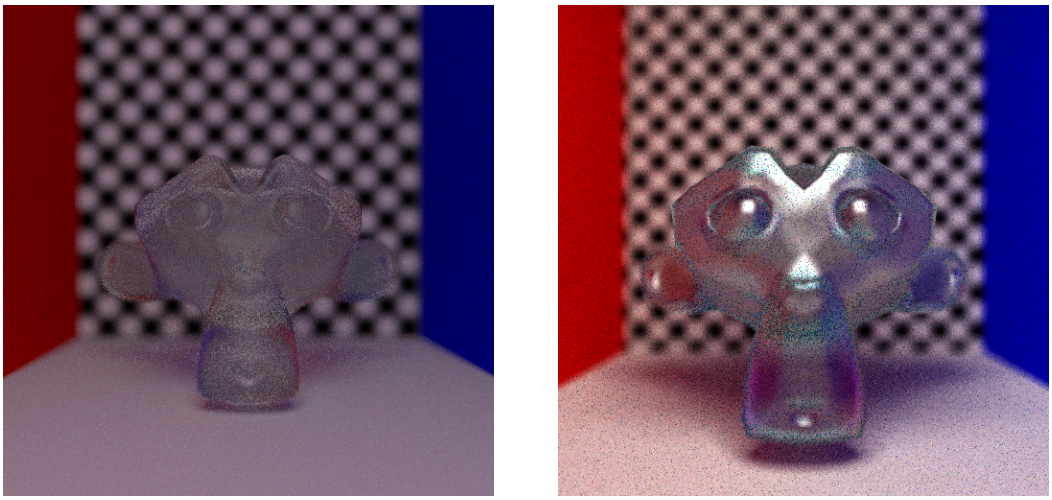


Figure 6.12: Question 5 breakdown

Question 6 Summary



(a) Cycles Render (b) AccuRay Render

Figure 6.13: Comparison of rendering rough transmissive materials and image textures

This question was designed to test the rendering of rough transmissive surfaces and their interaction with image textures in both AccuRay and Cycles. With almost 75% of the votes, AccuRay was voted to be the most accurate. Once again, this was due to the lack of specular highlights in the Cycles render, with participants claiming they chose the AccuRay render as the highlights give form to the objects allowing it too look “less flat”.

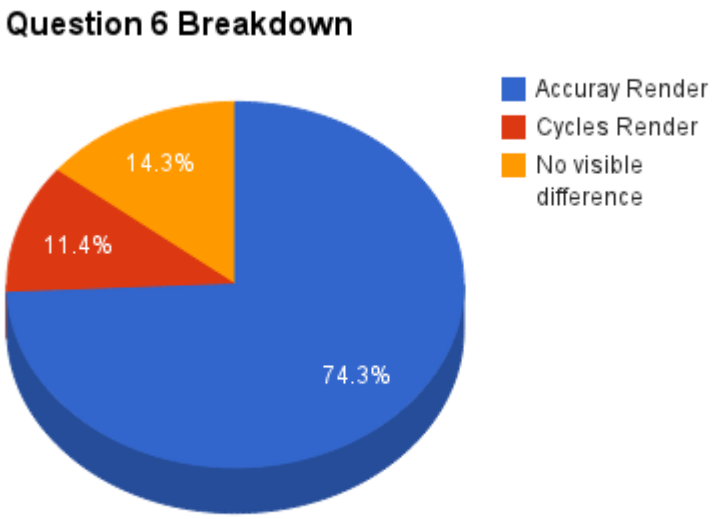
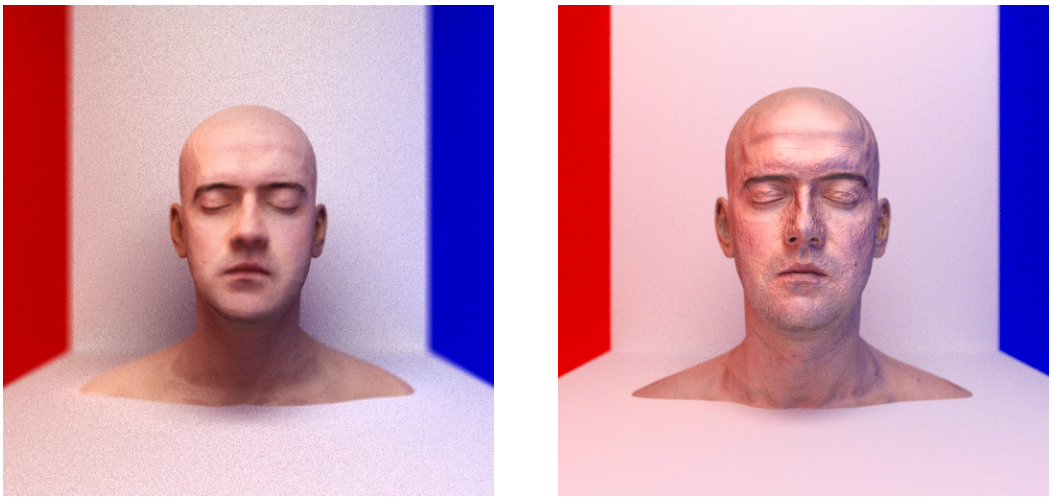


