CSC4140 Assignment VII

Computer Graphics

May 23, 2022

3 - Spectral Ray Tracing

This final is 30% of the total mark.

Strict Due Date: 11:59PM, May 23^{th}, 2022

Student ID: 118010141

Student Name: Jingyu Li

This assignment represents my own work in accordance with University regulations.

Signature: JINGYU LI

Contents

1 Overview

This final project aims to extend the knowledge of the course and realize advanced computer graphic features and finish a series of tasks. I choose the topic 3 - **Realize Spectral Ray Tracing** in this project. Ray-tracing technique is vital in computer graphics field since it compensates the weakness of rasterization. Ray-tracing is slow but can obtain a high-quality realistic picture. It usually works off-line instead of real-time rendering. The ray-tracing process really consumes a huge amount of CPU time and patience!

The current implementation of the ray tracer cannot model dispersion and chromatic aberrations because its light model is not wavelength-dependent. Currently, indices of refraction are constant rather than different for each wavelength. Based on the code of assignment 6 & 7 and the lentester resources on GitHub, I add additional wavelength features on the renderer to make it more powerful. I Implement spectral ray tracing by tracing rays of different wavelength sampled using the human eye's wavelength profile for each color (RGB). By modeling different indices of refraction based on those wavelengths for glass-like materials, I also try to reproduce effects such as the dispersion of light through a prism as well as model chromatic aberrations present in real camera systems with lenses. Additionally, I create wavelength dependent bsdf's and lighting, hoping to model different temperature lights.

The final project is not only a hard coding assignment but also a creation of art. In the following part, I would introduce what I have tried and achieved.

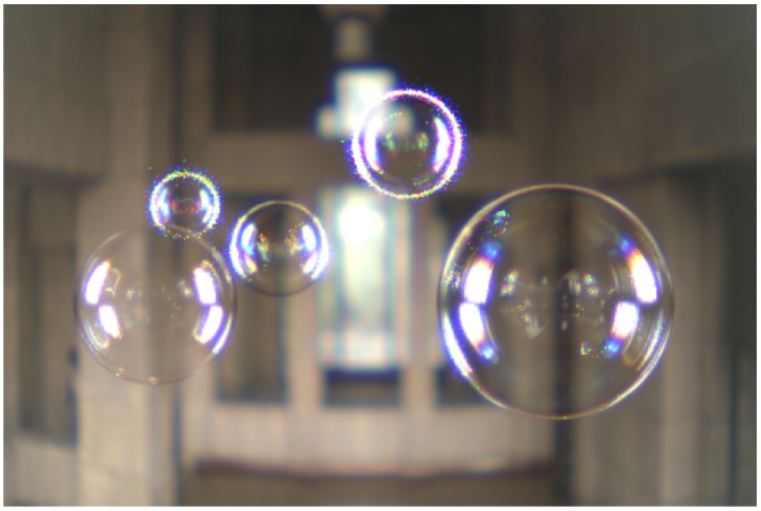

Figure 1: Bubble

2 Task 1: Lentester

2.1 Design and Implementation

This task serves as the first requirement of the third topic: **Change lenstester to also include wavelength argument that the user can set. (mainly for debugging purposes)**. Since we haven't finished relevant project about complex lens, I have searched the Internet and found a resource on GitHub. I implement lenstester based on the code and add a wavelength argument. I have sampled different colors (wavelength) for the rays and realize different refraction effect.

```
int count = 0;
int size = curr_traces.size();
for (vector<Vector3D> &trace : curr_traces) {
  if (3 * count < size) glColor4f(1, 0, 0, 1);
  else if (3 * count < 2 * size) glColor4f(0, 1, 0, 1);
  else glColor4f(0, 0, 1, 1);
  count++;
  draw_trace(trace);
}
```

Figure 2: lentester code

2.2 Results and Analysis

Run the executable lentester and add a wavelength argument directly to check the result.

```
graphics@Graphics-CUHKSZ:~/Project/CG_final_project/build$ ./lenstester 600
[Lens] Loading lens file # Muller 16mm/f4 155.9FOV fisheye lens
[Lens] Infinity focus depth is 28.6069
[Lens] Close focus depth is 32.0147
[Lens] True focal length is 9.92369
loading:"../lenses/fisheye.10mm.dat"
[Lens] Loading lens file # D-GAUSS F/2 22deg HFOV
[Lens] Infinity focus depth is 51.2545
[Lens] Close focus depth is 63.4623
[Lens] True focal length is 50.3236
loading:"../lenses/dgauss.50mm.dat"
[Lens] Loading lens file # Wide-angle (38-degree) lens. Nakamura.
[Lens] Infinity focus depth is 28.7423
[Lens] Close focus depth is 34.4877
[Lens] True focal length is 21.9877
loading:"../lenses/wide.22mm.dat"
[Lens] Loading lens file # SIGLER Super achromate telephoto, EFL=254mm, F/5.6"
[Lens] Infinity focus depth is 188.692
[Lens] Close focus depth is 258.185
[Lens] True focal length is 249.441
loading:"../lenses/telephoto.250mm.dat"
[LensTester] Fixed point is now (-55.1674,-2.22898)
```

Figure 3: lentester command

The effects are shown as follows.

As can be seen from the results, there are multiple rays emitted from a certain starting point. The light source is defined by user. After passing through the compound lens, the red, green, blue light refract with different angles, showing wonderful visual effects. The lentester is mainly used for debugging the lights and lens.

