COLOUR MANAGEMENT AND SPECTRAL RAY TRACING

MASTERS THESIS

A DEVELOPMENT OF THE TRADITIONAL RAYTRACER TO INCORPORATE PHYSICAL TREATMENT OF LIGHT ACTING AS ELECTROMAGNETIC WAVES

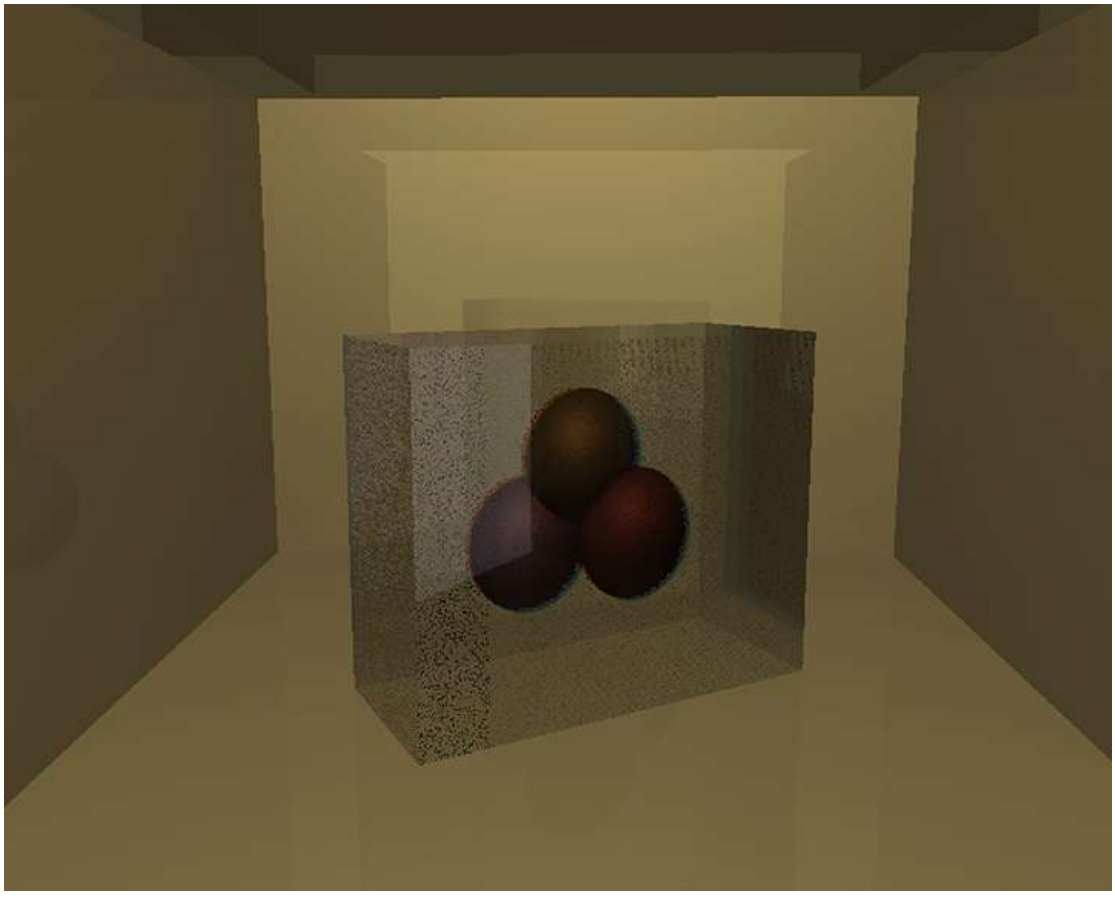


Figure 1: Refractive Glass Cube

LUCY WARD

N.C.C.A. BOURNEMOUTH UNIVERSITY

2nd September 2005

Contents

COLOUR MANAGEMENT AND SPECTRAL RAY TRACING....1
MASTERS THESIS....1
N.C.C.A. BOURNEMOUTH UNIVERSITY....1
2nd September 2005....1

CONTENTS....2

LIST OF FIGURES....5

LIST OF TABLES....6

LIST OF EQUATIONS....7

CHAPTER 1 INTRODUCTION....8

2.1 Introduction....9
2.2 History of Ray Tracing....9
2.3 Why colour management was developed....9
Methodology for image storage and display :....10
ideal methodology:....10
current practice:....10

CHAPTER 3. TECHNICAL BACKGROUND....11

3.1 Ray tracing....11
3.2 Physics of Light....11
Refraction and Reflection....12
Total Internal Reflection....12
Absorption, Reflection and Transmission....12
3.3 Material Properties....13
3.4 Spectral Power Distribution (SPD)....13

CHAPTER 4. SOLUTION....14

4.1 Reading the Light and Material Data....14
The Light Data File....14
The Material Data File....14
4.2 Parser....15
Class Diagram....15
Pushing data....16
Retreiving data....16
4.3 Colour Management....16
where xr, yr, zr are the chromaticity coordinates of the red phosphor, xg, yg, zg of the green and xb, yb and zb of the blue.[]....16
4.3.1 Calculating the RGB values....17
4.3.2 Colour scaling and correction....17
Chosen Scaling Method....17
Gamma Correction....17
4.4 Reading in the scene file....18
4.5 Spectral Ray Tracer Extension....19
4.5.1 Splitting the rays....19
4.5.2 Refraction of each ray....20
4.6 Light Interaction with objects in the scene....21
4.6.1 Fresnel....22
4.7 The cornell scene....23
Metamerism....23
Additive Mixture....23
4.8 The prism scene....24
4.8.1 Spotlight....24

4.8.2 Prism *25*
4.9 LENS 26
4.9.1 Chromatic Aberration *27*

5. CONCLUSIONS AND FUTURE WORK **29**

6. APPENDIX A **30**

Example of the light data used to find the light's overall intensity and colour 30
Example of the material data used to find the material's overall intensity and colour 31

APPENDIX B **32**

CHAPTER 7. REFERENCES **34**

CONTRIBUTIONS AND CHANGES 34
Files added to original Raytracer 34

BIBLIOGRAPHY **35**

List of Figures

FIGURE 1: REFRACTIVE GLASS CUBE.....1

FIGURE 2: COLOUR HANDLING PIPELINES.....8

FIGURE 3: CLASSICAL RECURSIVE RAY TRACING [REF].....9

FIGURE 4 : REFRACTION.....10

FIGURE 5: RGBSTACK CLASS DIAGRAM.....13

FIGURE 6: COLOUR BANDING.....16

FIGURE 7: TOTAL COLOUR OF LIGHT SPLIT INTO N RAYS.....18

FIGURE 8: GRAPH OF INTENSITIES OF LIGHT.....21

FIGURE 9: SIMILAR COLOURED OBJECTS ACTING SEPARATELY UNDER VARYING LIGHTING.....22

FIGURE 10: PRISM.....22

FIGURE 11: PRISM DIMENSIONS.....23

FIGURE 12 : RAY PASSING THROUGH PRISM.....24

FIGURE 13: CHROMATIC ABERRATION OF A LENS.....25

FIGURE 14: CHROMATIC ABERRATION WHEN LIGHT SPLIT INTO 3 RAYS.....25

List of Tables

TABLE 1: PLANES OF THE PRISM.....23

List of Equations

EQUATION 1: SNELL'S LAW OF REFRACTION.....10

EQUATION 2: TOTAL INTERNAL REFLECTION.....10

EQUATION 3 : XYZ CALCULATION FROM SPD.....11

Chapter 1 Introduction

This thesis outlines the design and implementation of a spectral extension to a ray tracer. The extended program handles the behaviour of light with improved physical accuracy. The developed ray tracer can handle refraction and splitting of light such as is seen in a rainbow or prism.

The first part of the project is a colour management system. Light and material data are converted from physical tables of spectral intensities, into the standard CIEXYZ colour space, and then from XYZ into RGB colour space. This system needed to maintain a degree of accuracy not usually required by standard ray tracers because the light needed to behave with more physical accuracy when interacting with the objects in the scene.