Figure 6.14: Question 6 breakdown

Question 7 Summary



(a) AccuRay Render (b) Cycles Render

Figure 6.15: Comparison of rendering image textures with normal mapping

This question was designed to test the rendering of image textured objects with normal mapping in both AccuRay and Cycles. The main difference in the two images is the strength of the normal mapping; Blender automatically doubles the strength of the normal mapping, which in turn gives the head in the Cycles render an aged and worn appearance. Many found this "extra detail" to be visually interesting and to provide a "greater feel of depth" to the render. Despite this, AccuRay still received the majority of the votes, with Cycles receiving less than a third.

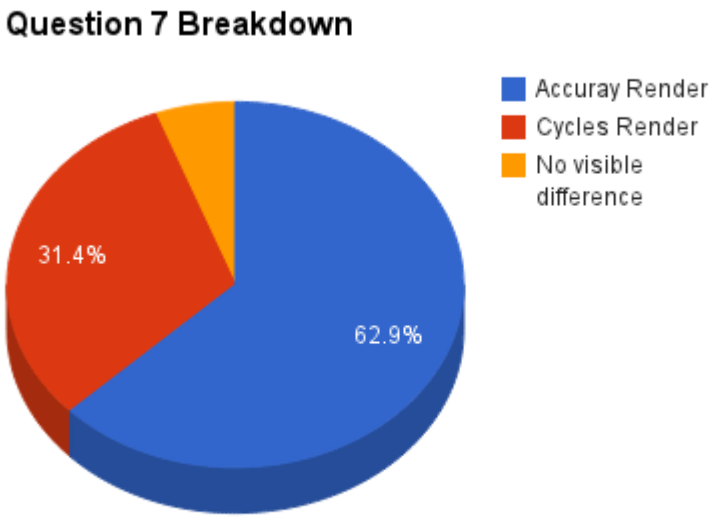
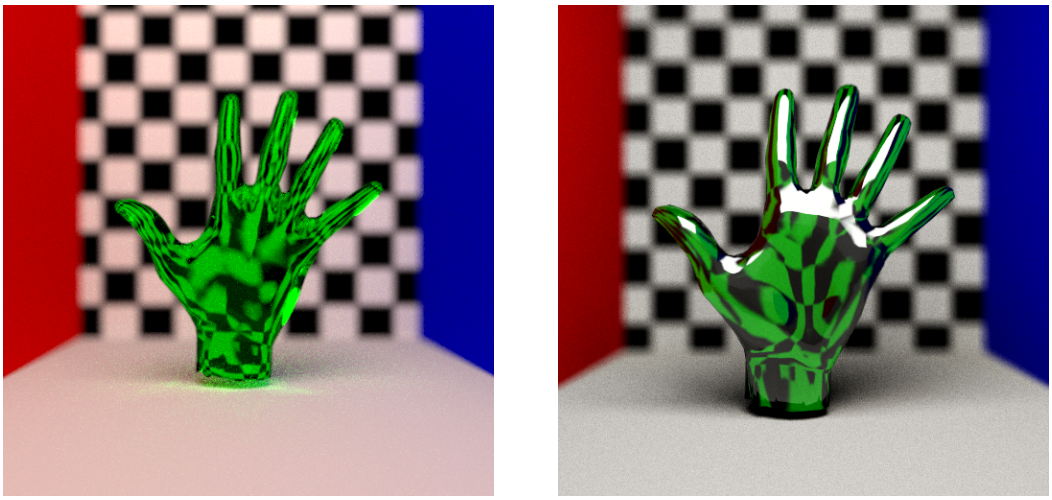


Figure 6.16: Question 7 breakdown

Question 8 Summary



(a) Cycles Render (b) AccuRay Render

Figure 6.17: Comparison of rendering coloured transmissive materials and image textures

This question was designed to test the rendering of coloured transmissive objects and their interaction with images textures in both AccuRay and Cycles. With over 85% of the votes, AccuRay's render was considered more accurate. This was commonly cited as due to the specular reflection of the light and the clarity of the image texture through the hand. Many referred to the Cycles render as appearing formless and flat due to its uniform colouring.