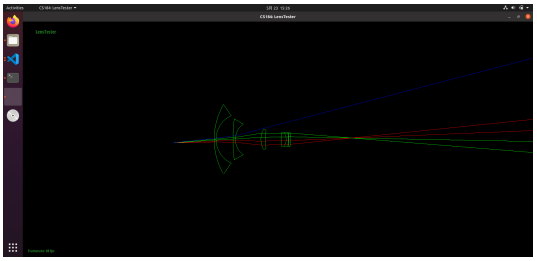

Figure 4: complex len 1

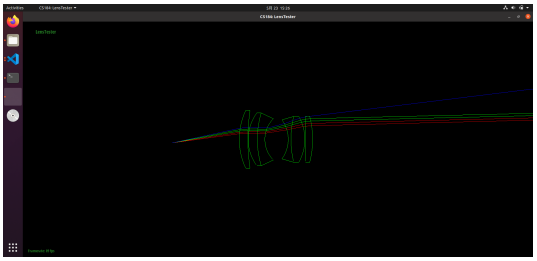

Figure 5: complex len 2

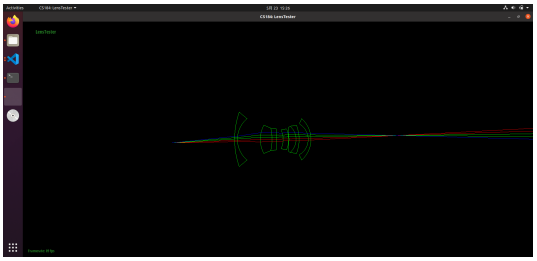

Figure 6: complex len 3

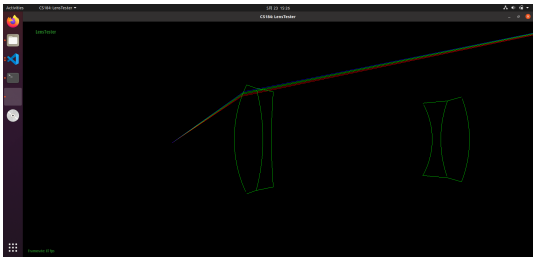

Figure 7: complex len 4

3 Task 2: Wavelength Argument

3.1 Design and Implementation

The requirement is: **Refactor code so that rays have a wavelength argument that can be passed in and checked as well as that functions that return Spectrum now return a single intensity value**. In this part I mainly add the wavelength and color variables in the ray structure. This two terms assign specific wavelength and color channel attributes to the ray. I also change the returned Spectrum to a single double value to represent the radiance for the certain channel. Here shows the code.

```
struct Ray {
  size_t depth;  ///< depth of the Ray

  Vector3D o;  ///< origin
  Vector3D d;  ///< direction
  mutable double min_t; ///< treat the ray as a segment (ray "begin" at min_t)
  mutable double max_t; ///< treat the ray as a segment (ray "ends" at max_t)
  double waveLength;  // unit: nm
  int color; // 0 for Red, 1 for Green, 2 for Blue
```

Figure 8: Add wavelength and color

```
//to be converted to wavelength dependent
double PathTracer::estimate_direct_lighting_hemisphere(const Ray &r,
                                                       const Intersection &isect) {…

double PathTracer::estimate_direct_lighting_importance(const Ray &r,
                                                       const Intersection &isect) {…

double PathTracer::zero_bounce_radiance(const Ray &r,
                                        const Intersection &isect) {…

double PathTracer::one_bounce_radiance(const Ray &r,
                                       const Intersection &isect) {…

double PathTracer::at_least_one_bounce_radiance(const Ray &r,
                                                const Intersection &isect) {…

double PathTracer::est_radiance_global_illumination(const Ray &r) {…
```

Figure 9: Change Spectrum to double

4 Task 3: Color channel

4.1 Design and Implementation

The requirement is: **Change raytrace_pixel to ask for multiple ray samples for each color channel, then combine those color channels**. Initially, raytrace_pixel will only sample one ray at a time, in total num_samples times. Now since I add the color channel, we need to sample all the three colors (R, G, B) for a single ray, so the total amount multiplies 3. After sampling I combine the three color channels to a Spectrum. Here shows the code.

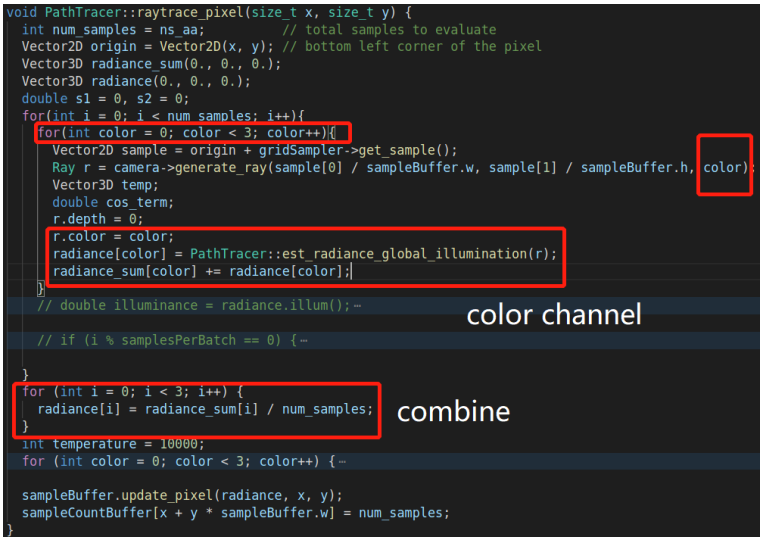

Figure 10: lentester code

5 Task 4: Sample wavelength

5.1 Design and Implementation

The requirement is: **Change camera.generate ray to take in a color channel argument and sample that color channel's wavelength distribution (Gaussian) to change the ray's wavelength**. In this step when we generate a ray, I approximate the wavelength according to normal distribution : Red N(600, 25), Green N(550, 25), Blue N(450, 15), and randomly sample from the three distributions to get a 'red', a 'green' and a 'blue' wavelength.