The second phase of the project was to develop the ray tracer, provided by Jon Macey[06HIL] to handle spectral light properties. A standard ray tracer casts a single ray for each pixel. The solution implemented casts a bundle of rays. These rays split up the light information to span the visible spectrum. The change in refractive index with frequency was implemented, making it possible to simulate prismatic effects and chromatic aberrations.

Chapter 2 briefly introduces ray tracing and colour management. Chapter 3 briefly describes the process of ray tracing, the physics of refraction and reflection and colour handling. Chapter 4 describes the program, how it works and problems that arose. Chapter 5 details what extra work could be done to improve the ray tracer.

The Appendices include an example of the light and material file.

The HTML and pdf documentation of the programs is presented in the C.D. at the end of this document.

Chapter 2. Previous Work

2.1 Introduction

To create a spectral ray tracer the light frequency data is converted into RGB using the colour management system. The light needs to be divided into a number of rays to represent electromagnetic waves acting independently with varying frequency. The average colour of each ray, calculated from averaging the spectral intensities for each ray needs to be determined, and scaled in intensity. For instance, if the light is very blue, with little red, the ray which covers the blue end of the spectrum will need to be more intense than the red. Once each ray's colour is determined each ray is traced through the scene separately then the colours from each ray is combined to create the final image.

Jon Macey provided a ray tracer, based on F.S. Hill's book on Computer Graphics using OpenGL [06HIL], which was then adapted to cope with the extra rays. A Colour Management system also needed to be implemented to treat the light correctly.

2.2 History of Ray Tracing

Ray tracing is a realistic way of rendering images. It traces the path of a ray of light as it moves through the scene, calculating the reflection, refraction or absorption of a ray when it hits an object. Because of the structure of ray tracing it is simple to implement reflection and shadows, which are less straightforward in other rendering methods.

For each pixel on the screen a ray is shot directly into the scene until it hits an object. The pixel is given a colour, determined by the material properties and the interaction of the light with the object.

Scientists at the Mathematical Applications Group, Inc in New York in the late sixties were the first to use ray casting for computer graphics by calculating how the rays bounced off surfaces and penetrated objects and refracted within. Initially the ray casting was used to help the government with military applications, but the company set up a commercial animation studio in 1972, using ray casting to produce films and commercials. These first ray tracers only sent one ray into the scene per pixel, and the process didn't recurse to produce more rays upon hitting a surface.

Turner Whitted made the next big jump in 1979 when he extended the ray tracer to carry the ray on when it hit a surface, with reflection, refraction and shadow rays.[13WIL]

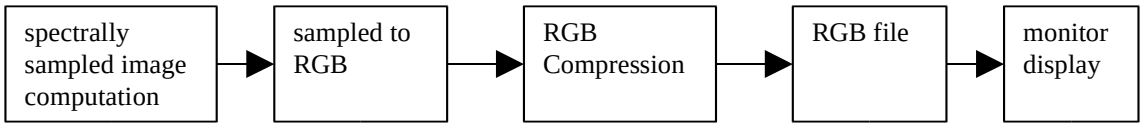
Ray tracing is limited because it traces rays from the camera, the opposite direction to the rays of light. This leads to problems with caustics and diffuse light calculations. Due to this constraint other methods have been developed, such as radiosity algorithms which produce photon maps. These send out photons from the light source, and have the disadvantage of being very slow and wasteful, as the whole scene is traced, including areas not seen from the camera.

2.3 Why colour management was developed

In the CG industry it is important that colour can accurately be reproduced on different monitors, printers and screen with consistent results. The transformations required to keep the colour consistent when using different devices require tables, equations and other tricks.

Methodology for image storage and display :

ideal methodology:



current practice:

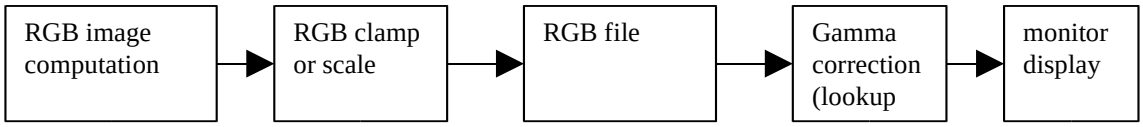


Figure 2: Colour handling pipelines [05HAL]

Monitors vary significantly in colour characteristics of their phosphors, and the amount of light emitted for a particular applied voltage.

CIE, the Commission Internationale d'Eclairage system of colourimetry is the standardised method of representing colour. The CIEXYZ space is the colour space to which display monitors can be calibrated. From this colour space the correct colours of the image can be calculated taking into account the method of display, such as the printer or monitor, which will have varying white points, and display colour with differing linearity. [05KIN]

Chapter 3. Technical Background

3.1 Ray tracing

When a light hits the surface it generates a possible three more rays, a reflected ray, a refracted ray and a shadow ray.

The reflected ray is traced until it hits another object in the scene, and the colour of the second object where it is hit is multiplied by how reflective the object is and added to the pixel colour. A refracted ray travels through the object and the colour when it next hits a surface is added, multiplied by how transparent the object is. The shadow ray is a ray traced between the hit point on the surface and the light. It tests whether the surface is visible to the light by checking if there is any opaque object between the surface and the light.

```
For each pixel in image
{
        Create ray from eyepoint passing through this pixel
        Initialize nearestT to INFINITY and NearestObject to NULL

        For every object in scene
        {
                If ray intersects this object
                {
                        If t of intersection is less than NearestT
                        {
                                Set NearestT to t of the intersection
                                Set NearestObject to this object
                        }
                }
        }

        If NearestObject is NULL
        {
                Fill this pixel with background color
        }
        Else
        {
                Shoot a ray to each light source to check if in shadow
                If surface is reflective, generate reflection ray: recurse
                If transparent, generate refraction ray: recurse
                Use NearestObject and NearestT to compute shading function
                Fill this pixel with colour result of shading function
        }
}
```

Figure 3: Classical recursive ray tracing [13HIL]

3.2 Physics of Light

Light is the small, visible, part of the electromagnetic radiation. Electromagnetic radiation acts both as a particle, photon, and a wave. White light can be separated into a range of colours which form a

spectrum of white light, from violet with a wavelength of about 3.8x10^-7, to red with wavelength of about 7.7*10^-7. Electromagnetic waves have frequency, f, and wavelength, λ, and speed, c such that

$$c = f \lambda$$

The speed of light is approximately $3*10^8 m/s$. The energy of a wave is proportional to the square of the amplitude of the wave [04DOB].

Refraction and Reflection

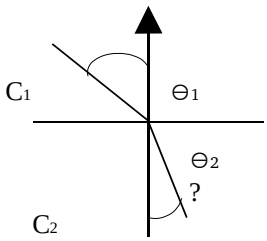


Figure 4 : refraction

$$n = \frac{\sin\theta_1}{\sin\theta_2} = \frac{c_1}{c_2},$$

Equation 1: Snell's law of refraction [04DOB]

where n = refractive index , c1 is the speed in the first medium, c2, the speed in the second medium. The refractive index of a material is the constant factor of how much the light slows down as it enters the material. Because lights of different frequencies travel at different speeds they will refract by different amounts. This is why prisms produce a spectrum of light from white light, because the light separates as it travels through the prism. The blue light deviates more than the red light. [04DOB]

Total Internal Reflection

When light is travelling inside an object, if the angle of incidence of a ray exceeds the critical angle, c, for the medium, total internal reflection occurs. When this happens no light passes through, all light is either absorbed or reflected. This has been implemented in the ray tracing code.
Total internal reflection occurs when:

$$\left[1 - \left(\frac{c_1}{c_2}\right)^2 \times (1 - N.dir)^2\right] < 0,$$

Equation 2: Total internal reflection [05HAL]

where N = normal, dir = direction of incident ray

Absorption, Reflection and Transmission

Initially the light's energy is determined by the sum of all the frequencies emitted from the light source. When it hits an object, the reflectivity and absorption of the object is taken into account. The absorption coefficient determines how much the energy of the light is decreased on contact with the

surface, the remaining light is divided between the reflected ray and the transmitted ray. The reflectivity determines what proportion of the light is reflected. The final factor to be accounted for is the angle at which the ray hits the surface. The smaller the angle, the larger the energy of the refracted ray, which decreases linearly until no energy is transmitted at the critical angle. The amount of energy reflected or refracted is calculated by Fresnel's equation, described in chapter 4.