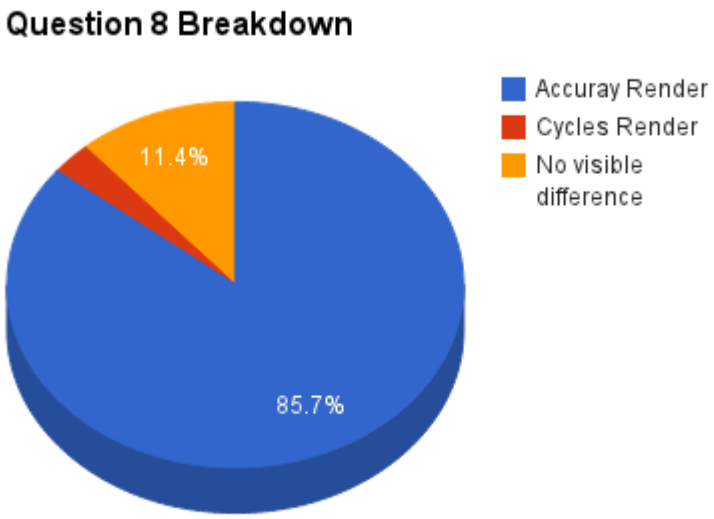


Figure 6.18: Question 28 breakdown

Chapter 7

Conclusion

This chapter concludes the report and provides a reflective overview of both the product and the development process. Also included is a list of additional features which could be used to extend AccuRay in the future.

7.1 Reflection

This section is a retrospective view of the project as a whole, commenting on the results achieved and the lessons learned from the overall project experience.

7.1.1 Achievements

Overall, I feel I have created a fairly fully featured raytracer from scratch.
The main achievement of the project is that, as the survey proves, it is convincingly realistic and it's results can compete with other popular renderers on the market.

7.1.2 Lessons Learned

As this was the largest project I'd ever personally undertaken, it taught me a lot about the importance of time management and self motivation.
A lot was learned about the benefits of prototyping. The prototype was a great way to grasp the fundamental principles of ray tracing, however I may have got carried away with the prototype instead of starting again from scratch using the knowledge I had acquired.
One important lesson learned from this project was knowing when to cut losses. After spending a month attempting to run the raytracing code on the GPU using Nvidia's CUDA API and constantly struggling with memory problems, I was

forced to move on to implement other features, as the deadline was fast approaching.

7.2 Future Work

As raytracing is a fairly broad topic with numerous applications, there are numerous directions in which this project could progress. Below are listed a number of features which could possibly be implemented in the future.

7.2.1 BVH Tree

If more work were to be done on the project, the first feature to be added would be a Bounding Volume Hierarchy Tree. This would drastically speed up rendering times, which has been a limiting factor throughout the project. A Bounding Volume Tree is an extension of the already implemented bounding volume optimisation. A Bounding Volume Hierarchy is a tree of nested bounding volumes, where a ray is only tested for intersections against the triangles within the bounding volume if it intersects the bounding volume.

7.2.2 Photon Mapping

Photon Mapping is another feature which would allow more realistic rendering of transmissive materials, allowing the simulation of caustics (light patterns caused by light passing through a transmissive material).

7.2.3 Chromatic Dispersion

Chromatic Dispersion takes into account the wavelength of light, a term which was previously ignored from the Rendering Equation. Chromatic Dispersion in order to better simulate refraction, dispersion and chromatic aberration.

7.2.4 Animation

Although the raytracer as it stands could be used to produce a series of image frames which could be stitched together using an external utility to create an animation, certain other effects could be achieved by implementing animation inside AccuRay. The most prominent of these being motion blur. Motion blur allows animations to look more natural, rather than a series of static images, and can be achieved by jittering the samples backwards through time.

7.2.5 Participating Media

In the real world, many interesting visual phenomena are caused by participating media such as fog and dust. Participating media receives its name from the way it participates in the light transport, by absorbing or scattering light rays. Adding this feature to AccuRay would give a greater sense of perspective and scale to renders.