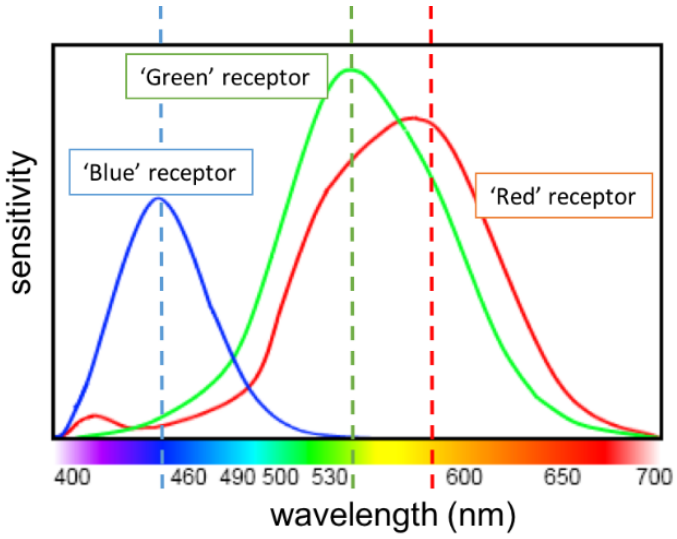

Figure 11: Color Sampling

The relevant codes (in camera.cpp and camera_lens.cpp) are shown as follows.

```
Ray ray(pos, world_coor.unit());
ray.min_t = nClip; ray.max_t = fClip;
ray.color = color;

unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
std::default_random_engine gen(seed);
std::normal_distribution<double> R(600.0, 25), G(550.0, 25), B(450.0, 15);
switch (color){
  case 0:
    ray.waveLength = R(gen);
    break;
  case 1:
    ray.waveLength = G(gen);
    break;
  case 2:
    ray.waveLength = B(gen);
    break;
}
return ray;
```

Figure 12: Color Sampling Code

6 Task 5: Lens tracing

6.1 Design and Implementation

The requirement is: **Change lens_camera's tracing through the lens to use the wavelength argument to change indices of refraction when tracing through the lens**. In this part I make use of dispersion function. Dispersion function describes the relationship between wavelength and refractive index of a medium. For a specific type of medium, 6 parameters are required to determine its dispersion function, exhibiting the dispersion power. The function is a decreasing function, where increasing wavelength brings decreasing refraction power. For example, a red light has a larger wavelength, so it bends less than a blue light. In this project I mainly use

SCHOTT_SF (Dense flint) glass material.

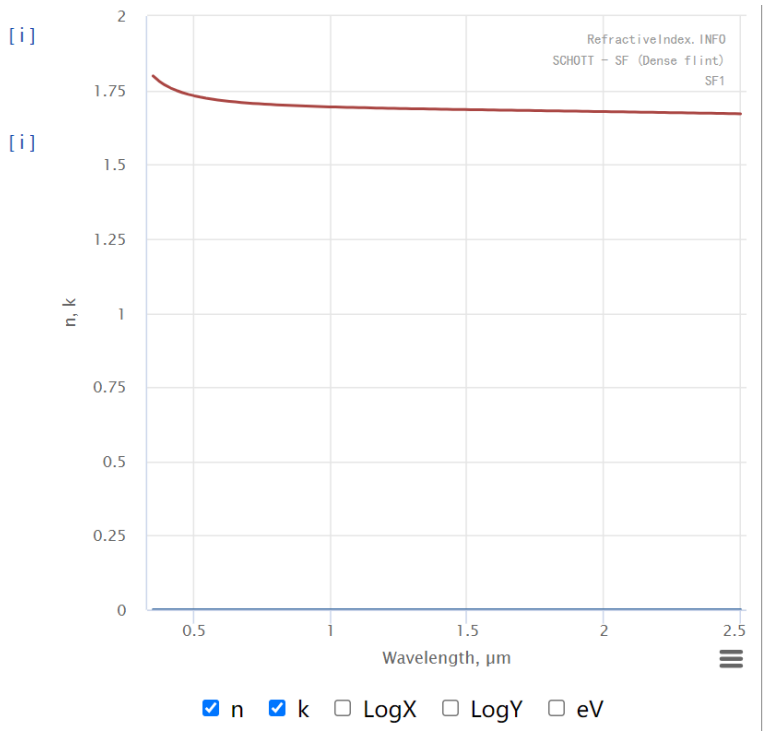

Figure 13: SF dispersion function

The formula and the specific 6 parameters are shown as follows.

$$n^2 - 1 = \frac{A_1\lambda^2}{\lambda^2 - A_2} + \frac{B_1\lambda^2}{\lambda^2 - B_2} + \frac{C_1\lambda^2}{\lambda^2 - C_2}$$

Figure 14: Formula

Dispersion formula

$$n^2 - 1 = \frac{1.55912923\lambda^2}{\lambda^2 - 0.0121481001} + \frac{0.284246288\lambda^2}{\lambda^2 - 0.0534549042} + \frac{0.968842926\lambda^2}{\lambda^2 - 112.174809}$$

Figure 15: Formula with parameters

More specifically, we searched for different types of lens materials to get their dispersion formulas in the given websites, and read the 6 parameters. Here I set the values as fixed in the code.

```
double A1, A2, B1, B2, C1, C2;
double lambda2 = r.waveLength * r.waveLength * pow(0.1, 6);

// SCHOTT SF
A1 = 1.55912923;
A2 = 0.284246288;
B1 = 0.968842926;
B2 = 0.0121481001;
C1 = 0.0534549042;
C2 = 112.174809;

// 1.7174

// Dispersion function
double real_ior = (A1 * lambda2) / (lambda2 - A2) + (B1 * lambda2) / (lambda2 - B2) + (C1 * lambda2) / (lambda2 - C2);
real_ior = sqrt(real_ior + 1);
```

Figure 16: Dispersion Code

6.2 Results and Analysis

The following results show the compound lens effect. I use the pre-set lens file to observe the object and color aberration effect. Since rendering an image costs a large amount of time and we need to adjust the sensor depth for each complex lens, the generated pictures are blurred. I adjust to different lens setting and try my best to find a best sensor depth. But it seems a really difficult work!