3.3 Material Properties

A material has electrical properties. The magnetic field of the wave affects the electrons in the material. How free the electrons in the material are controls the optical properties. There are two types of material, dielectric, which allows light through, or conductors, which reflect all light. There are also composites which allow for subsurface scattering type effects.

Dielectrics have very stable electrons so are mostly unaffected by the light, hence light is largely unaffected as it passes through the dielectric material. Refraction is the la=rgest factor, and the change of speed.

Conductors reflect light because the electrons are freer and oscillate when hit by the wave. As there are still damping forces on the electrons there is not total energy conservation, the absorption index of the material needs to be taken into account.

An illuminated surface needs to maintain energy equilibrium,

energy in = energy reflected + energy refracted + energy absorbed

Reflection and refraction have two components, coherent and incoherent (scattered). Incoherent components can't be recorded as there's no phase information about them but the incoherent components can be estimated by diffuse calculations. [05HAL]

3.4 Spectral Power Distribution (SPD)

Light is defined by its spectral power distribution, the power of the light at each wavelength along the visible spectrum. The SPD contains all the basic physical data of the light. It can be measured by a spectrophotometer. From this the luminance and chromaticity of a colour can be determined to find the CIExyz colour. The SPD of light from an illuminated surface is the product of the percentage of reflectance of the surface and the SPC of the incident light on the surface. [5]
P (?) = spectral power. XYZ can be determined by the following equations:

$$X = \int P(\lambda)\bar{x}_\lambda d\lambda \,,\; Y = \int P(\lambda)\bar{y}_\lambda d\lambda \text{ and } Z = \int P(\lambda)\bar{z}_\lambda d\lambda \,,$$

Equation 3 : XYZ calculation from SPD [05HAL]

where $x_\lambda, y_\lambda, z_\lambda$ = colour matching functions. However a more 'forced' method is often used, taking a spectrophotometer to measure the spectral power of the emitted light, and hence calculate the XYZ values by calibration either by parameterised functions of an interpolated table. This is what was implemented in the colour management system. This provided sufficient accuracy as it matched the accuracy of the spectral data of the light.

Chapter 4. Solution

The purpose of this project was to treat light and colour with better physical accuracy. The emphasis was on the representation of light as electromagnetic waves of varying frequency, rather than a visually stunning piece.

4.1 Reading the Light and Material Data

Originally the ray tracer first produced an openGL scene but as this was irrelevant this part was removed.

Firstly the user specifies the scene file to ray trace, then the number of rays the light is to be split into. SceneRib.::scene() is called which passes the material and light files to be read in the SceneReader file.

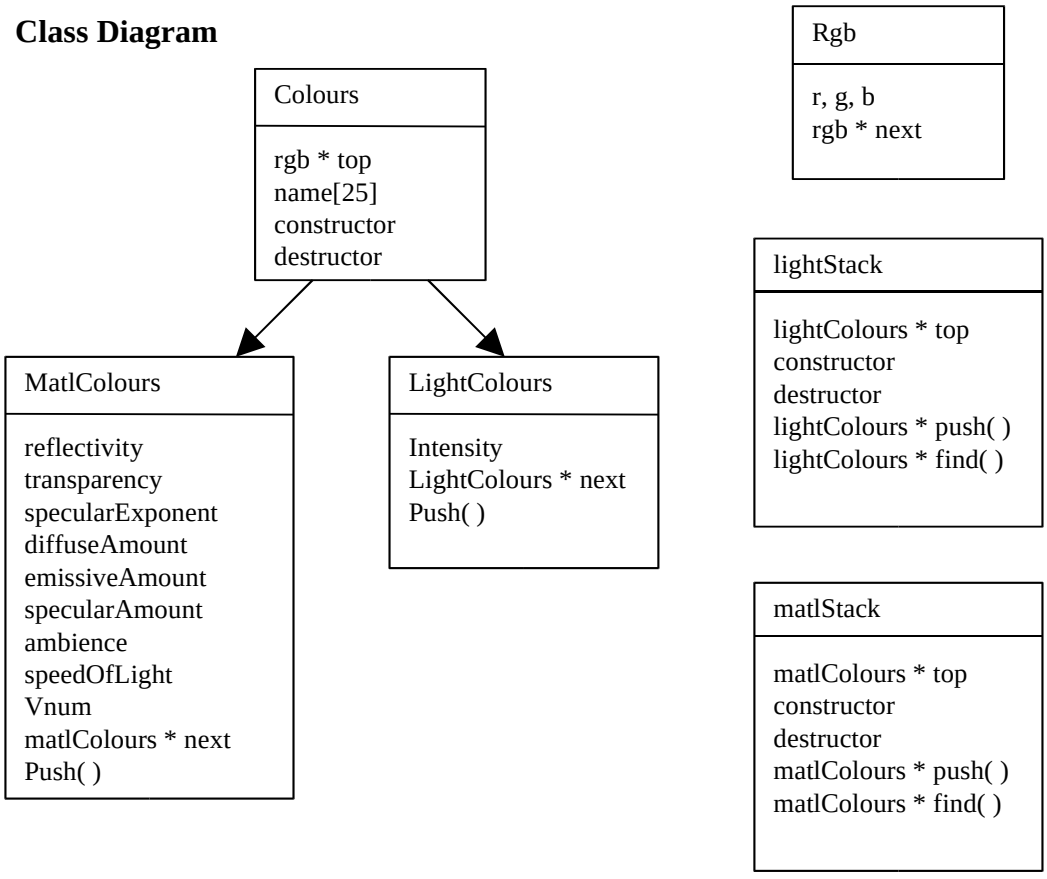
The light file is loaded by the function loadlight(), which is a parser, developed from a piece of code written by Jon Macey which has been adapted. This reads through the light file, finding the key words, followed by their relevant values. Each light is pushed onto the lightStack class, written by Chris Ward, which holds the intensity value and the name of the light. The lights are separated by the term 'endfilter'. The light spectral data is passed into the FreqList array, which is accessed in the SceneReader::getLightColour() function. The material file is read in the same way, but the material stack holds more information.

The Light Data File
The light data file, ***lights.filter*** holds the overall intensity of the light and the intensities at each of the frequencies. See appendix A for an example of the file.

The Material Data File
The material data file, ***all.mtl*** holds the reflectivity, specular exponent, transparency, diffuse amount, emissive amount, specular mount, ambience, speed of light values and the intensities at each of the frequencies. See appendix A for an example of the file.

4.2 Parser

Class Diagram



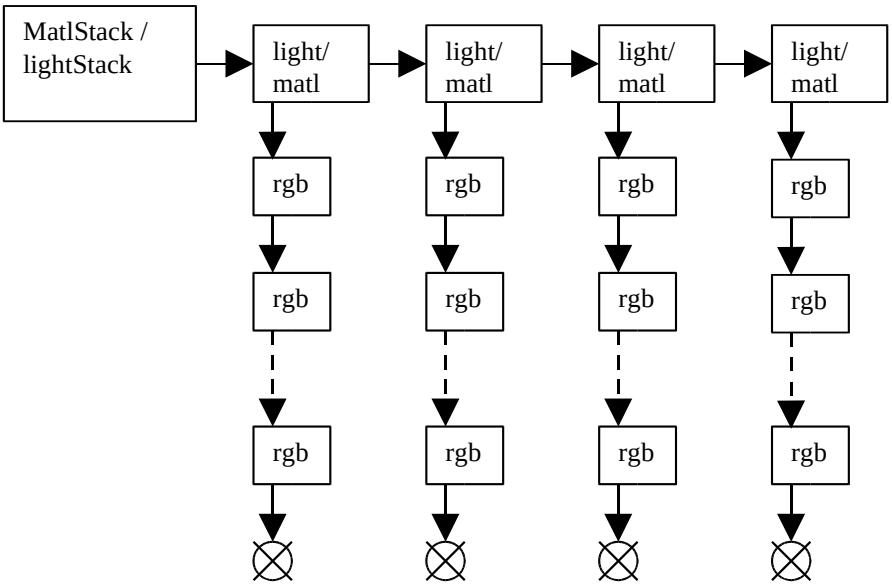


Figure 5: rgbStack class diagram

Pushing data
The push functions in the lightStack and matlStack classes are used to add a new member to the list of colours. These push functions return the address of the newly created colour. The caller then pushes all the rgb items onto the colours using the address passed back.
Retrieving data
In Scene::getObject the name is used to find the material in the stack. The materials list is scanned through, taking off all the rgb values.