7.2.6 Displacement Mapping

Displacement mapping is another image mapping technique which alters the geometry of an object. This effect is achieved by displacing the co-ordinates of an intersection point along the surface normal by a height given by the image map. Displacement Maps are greyscale images where white corresponds to maximum outward displacement and black corresponds to no displacement. This allows the silhouette of an object to be altered, an effect not possible by normal mapping, however it requires a high number of small polygons in order to give realistic looking results.

7.2.7 Other Image Maps

Specular Mapping

Not all surfaces have uniform specularity. In order to achieve this effect, a similar approach to mapping the colour from an image to diffuse values can be used to map an image value to the specularity at an intersection point. Specular maps are greyscale images where black corresponds to zero specularity and white is perfect specularity. These specular maps can be used to accurately simulate variably specular materials such as distressed metal and skin.

Roughness Mapping

Likewise, not all surfaces have uniform roughness. A roughness map feature could easily be added to AccuRay, with black corresponding to zero roughness, and white to a maximum roughness. This effect would allow the rendering of glass with frosted segments and metal surfaces with distressed or worn sections. Although, theoretically, this effect could also be simulated using a normal map with RGB noise in rough areas of varying opacity depending on how rough the surface should be. This however would require a very high resolution normal map, especially for large models.

7.2.8 GPU Acceleration

Taking advantage of the inherent parallelism of raytracing could drastically reduce render times. Raytracing is deemed an embarrassingly parallel problem as no ray is dependent on another and so in theory, they can be all run in parallel. The GPU could be utilised by running each primary ray as a separate thread, doing all ray computation on the device and then the samples pixel colour copied back to the host. This could possibly be further optimised by rendering a chunk of the image eg 32x32 sections to maximise the use of temporal locality (recently used data may still be in cache for faster recall).

7.2.9 SSS

In reality, light doesn't always reflect off the surface of an object. Many materials allow a small amount of light to transmit slightly into the object. Once inside the surface, the light bounces around before exiting, resulting in a higher saturation of the colour. This is known as Sub-Surface Scattering (SSS). Notable examples of this include skin, milk and marble.

7.2.10 GUI

As it stands, AccuRay has no real user interface. The rendering parameters (mesh files, material properties, camera properties, and output images properties) can only be manipulated in the source code. This is obviously not suitable for anything other than personal use. The addition of a Graphical User Interface (GUI) would make AccuRay more user friendly and make altering render properties substantially easier. The GUI could possibly even use OpenGL in order to visualise and manipulate the 3d scene before rendering with the raytracer.

7.2.11 Importance Sampling

Importance sampling is a way of firing rays where they are needed most, rather than uniformly over the image. These are usually at soft shadow boundaries and caustics. Good as requires less samples overall, but it breaks the independence of samples, and so samples cannot be distributed across multiple machines and run in parallel.

Bibliography

[1] James T Kajiya. *The rendering equation. Computer Graphics Volume 20 Number 4 August 1986*. Siggraph '86, 1986.

[2] Michael F.; Greenberg Donald P. Immel, David S.; Cohen. *A radiosity method for non-diffuse environments. Computer Graphics Volume 20 Number 4 August 1986*. Siggraph '86, 1986.

[3] Matt Pharr; Randima Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, 2005.

[4] John Hughes. *Computer graphics principles and practice*. Addison-Wesley, Upper Saddle River, N.J, 2014.

[5] Mateu Sbert. *Information theory tools for computer graphics*. Morgan & Claypool, San Rafael, Calif, 2009.

[6] Matt Pharr. *Physically based rendering from theory to implementation*. Morgan Kaufmann Elsevier Science distributor, San Francisco, Calif. Oxford, 2010.

[7] J. D. Foley and Turner Whitted. An improved illumination model for shaded display, 1979.

[8] Monte Carlo Can H. • unbiased methods – (bidirectional) path tracing [kajiya 1985, lafortune et al. 1993] – metropolis light transport, 1993.

[9] Introduction to feature driven development. `http://agile.dzone.com/articles/introduction-feature-driven`.

[10] Smooth shading. `http://www.cs.berkeley.edu/~sequin/CS184/LECT_2011/L15.html`.

[11] Bram de Greve (bram.degreve@gmail.com). Reflections and refractions in ray tracing. `http://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf`, 2006.

[12] Tomas Möller & Ben Trumbore. Fast, minimum storage ray-triangle intersection.

[13] Cornell box scene. `http://homepage.cs.uiowa.edu/~cwyman/classes/spring07-22C251/code/cornellBoxScene.txt`.

Appendix A

List of Figures

List of Figures