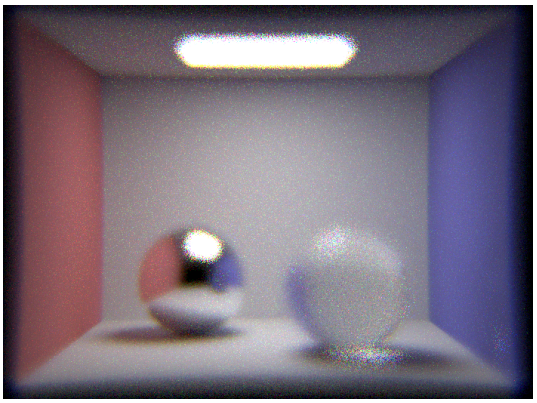

Figure 17: d0

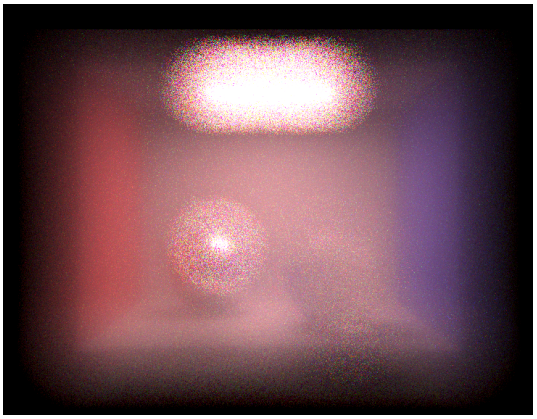

Figure 18: d1

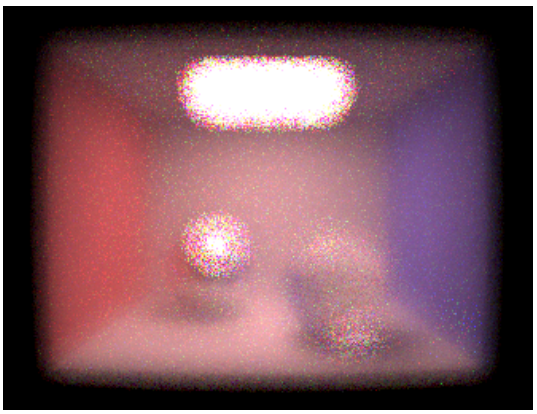

Figure 19: d2

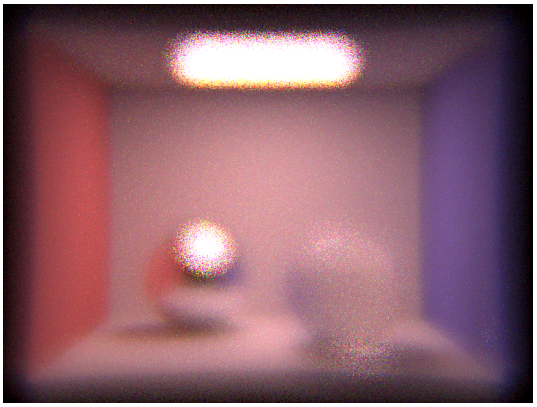

Figure 20: d3

As can be seen from the resulting images, d0 gives out the best effect and visual experience. The temperature is higher and it's clear. From d1 to d4 shows my effort on changing different

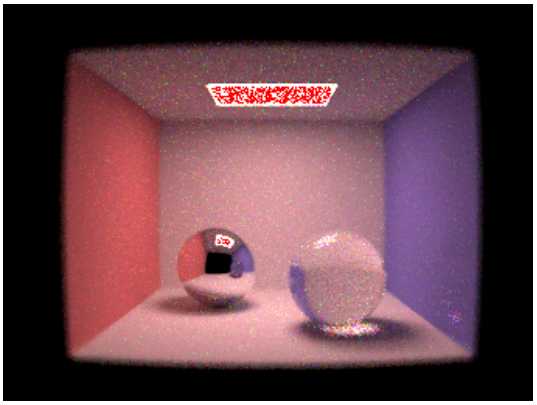

Figure 21: d4

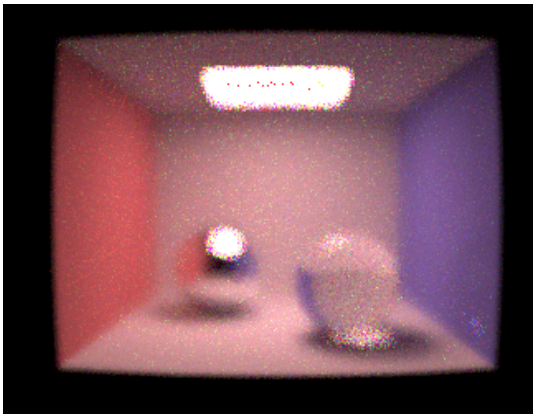

Figure 22: dfish

lens and glass material and adjusting the sensor depth. The images can tell the effect of applying compound lens to check the object. It seems blurred, right. It's hard to adjust! But you can observe the dispersion and color aberration effects!! d4 displays a strange red color aberration effects, this is caused by the low sample rate and only red color channel left. The light is filled with red elements. The last two images shows the usage of fish eye camera, you can see the camera now is not exact square but has some different visualization.