4.3 Colour Management

The first part of the project involves writing a colour management system, which treats light accurately by taking the frequency data of the material or light and converting that through to the standard CIExyz then through to RGB values for displaying on the monitor.

In the SceneReader::getLightColour() function the light spectral data is converted through the colour management system into CIEXYZ colour space by multiplying the spectral data by the XYZ conversion tables in spectralLocus.h. The polyfit program was implemented in an attempt to replace the tabular lookup between spectral data and XYZ, but the accuracy given by the tables in spectralLocus.h was sufficient for the accuracy of the ray tracer. These XYZ values are summed to get the total X, Y and Z value, then normalised before being passed to the SceneReader::convertXYZtoRGB() to get the RGB values.

In the convertXYZtoRGB function the XYZ data is scaled by a matrix which takes into account the monitor specifications, the white point, red, green and blue values, and the gamma value, how non-linear the screen is. The values are obtained from ColourProfile.h, which holds information for various monitors. The white point can be specified separately, in LightPoint.h, as the white point on the monitor can be changed.

The relationship between the primaries' luminance and the colour's CIE tristimulus values is given by

$$T = C*L$$

where L is a vector containing the primaries luminances.
T is the vector containing the colour's tristimulus values.

C is the 3x3 matrix:

$$\begin{bmatrix} \frac{x_R}{y_R} & \frac{x_G}{y_G} & \frac{x_B}{y_B} \\ 1 & 1 & 1 \\ \frac{z_R}{y_R} & \frac{z_G}{y_G} & \frac{z_B}{y_B} \end{bmatrix}$$

where xr, yr, zr are the chromaticity coordinates of the red phosphor, xg, yg, zg of the green and xb, yb and zb of the blue.[1]

RGB values are now obtained, however, because of the nature of frequency light being represented by RGB, not all visible colours can be represented by RGB in the 0 to 1 range. The problem with RGB colour, when converted from frequency data of the light spectrum, is it doesn't stay in the positive octant, which it needs to for displaying purposes.. CIEXYZ is the standard hypothetical primary set of all visible colours in the positive octant. It is a standardised method of measuring colours. Knowing the XYZ co ordinates allows the image to be reproduced easily and exactly in many different media, and on screens with varying monitors.

CIE chromaticity coordinates for an object can be calculated by first measuring the objects spectral power at each wavelength by the weighting factor from each of the three colour matching functions. Summing these gives the three tristimulus values, X, Y and Z, and then normalised. [05HAL]

The CIE procedure converts the spectral power distribution (SPD) of light from an object into a brightness parameter, Y, and two chromaticity coordinates, x and y, (where z=1-(x+y)). The chromaticity coordinates map the colour with respect to hue and saturation on the 2D CIE chromaticity diagram.

4.3.1 Calculating the RGB values

Converting XYZ into RGB for a screen is done using a transformation matrix generated from the monitor phosphors and the white point by expressing the chromacity relationships in matrix form. RGB has several problem areas. Firstly, information is lost when the spectral curves are sampled and represented by only three colours. Secondly, RGB Sampling functions are not selected on the basis of sampling theory, but only on what the viewer perceives visually to create the correct colour, looking only at colour reproduction, not colour computation requirements. Thirdly, RGB sampling functions are specific to the RGB primaries used. Computations in different RGB colour spaces may not produce the same result.

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} S_r(r_x) & S_g(g_x) & S_b(b_x) \\ S_r(r_y) & S_g(g_y) & S_b(b_y) \\ S_r(r_z) & S_g(g_z) & S_b(b_z) \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

red phosphor : r_x, r_y, $r_z = 1-(r_x + r_y)$

green phosphor : g_x, g_y, $g_z = 1-(g_x + g_y)$

blue phosphor : b_x, b_y, $b_z = 1-(b_x + b_y)$

white point : wx, wy, wz=1-wx-wy

[05HAL]

4.3.2 Colour scaling and correction

When RGB values are computed from XYZ they may fall outside the range 0-1, the RGB*gamut*. There is no perfect way to deal with this problem, the simplest approach is to clamp the RGB values into this range, by truncating any values which fall outside the range, however this changes the hue and saturation. The method chosen is to scale the RGB values, preserving the hue and saturation but changing the intensity. The hue and saturation was more important than the intensity, however the intensity of each ray did cause problems. It may have been better to scale all the values by the range of the RGB which fall furthest outside the gamut, but this wasn't implemented due to time constraints.

Chosen Scaling Method

For example if the rgb value was {0.4, -0.6, 1.2} the smallest value is 0.6 below the lower limit, 0, and the largest value is 0.2 above the upper limit, 1. The total amount out of range is $0.6+0.2=0.8$. The first step is to shift all the values up so the minimum is 0, then divide by the range, 1.8,

{0.4, -0.6, 1.2} ?{1.0,0.0, 1.8} ?{0.556,0,1}

Gamma Correction

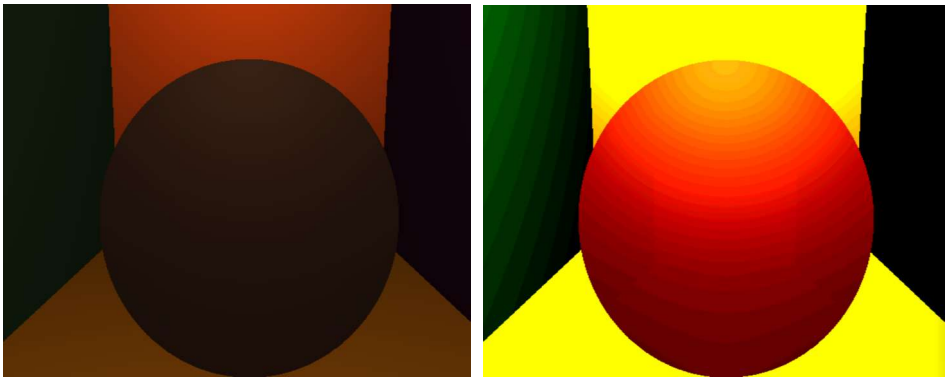
The next crucial stage is gamma correction. Images are stored in files of linear intensity byte data, one byte for each red, green and blue value for each pixel. By this method RGB{0.5, 0.5, 0.5} would have half the intensity of RGB{1.0, 1.0, 1.0}. However video colour monitors and the human visual system is non-linear. Colour computations are based on linear intensity values.

Lookup value $= Intensity^{1/\gamma}$, where gamma, ?, is the non-linearity of monitor, this value is usually between 2 and 3, where 1 is linear. If gamma is miscalculated the images won't have the correct intensity. As the shape of the graph goes, lower colours are more compressed than high ones, which, when they go through the process of Gamma correction, can lead to the lower intensity regions being banded.

The RGB values need to be multiplied to the power $= 1/\gamma$. For the above example, RGB{0.556, 0, 1}, with a gamma value of 2.2 becomes $\{0.556^{1/\gamma}, 0^{1/\gamma}, 1^{1/\gamma}\} = \{0.556^{1/2.2}, 0^{1/2.2}, 1^{1/2.2}\} = \{0.766, 0, 1\}$. Because the outer ranges of 0 and 1 are preserved, the gamma value only changes the red value. Using the wrong value of gamma will result in incorrect image contrast and chromaticity shifts. If gamma is too large it will result in intensities being mapped too bright, if gamma is too small the image will be less intense than it should be.[1]

When an image has been stored with the wrong intensity, and the image is brightened the effects of banding can be seen. The lower 10% of the intensity values in the image are stretched into the lower 35% of the display resolution range. The image file is mapped into a larger increment for display, thus the available resolution of the display system in the low intensity range, where the visual system is most sensitive, is not being used. The observable result is that images tend to be banded in the low intensity regions due to intensity quantizing to a lower resolution that the available resolution of the display device. The gamma function can be applied to the computed colour values from the illumination model before they are converted to byte values for storage.

The image below shows how the banding is more evident on the lower range of colours when an image is brightened after storage:



original image after brightening

Figure 6: Colour banding

4.4 Reading in the scene file

Once all the lights and materials have been stored with the RGB data evaluated the program returns from the SceneRib file and calls the Scene::read() function, where it reads the scene file.

Each light specified in the scene file has a name, position, and locality. The name is used to find the relevant colour data, from the lightStack class. When the correct light is found the RGB values for each ray need to be found and stored for the light colour. The rgb values are stored in a stack, which is looped through and assigned for each ray:

```
n=0;
for(rgb value at top of the light; while currLight not null; go to next rgb value on stack)
{
        assign the rgb values to the nth light ray colour
        increment n.
}
```

For each object in the scene a material is assigned by the command UseMaterial*materialName*. The material data is assigned in much the same way as the light, only because the material has a diffuse, specular, ambient, emissive, reflective and transmissive colour, the process is more complex. The colour value found is scaled by the reflectivity, transparency, specular, diffuse, emissive and ambient terms to get each of the respective colours.

4.5 Spectral Ray Tracer Extension

The second stage of the project is the spectral extension for the ray tracer which, when refracting and reflecting light will take into account the frequency of the light hitting the object, and hence reflect and refract the correct frequencies at the correct angles.

4.5.1 Splitting the rays

In a standard ray tracer the ambient, diffuse and specular terms have one RGB value each and one light ray is sent out into the scene from each pixel. Light is usually a combination of electromagnetic waves of different frequencies. When one RGB value is found for a light it is, in effect, averaging the values of all the waves to get one ray of light. The program created allows the light to be represented by up to 81 rays.

The rays are divided up into n rays, depending on the accuracy required. The frequency data will be split up into n sections. The rgb value will then be calculated by each of these.

Light and material colour frequency data has been put into an array of 81 intensities of frequencies ranging from 380nm to 780nm. If the accuracy is changed from 1 ray to 3 this states that 3 rays which start at the same light source and start with the same angle are to be traced through the scene. These rays are divided into different frequencies, low, medium and high. These frequencies are then converted into 3 RGB values.

The first step in splitting up the light is to divide the 81 values by three groups of 27:

$$81/3 = 27$$

The first colour is taken by summing the first group of XYZ colours and leaving the other XYZ values to be used by the other groups.

For each ray the spectral frequency data is multiplied by the XYZ conversion table, summed, then normalised.

For the second step the XYZ value is converted, as is standard practice when dealing with the usual, single ray, into an RGB value. For instance, if the light is split up into 9 rays, 9 RGB values are found, which span the spectrum

giving the combined ray colour below

The third step sees the RGB value stored in the rgbStack to be collected in Scene::shade( ).

These 9 RGB values need to be scaled to have the same combined intensity as when the light was only one ray. Because the XYZ values are normalised this isn't straightforward. The way this was approached was to find the sum of the spectral values, and calculate the sum of intensities in each ray, and divide it by the total sum of values. This still didn't work completely. Because of the way the values fall outside the gamut, when the rays are split they may need to be scaled more or less than before. When only one RGB value is found for a light, this is much more of an oversimplification than when one ray is calculated. As can be seen in the diagram below, the summed colour of the rays gets darker when the colour is divided into more rays. However, it clearly shows the colour calculation gets more accurate as the colours are almost indistinguishable when the colour is split into 27 or 81 rays.

Figure 7: Total Colour of light split into n rays

In Scene::shade( )

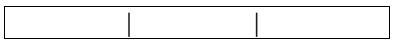
Loop for(each ray)
{
add the light contributions, taking as the light colour, material diffuse colour, and material specular colour the individual colour from the rgbStack.
}

This is done while looping for each light, so the light contribution can interact more accurately with the objects in the scene.

4.5.2 Refraction of each ray

The refractive index needs to be calculated for each frequency. For instance, if the accuracy is 3 when the frequency bands are split, the boundaries are calculated as follows:

380 780



The band width $= (780-380)/3$, with the middle points of the boundaries at

$380+\left(\frac{1}{2}\right)\times\left(\frac{780/380}{3}\right)$, $380+\left(\frac{3}{2}\right)\times\left(\frac{780/380}{3}\right)$ and $380+\left(\frac{5}{2}\right)\times\left(\frac{780/380}{3}\right)$.

At these frequencies the refractive angles need to be found.

To take a more accurate frequency to find the refractive index from a better method would be to sum up the total intensities in the group and divide by 2. Then find which frequency the half point of the intensities was found at and use that to find the refractive index, instead of just the midway point between the boundaries.

With 81 frequency values the colour can be split up into:

1 group of 81 (default)	Low	Level 1
3 groups of 27		Level 2
9 groups of 9	Medium	Level 3
27 groups of 3		Level 4
81 groups of 1	High	Level 5

When it came to determining the change in refractive index of a light with frequency when passing through a particular material there were several methods written. The first method used the polyfit program to estimate the refractive index for frequency. Data was used from a table of refractive indices of crystal quartz at 20 degrees C, from Chance Catalogue OC1, 1962, but it wasn't successful at getting a good fit of the tabular data, even with considerable accuracy:

RefractiveValue = 1.816731 – 1.85629*frequency+3.514845*pow(frequency, 2)-3.12787*pow (frequency, 3)+1.310301*pow(frequency, 4) – 0.2085667*pow(frequency, 5);

The second method takes the end values and linearly interpolates. The refractive index at 380nm us 1.56973. At 780nm it is 1.54769.

This gives a linear interpolation of

RefractiveValue = 1.56973 – 0.000054419*frequency

The third method takes the general refractive index of the material into account and scales the second method to get the range change in refractive index.

RefractiveValue = speed of light + range/2 – range * frequency/405

Naturally, the chromatic effect is very subtle, so to show the effect in transparent, refractive objects, the change in refractive index was heavily multiplied.

4.6 Light Interaction with objects in the scene

For each pixel rendered in the scene the colour is calculated. The light for each of these pixels is built up in layers.

Firstly the first object hit is found, if nothing is intersected the background layer is given and nothing further is done. If the ray hits an object the colour is initially set to the scene's global ambient colour the object's emissive colour, the glow, is added, along with the ambient term of the object.
For each light in the scene the light contribution to the object hit is added. There are two equations the light term adds, the diffuse colour and the specular colour.
The next part is the ray tracing part, where the reflected and refracted rays are determined and added. In this next recursion they are traced until they hit another object where their colour contribution, scaled by how reflective and transmissive the object is, is added to the nexthit point, like the light contributions are.

The total light intensity observed by the eye, consisting of the ambient, diffuse and specular components, is

I(?) = ambient + diffuse + specular + transmitted

$$I = I_a \rho_a + I_d \rho_d \times lambert + I_{sp} \rho_s \times phong^f ,$$

where I_a is the ambient intensity and I_s is the intensity of the light source and where we define

$$lambert = \max\left(0, \frac{s.m}{|s||m|}\right) \text{and } phong = \max\left(0, \frac{h.m}{|h||m|}\right)$$

ρ_a, ρ_d and ρ_s are reflection coefficients and f is the specular exponent, as is declared in Hill's OpenGL book.[]

Specular light is calculated as below:

$$normal.(v+s)^{specularExponent} \times (lightColour \times materialSpecularColour)$$

and diffuse light is calculated as below:

$$(s.normal) \times materialDiffuseColour \times lightColour$$

[05HAL]

To create further physical accuracy the change in refractive index with frequency is taken into account. The light is split up into a number of rays which span the visible light spectrum. Each of these rays are traced separately, in effect, different passes are done.

Because a light source is a combination of electromagnetic waves which create an overall emissive colour, these waves can be divided up. The XYZ value is calculated for each frequency.

The tables used provide light data composed of 81 intensities of frequencies, ranging from 380 to 780. These can therefore be divided into up to 81 different rays, each with a different light colour and intensity.