7 Task 6: Color temperature

7.1 Design and Implementation

The requirement is: **Change sample_L of lights to have a wavelength-dependent intensity to simulate different colors of lights (maybe initialize lights with a temperature argument and model them as ideal black bodies to get the intensities for each color)**. A little bit different from the requirement, I set the code in ray_trace_pixel step instead of sample_L step and the effects are generally the same. When the temperature changes, the intensity of the light also changes, giving out the cold-warm effect. I use a ingenious algorithm but not black body simulation here to convert temperature to RGB intensity and approximate the effect and generate satisfying results. The author observes the black-body data file and fit the function. He finds the internal maths rules and express the temperature-intensity relationship in mathematical way. This is so called reverse-engineering. It is really a simple and effective algorithm.

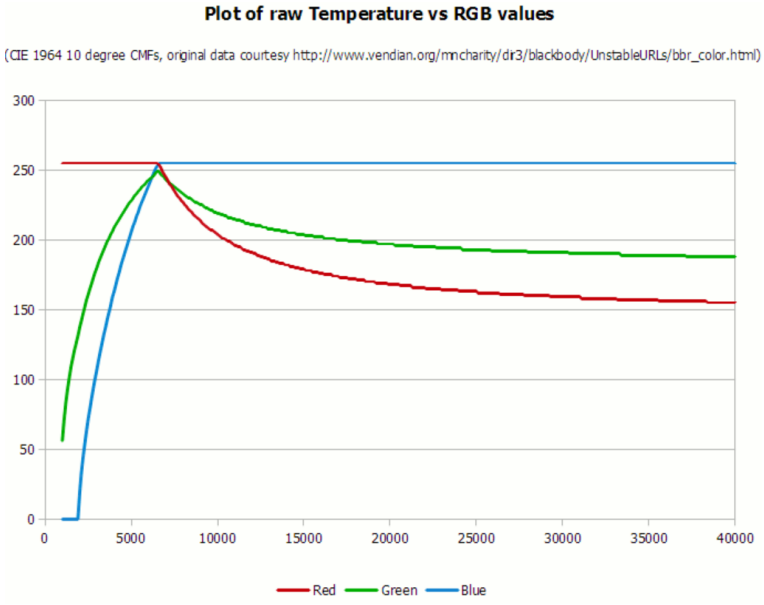

Figure 23: Temperature to RGB intensity

Reference: https://tannerhelland.com/2012/09/18/convert-temperature-rgb-algorithm-code.html

Here shows the relevant code.

```
// temperature: 1000 - 40000
double PathTracer::color_temperature(int temperature, int color) {
  double radiance;
  temperature /= 100;
  switch (color)
  {
  case 0: // R
    if (temperature <= 66) {
      radiance = 255;
    } else {
      radiance = temperature - 60;
      radiance = 329.698727446 * (pow(radiance, -0.1332047592));
      if (radiance < 0) radiance = 0;
      if (radiance > 255) radiance = 255;
    }
    break;

  case 1: // G
    if (temperature <= 66) {…
    } else {…
    break;

  case 2: // B
    if (temperature >= 66) {…
    } else {…
```

Figure 24: Temperature code I

```
int temperature = 10000;
for (int color = 0; color < 3; color++) {
  double radiance_cof = color_temperature(temperature, color);
  radiance_cof /= 255;
  radiance[color] *= radiance_cof;
}

sampleBuffer.update_pixel(radiance, x, y);
sampleCountBuffer[x + y * sampleBuffer.w] = num_samples;
```

Figure 25: Temperature code II

7.2 Results and Analysis

The results are shown as follows. I set the temperature as 1000K, 3000K, 10000K, 30000K, respectively. As can be seen from the four diagrams, the higher the temperature, the colder the atmosphere, which is lighter. While low temperature can create a warm feeling. This is because red color dominates under low temperature while blue color makes a larger influence. The **Temperature to RGB intensity** diagram clearly shows the internal reasons.