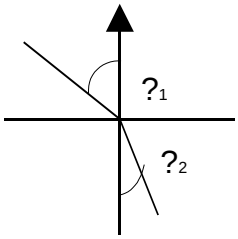
4.6.1 Fresnel

Fresnel equations calculate how much light is reflected / refracted which is dependent upon the polarization of the incoming light. Light polarized parallel to the reflecting surface:

$$p_{para} = (\cos\theta_1 - \varepsilon \cos\theta_2) / (\cos\theta_1 + \varepsilon \cos\theta_2)$$
$$p_{perp} = (\varepsilon \cos\theta_1 - \cos\theta_2) / (\varepsilon \cos\theta_1 + \varepsilon \cos\theta_2)$$

Combined reflected light, $F_r(\theta) = \frac{1}{2}\left(p_{para}^2 + p_{perp}^2\right)$

combined transmitted light, $F_t(\theta) = 1 - F_r(\theta)$, where ?is the angle of incidence. [12STE]



The Fresnel calculations were attempted, but unfortunately, due to time constraints, it wasn't possible to ensure these were working correctly so the reflection and refraction were merely multiplied by the constants reflectivity and transparency. This meant that when ray tracing a sphere, instead of the sphere being more reflective at the edges, it was a constant transparency and reflectivity. This is a fairly subtle difference but it does add to the realism.

4.7 The Cornell scene

Metamerism

Metamerism is when light of different spectral composition can be seen as the same colour. Spectral light compositions can be split into sets of metamerism colours, a group which all look the same and these can then be translated into another colour system.

Additive Mixture

An additive mixture is when two colours are close enough together to be seen as a single colour. Two objects with different colours can look the same under certain lights. The reason for this is the range of frequencies which combine to give the object its overall colour will match the overall colour of the other object which is produced by a different sum of frequencies. [05HAL]

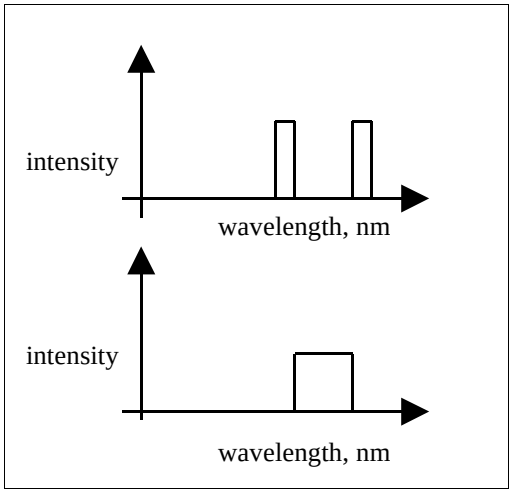
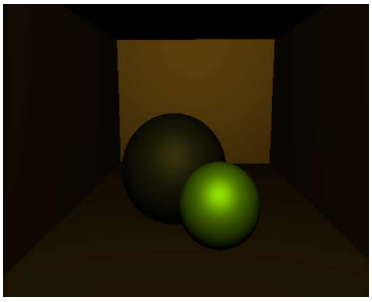
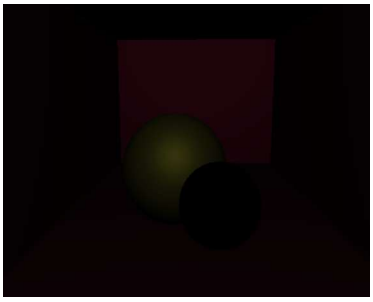


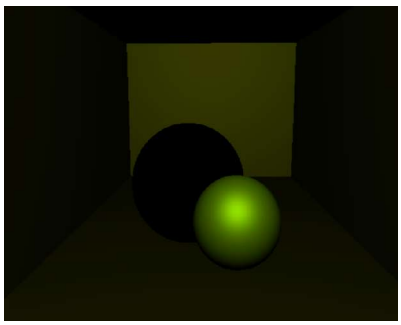
Figure 8: graph of intensities of light



whole light, illuminates both objects



light completely illuminates back sphere but not front sphere



light completely illuminates front sphere but not back sphere

Figure 9: Similar coloured objects acting separately under varying lighting

4.8 The prism scene

Figure 10: Prism

4.8.1 Spotlight

The first things which needed adding to create a prism to demonstrate the light splitting when it refracts through the prism, was a spotlight. A narrow beam of light was needed, heading in one direction and hitting the prism.
RtLight is amended to hold direction and angle data.

4.8.2 Prism

The ray tracer only provided for simple object shapes, such as the sphere, cube and tapered cylinder. A prism was created as an equilateral triangle with height. Normals were calculated, which with the equations of the planes and a point on each of these planes gave the data to alter the cube class into a prism class.

The prism is designed as follows:

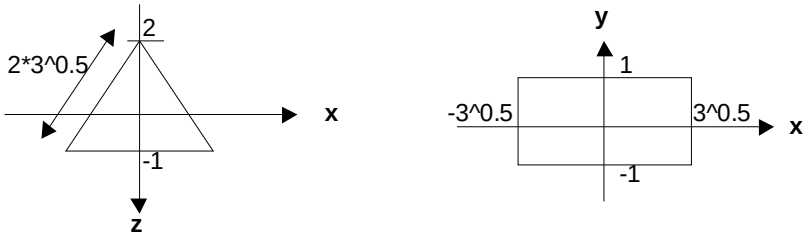


Figure 11: Prism dimensions

Plane	name	Equation	Outward normal	Spot
0	Top	Y=1	(0,1,0)	(0,1,0)
1	Bottom	Y=-1	(0,-1,0)	(0,-1,0)
2	Front	Z=1	(0,0,1)	(0,0,1)
3	Left	Z= - ?3 x-2	(-?3/2,0,-1/2)	(-?3/2,0,-1/2)
4	right	Z= +?3 x-2	(+?3/2,0,-1/2)	(+?3/2,0,-1/2)

Table 1: planes of the prism

To determine whether a ray is inside or outside an object each plane in turn needs to be considered. If the equation of the plane is know, and the normal of the plane, the hit point can be found at $m.(s+ct-B)=0$. This equation can be reordered to find for t, $t=\frac{m.(B-s)}{m.c}=\frac{numer}{denom}$

Where m is the normal of the plane, B is a point on the plane, S is the start point of the ray and c is the direction of the ray. By finding the top and bottom half of the equation it is possible to determine whether the ray is entering, exiting or running parallel to the plane at this hit point. [12HIL]

If the $denom > 0 \Rightarrow exiting$

$denom < 0 \Rightarrow entering$

$denom = 0 \Rightarrow parallel$, a further test is then done.

$numer > 0 \Rightarrow$ wholly outside the object

$numer < 0 \Rightarrow$ wholly inside the object

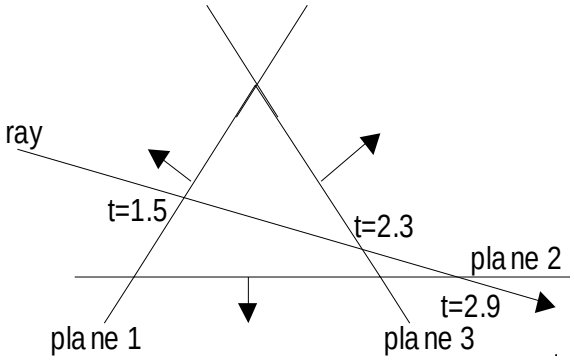
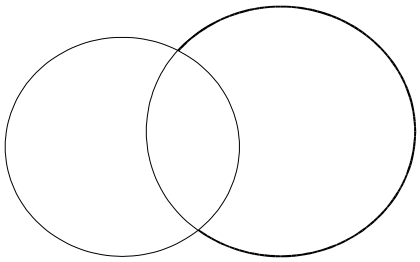


Figure 12 : ray passing through prism

Each plane is tested in turn to find the CI, the candidate interval, the interval of time in which the ray is inside the object. The CI starts at $(-\infty, \infty)$. In the above case the first plane will change it to $(1.5, \infty)$, the second plane to $(1.5, 2.9)$ and plane three, completing the test will give the final CI of $(1.5, 2.3)$

4.9 Lens

The Lens object is based on the boolean class. The lens is created by taking the intersection of two spheres. This gives the necessary details of the lens, the refractive index and the radius of each surface, which is determined by the size of each sphere.