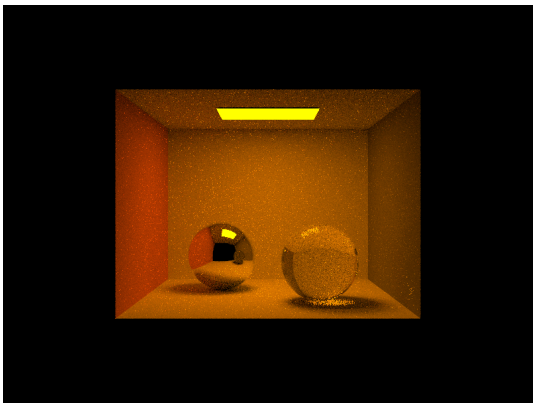

Figure 26: 1000K

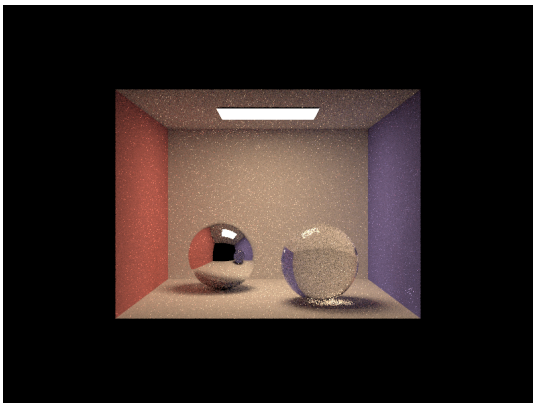

Figure 27: 3000K

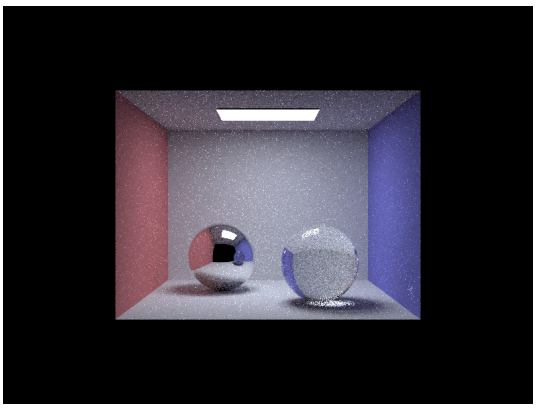

Figure 28: 10000K

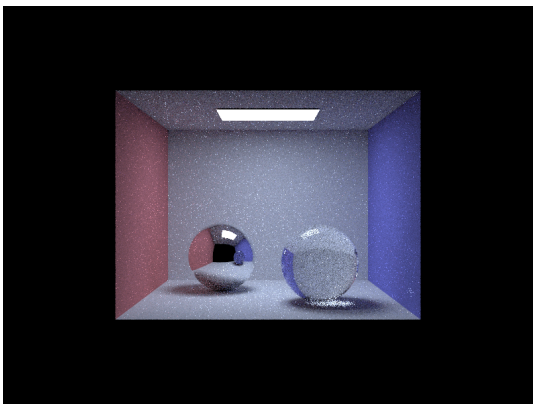

Figure 29: 30000K

8 Task 7: Wavelength dependent bsdf

8.1 Design and Implementation

The requirement is: **Rewrite BSDFs of colored objects to return a wavelength-dependent magnitude instead of a constant spectrum argument**. The modification happens at bsdf.cpp and advanced_bsdf.cpp. Now the f function returns a double scalar instead of a 3-dimension spectrum. Also, the BSDF becomes wavelength dependent. The formula is here! We perform weighed average on the three color channel and make the reflectance accurate

$$albedo_\lambda = \frac{1}{N}\left(albedo_{red}\frac{1}{(\lambda-\mu_{red})} + albedo_{green}\frac{1}{(\lambda-\mu_{green})} + albedo_{blue}\frac{1}{(\lambda-\mu_{blue})}\right)$$

Wavelength-dependent albedo formula

Figure 30: Wavelength dependent formula

The following shows some relevant code. Applying the function to all the bsdf albedo results can generate a wavelength dependent bsdf.

```
double BSDF::wavelength_dependent_BSDF(double wavelength, Vector3D reflectance) {
  double red_cof = 1.0 / (wavelength - 600);
  double green_cof = 1.0 / (wavelength - 550);
  double blue_cof = 1.0 / (wavelength - 450);
  double N = red_cof + green_cof + blue_cof;
  double albedo = (red_cof * reflectance[0] + green_cof * reflectance[1] + blue_cof * reflectance[2]) / N;
  return albedo;
}
```

Figure 31: Wavelength dependent code

8.2 Results and Analysis

I will illustrate the effect using the golden dragon appeared in assignment 6 and 7.

The followings show the normal dragon as well as the dragon adding dispersion and chromatic aberrations effect. I set the parameters of sampling rate (-s), light sample rate (-l) temperature different. Also, the random sampling will also cause the difference. As can be seen from the dragons, their bodies exhibit different dispersion and chromatic aberrations effect. Some can reflect lights and some is darker. In the last dragon, yellow light is obvious, showing the wavelength dependent albedo effect. Yellow wavelength are reflected more.