The lens is called in the scene file by, where the lens class acts as a simple, standard ray tracing intersection class.

```
lens
    sphere
    push
        translate 1 0 0
        sphere
    pop
```

4.9.1 Chromatic Aberration

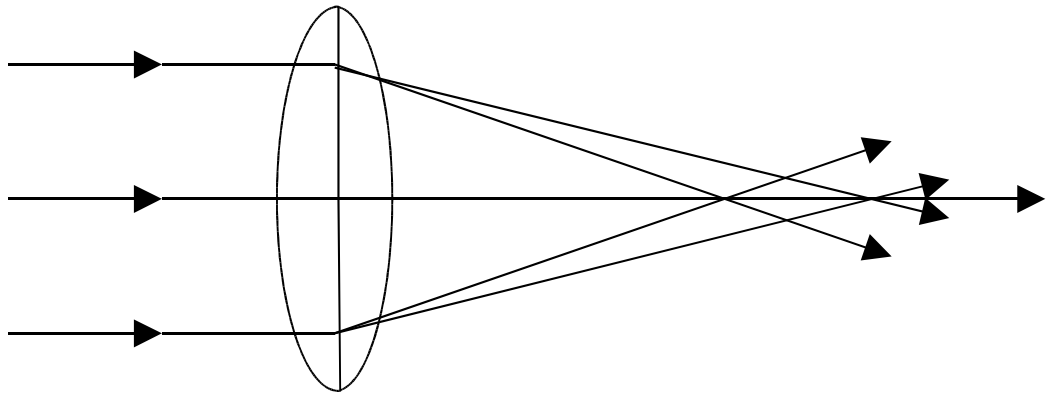


Figure 13: Chromatic Aberration of a lens

A major problem for makers of cameras is the effect of chromatic aberration in a lens. The larger the aperture of the lens, the more light passes through but the more aberration occurs. Aberration is the blurred focus point. In cheap cameras the slit is smaller so it doesn't have to account so much for aberration, but as a result it will sacrifice resolution. A larger aperture will have better resolution but more aberration. In the image below the view through the lens is blurred and the separation of the light is visible.

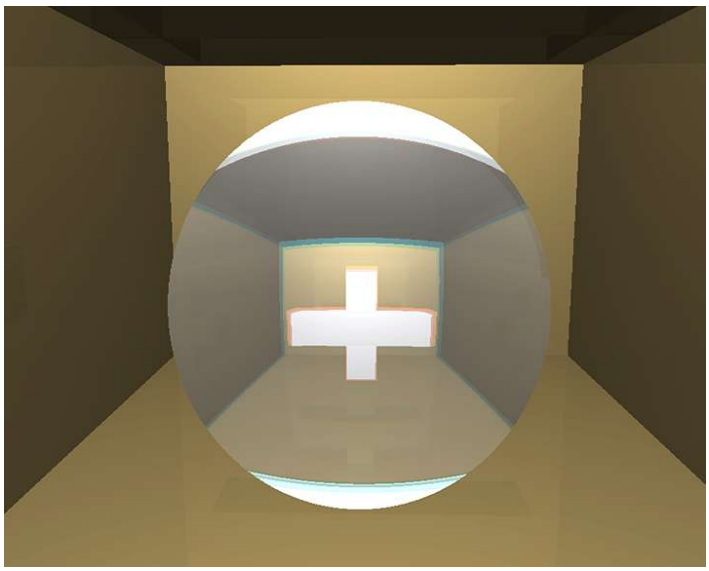


Figure 14: Chromatic Aberration when light split into 3 rays

The spectral ray tracer has taken parts of other peoples code, which are combined to produce a coherent program. The concepts behind the program were taken from Physics ideas of refraction and mostly using the concepts put forward in Roy Hall's book on Illumination and Colour in Computer Generated Imagery. However, all the ideas of how to split and ray trace the spectral light were original.

The program is based on Jon Macey's ray tracer. For the Conversion of spectral data to RGB part of Ian Stephenson's colour management system was used. Jon Macey provided parser code used in SceneReader::loadLight(), SceneReader::loadMaterial() and SceneReader:Tokenize(). The code was then adapted to the format needed for the material and light files. Chris Ward wrote the data stack functions in rgbStack.h and rgbStack.cpp which store the light and material data in SceneReader::loadMaterial and SceneReader::loadLight. Graeme Webster provided code which creates a polynomial function out of tabular data. This is not currently used but the option can be switched on for the change of refractive index. Finally, Jon Macey helped change the code to render the image in tiff format. This was done by storing the rgb colours for each pixel in an array and passing that to the tiff writer.

5. Conclusions and Future work

The ray tracer achieved the goal it was set to. A ray tracer was extended to deal with spectral data of light and to treat that light with more physical accuracy. The colour management part of the program successfully converts spectral data into RGB data for each ray of colour. The ray tracer successfully loops for each ray, adding up the colours for the final image.

Most of the time assigned for the project was spent on the theory, design and implementation of the code. With more time, better images could be created with relatively little effort, but with the time allowed, it was necessary to assign most of the time to the programming of the ray tracer.

The program could be improved by making further improvements to the physical accuracy. Ideally the Fresnel equation would have been implemented to have the reflection and refraction ratio of a surface vary accurately.

When 27 or 81 rays are specified the renders get too long. If this problem was solved, anti aliasing could also be implemented, as is evident in the renders of the cube and the prism in particular. To solve this problem each ray was rendered out separately and composited in Shake. In the Scene::shade() function the loop for each ray was only run for the individual ray, leaving the ray tracer to do all other calculations as normal, only skipping the other ray calculations in the shade function.

Ideally the ray tracer would be developed further to deal with refractive effects better, what was touched on was only a small part of what can be done with it. A rainbow scene was planned which wasn't created and the prism image wasn't completed, because there wasn’t time to work on the shadows. The initial ray tracer just sent out a feeler ray from the hit point to each light, and if the feeler ray hit any object the hit point was declared as being in shadow from that light, and no further colour contributions were added for that light. This is a huge oversimplification of physical reality, and this is an area that ray tracing falls down on in general, because of the way the light is traced through the scene. The problem comes when the object between the hit point and the light is at all transparent. It is not possible to trace a ray directly to the light because when the ray hits the object it will be refracted and will miss the light source it would have hit if there was no refraction. This problem is solved in computer graphics by rendering the image by the process of photon mapping and radiosity, where light is traced from the light source, instead of from the camera. Given more time I would have liked to attempt to solve the problem of shadows by sending out multiple rays from the hit point in shadow towards the light with an offset. If enough extra rays were sent at different angles, a fairly accurate image could be created.