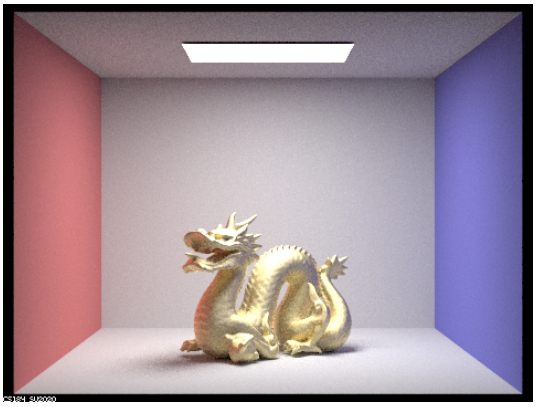

Figure 32: Normal dragon

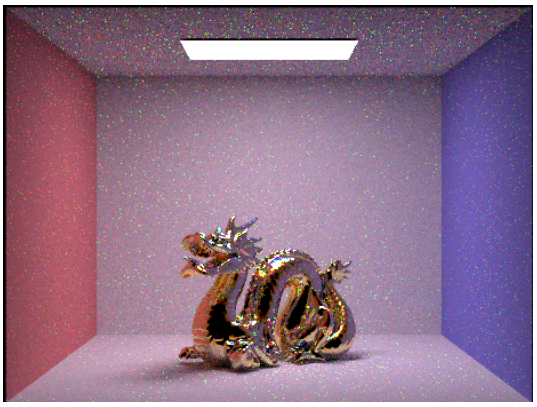

Figure 33: dragon

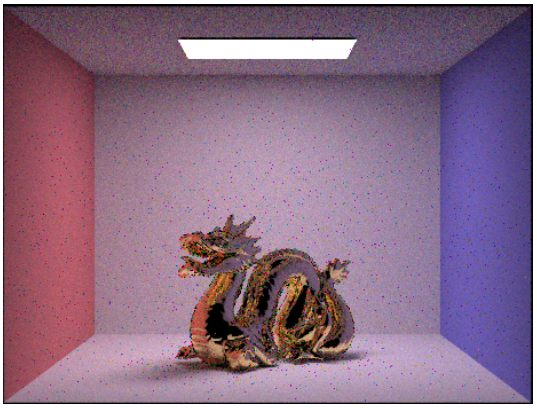

Figure 34: dragon aberration 1

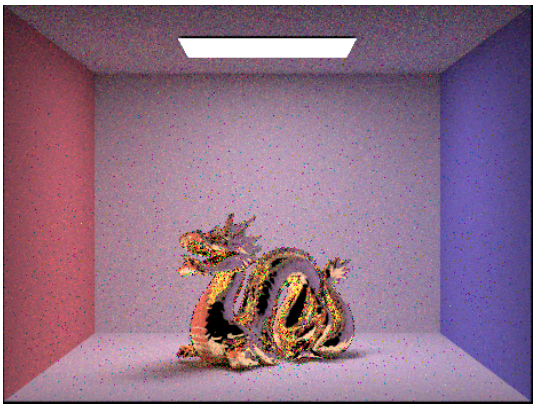

Figure 35: dragon aberration 2

9 Task 8: Glass bsdf

9.1 Design and Implementation

The requirement is: **Rewrite/write glass BSDF to have wavelength-dependent indices of refraction (similar code as lens_camera's tracing)** Reflectance is also commonly called the "albedo" of a surface, which ranges from [0,1] in RGB, representing a range of total absorption(0) vs. total reflection(1) per color channel.

I modify the glass BSDF utilizing the previous task's function and similar code with task 5's function. I use the dispersion function: six parameters of SCHOTT SF glass and the given color channel ray wavelength to calculate the glass bsdf reflectance and then pass to wavelength-dependent albedo formula to obtain the final albedo. Here shows the code.

```
double GlassBSDF::sample_f(const Vector3D wo, Vector3D* wi, double* pdf, const double waveLength, const int color) {
  double A1, A2, B1, B2, C1, C2;
  double lambda2 = waveLength * waveLength * pow(0.1, 6);

  // SCHOTT SF glass
  A1 = 1.55912923;
  A2 = 0.284246288;
  B1 = 0.968842926;
  B2 = 0.0121481001;
  C1 = 0.0534549042;
  C2 = 112.174809;

  // Dispersion function
  double new_ior = (A1 * lambda2) / (lambda2 - A2) + (B1 * lambda2) / (lambda2 - B2) + (C1 * lambda2) / (lambda2 - C2);
  new_ior = sqrt(new_ior + 1);

  double eta;
  if(wo.z > 0){
    eta = 1 / new_ior;
  }
  else{
    eta = new_ior;
  }
  double f = wavelength_dependent_BSDF(waveLength, reflectance);
  double t = wavelength_dependent_BSDF(waveLength, transmittance);

  if(!BSDF::refract(wo,wi,new_ior,waveLength)){
    BSDF::reflect(wo,wi,waveLength);
    *pdf = 1;
    return f / abs_cos_theta(*wi);
  }
```

Figure 36: Wavelength dependent code

9.2 Results and Analysis

I use the sphere to illustrate glass bsdf results. Generally, I modify the ray to be wavelength-dependent and make some progress on showing the dispersion and color aberration effects.