6. Appendix A

Example of the light data used to find the light's overall intensity and colour

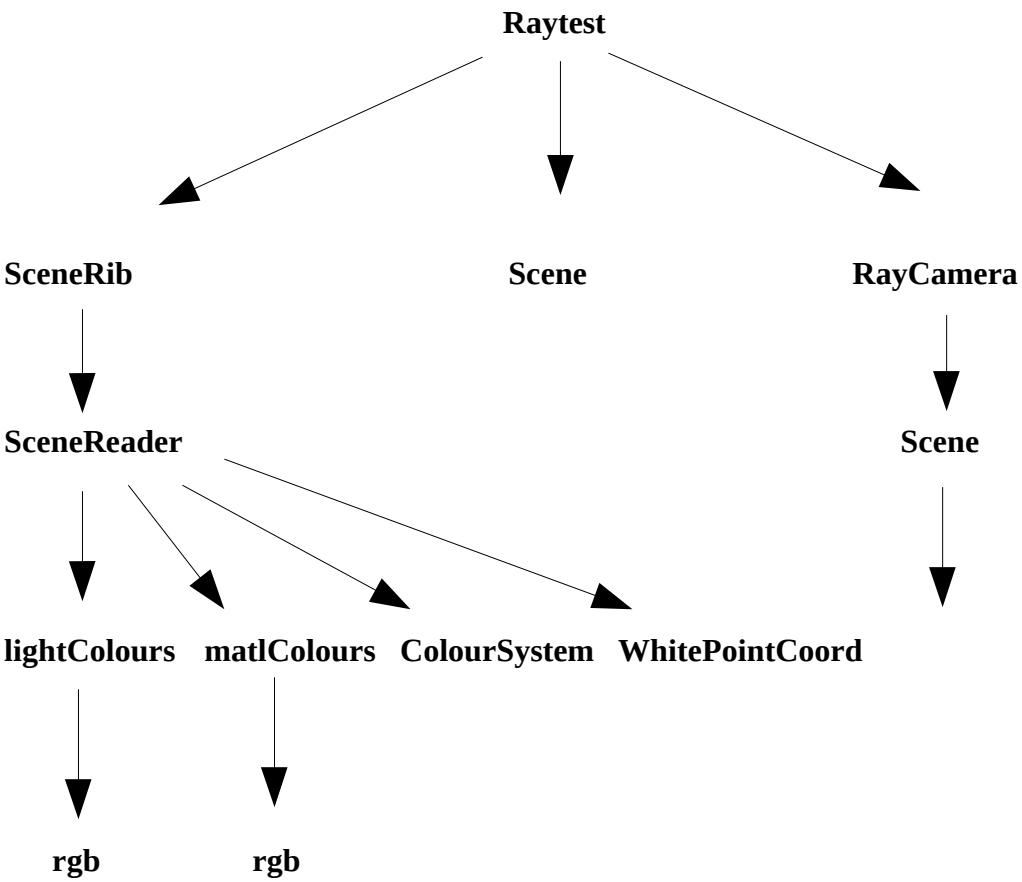
```
filter 2a
{
        intensity 0.85
        colour 81 0.000 0.000 0.000
                  0.000 0.000 0.000
                  0.040 0.023 0.420
                  0.580 0.740 0.784
                  0.827 0.842 0.856
                  0.863 0.870 0.876
                  0.881 0.885 0.888
                  0.891 0.894 0.896
                  0.897 0.898 0.900
                  0.901 0.902 0.902
                  0.903 0.903 0.904
                  0.904 0.905 0.905
                  0.906 0.906 0.906
                  0.907 0.907 0.907
                  0.907 0.907 0.908
                  0.908 0.908 0.908
                  0.909 0.909 0.909
                  0.909 0.909 0.909
                  0.910 0.910 0.910
                  0.910 0.910 0.911
                  0.911 0.911 0.911
                  0.911 0.450 0.000
                  0.000 0.000 0.000
                  0.000 0.000 0.000
                  0.000 0.000 0.000
                  0.000 0.000 0.000
                  0.000 0.000 0.000

}
endfilter
```

Example of the material data used to find the material's overall intensity and colour

```
def StrongOrangeYellow
{
        reflectivity 0.5
        specularExponent 12.8
        transparency 0.1
        diffuseAmount 0.3
        emissiveAmount 0.0
        specularAmount 0.2
        ambience 0.3
        speedOfLight 1.33
        Vnum 1
        freqBreakdown 81 0.0538 0.0552 0.0564
                         0.0576 0.0585 0.0591
                         0.0595 0.0592 0.0596
                         0.0594 0.0591 0.0590
                         0.0592 0.0602 0.0619
                         0.0646 0.0679 0.0720
                         0.0766 0.0817 0.0873
                         0.0934 0.0999 0.1070
                         0.1147 0.1235 0.1337
                         0.1460 0.1619 0.1831
                         0.2107 0.2460 0.2918
                         0.3420 0.3870 0.4292
                         0.4692 0.5054 0.5350
                         0.5577 0.5761 0.5916
                         0.6049 0.6165 0.6265
                         0.6347 0.6415 0.6471
                         0.6518 0.6561 0.6599
                         0.6633 0.6665 0.6694
                         0.6720 0.6745 0.6767
                         0.6786 0.6804 0.6819
                         0.6833 0.6847 0.6860
                         0.6875 0.6890 0.6907
                         0.6924 0.0000 0.0000
                         0.0000 0.0000 0.0000
                         0.0000 0.0000 0.0000
                         0.0000 0.0000 0.0000
                         0.0000 0.0000 0.0000
}
endmaterial
```

Appendix B

File Order



Added or Changed Classes

SceneRib

+light0_position
+objectNumber

assignM()
assignL()
scene()

SceneReader

+*Lfreq
light0_pos

SceneReader()
~SceneReader()
convertrXYZtoRGB()
getLightColour()
Tokenize()
loadLight()
loadMaterial()

ColourSystem

+ *name
+ xRed
+ yRed
+ xGreen
+ yGreen
+ xBlue
+ yBlue
+ xWhite
+ yWhite
+ gamma

WhitePointCoord

+ *name
+ xWhite
+yWhite

Prism

Prism()
hit()

Lens

+ *left
+ *right

Lens()
hit()

RtLight

pos
dir[3]
spot_angle
colour[81]
LightNo
LightMode
rgbColour[81][3]
freq[81]

+ setPosition()
+ setColour()
+ RtLight()
+ RtLight()
+ RtLight
+ Disable()
+ Enable()
+Enable()
+ *getNextLight()
+ Show()
+ getPos()
+ getAngle()
+ getDir()
+ getColour()

Chapter 7. References

Contributions and Changes

Because the project undertaken involves changing code already written, it is important to stress which areas of the code are added to the original ray tracer, and to reference code written by other people for myself.

Files added to original Ray tracer		
.cpp files	.h files	other
SceneRib.cpp	SceneRib.h	All.mtl
SceneReader.cpp	SceneReader.h	Lights.filter
Lens.cpp	Lens.h	Lens.sdl
Prism.cpp	Prism.h	PrismA.sdl
rgbStack.cpp	rgbStack.h	PrismB.sdl
	LightPoint.h	PrismC.sdl
	SpectralLocus.h	Cube.sdl
		Rainbow.sdl
		Cornell.sdl
		Sphere.sdl

Chris Ward, a professional programmer, gave me advice on C++ and debugging. He wrote a large proportion of the new parser used in sceneReader.cpp, and wrote the group of rgbStack classes.

Jon Macey provided the original ray tracer. An attempt to record all the changes has been made, by the insertion of '//!!' at the end of lines which have changed. He also wrote a parser to read in the material and light files, which has now been heavily altered.

Ian Stephenson has provided me with code with changes frequency data into RGB data, which was drawn on to produce the colour management system.

Graeme Webster provided an analytical programme which takes tabulated data and estimates a function of the form $y = a_1 + a_2 x + a_3 x^2 + a_4 x^3 + ... + a_n x^{n-1}$, for n coefficients. This was used to find a formula for the change of refractive index with frequency.

Bibliography

[01AIM] *www.aim-dtp.net/aim/technology/cie_xyz/cie_xyz.htm,* accessed 18/07/2005

[02CAM] Campeanu, R.I. & McFall, J. D., *"Colour Monitor Calibration Based on CIE Standards"*

[03RIT] *www.cs.rit.edu/ncs/color/t_convert.html,* accessed 18/07/2005

[04DOB] Dobson, K. et al, 1997 *"Physics"*, Collins Educational, 77-85 Fulham Palace Road, London

[05HAL] Hall, R., 1989, *"Illumination and Color in Computer Generated Imagery",* Springer-Verlag New York Inc., New York.

[06HIL] *Hill, F. S. Jr., 1990, "Computer Graphics Using Open GL",* Second Edition, Macmillan Publishing Company, New Jersey.

[07HYPA] *http://hyperphysics.phy-astr.gsu.edu/hbase/vision/ciecal.html* ,accessed15/08/2005

[08HYPB] *http://hyperphysics.phy-astr.gsu.edu/hbase/vision/cie1976.html#c1,* accessed 05/08/2005

[09KIN]King, J. C., *"Why Color Management",* Adobe Systems Incorporated.

[10KEN] *Kenton, F., 1989."Prisms and Rainbows: A Dispersion Model for Computer Graphics"*. Proceedings of Graphics Interface 1989.

[11KOL] Kolb, C. et al., 1995. *"A Realistic Camera Model for Computer Graphics"*. Proceedings of SIGGRAPH '95, ACM SIGGRAPH, 1995, pp. 317-324.

[12STE]Stephenson, I. (Ed.) 2005, *"Production Rendering, Design and Implementation",* Springer London Limited 2005, USA.

[13WIL] Wilkipedia*, http://en.wikipedia.org/wiki/Ray_tracing, a*ccessed 01/09/2005