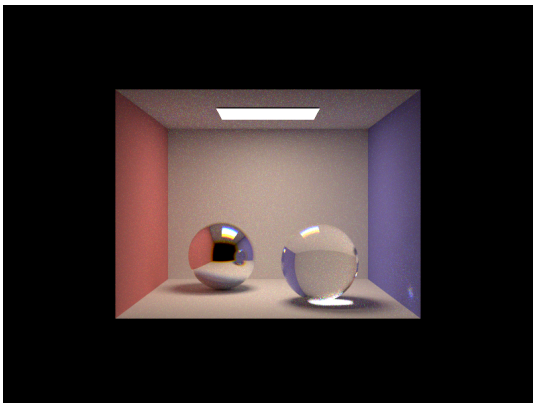

Figure 37: Color aberration 1

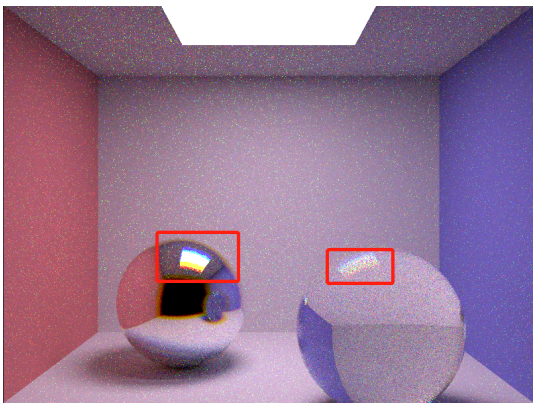

Figure 38: Color aberration 2

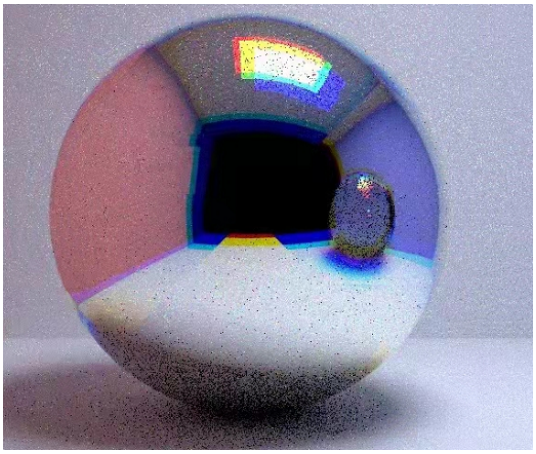

Figure 39: Color aberration 3

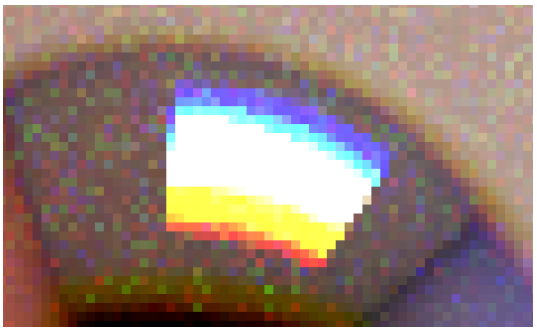

Figure 40: Color aberration 4

As can be seen from these glass spheres. The color aberration effect occurs at the light reflecting area of the ball. There are different colors there! Yellow, blue, green... The light colors are ordered according to the wavelength. I enlarge the image to emphasize the color aberration effect, Beautiful!

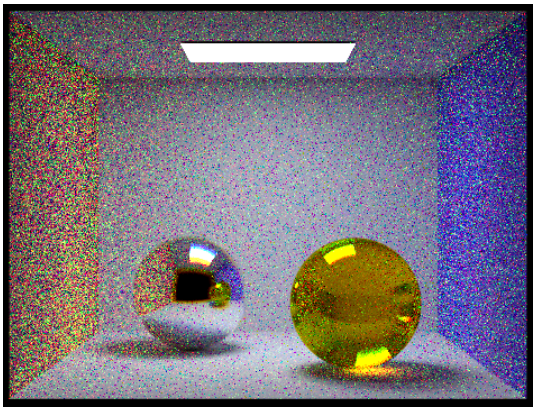

Figure 41: Yellow sphere

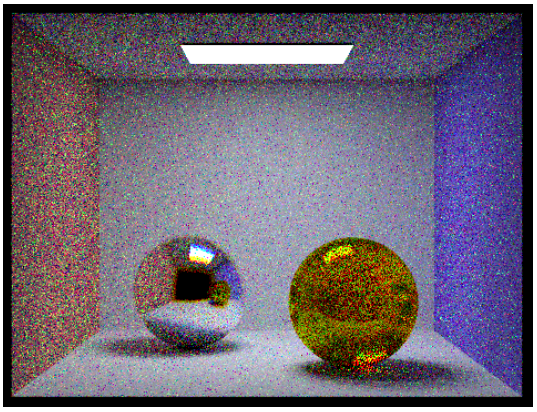

Figure 42: Yellow sphere II

Through another implementation and parameter setting method, I generate two very "yellow" spheres ... It seems that the color aberration effect is too strong! The right sphere becomes complete yellow!

10 Problems and Solutions

In this section, I will share some problems/bugs/eventful debugging experience when finishing this assignment.

(a) Fail to find lentester - GitHub

(b) The unit of wavelength mistake - mm -> nm

(c) Forget to iterate all the three color channels when sampling the light - add for loop

(d) Fail to add wavelength and color parameters - discussion with classmates

(e) Fail to adjust an appropriate sensor depth - render a long time

(f) Fail to apply black-body radiation formula - find a wonderful approximation algorithm for replacement

(g) Fail to produce good color aberration effect - explore more about wavelength- dependent

11 Execution

(a) Run **sh ./compile_run.sh** to compile and run

(b) Render with fewer samples reduces the time.

(c) Many results are not shown but the graph can be rendered properly.

(d) Read the code for more details

12 Reference

I would like to express my gratitude to the following students for their help on this project. We conducted friendly and effective discussion, explained each other's questions, raised innovative idea, and successfully handled this project "hand in hand". They are:

1. **Yang Supei**, 118010370
2. **Qi Xiaonan**, 118010238
3. **Wang Tengfei**, 118010307
4. **Zhang Yiwei**, 118010429

I also reference the following sources on the Internet:

1. A valuable report: https://ceciliavision.github.io/graphics/a6/part1
2. Another good report: https://rod-lin.github.io/cs184-final-project/
3. Lentester code source: https://github.com/rod-lin/cs184-final-project/tree/master
4. Material refraction coefficient: https://refractiveindex.info/?shelf=glass&book=SCHOTT-SFpage=SF1
5. A powerful approximation algorithm for temperature - radiance transformation https://tannerhelland.com/2012/ temperature-rgb-algorithm-code.html
6. The research paper:

Resources

1. Prisms and Rainbows: a Dispersion Model for Computer Graphics
2. Iridescent Surface Rendering with Per-channel Rotation of Anisotropic Microfacet Distribution
3. Rendering Iridescent Colors of Optical Disks
4. Derive spectrum from RGB triple
5. soap bubbles 1
6. soap bubbles 2

Other useful links: [1] refractive index [2] refractive indices [3] glassner [4] hyperphysics [5] Morris, Nigel. "Capturing the Reflectance Model of Soap Bubbles." University of Toronto (2003).

Figure 43: Research paper reference

13 Summary

Here I will summarize what I have learned when writing the final project.

(a) Greater comprehension of spectrum ray tracing in computer graphics. For example, wavelength-dependent ray, color aberration, dispersion function, light temperature, compound lens, and so on.

(b) Solve programming problems and have eventful coding time. Although the workload is pretty heavy!!

(c) Object-Oriented programming paradigm, use class to encapsulate.

(d) Document reading and understanding, much material and extended research paper!!

(e) Meaningful time on debugging and coding skill improvement.

(f) Meaningful discussion and encouragement with peers.

(g) Time Scheduling. I should really start earlier before the final week!

(h) Report writing. A high-quality take-away write-up for future review.

(i) This is the last coding project, the last deadline of my undergraduate academic life. After submitting this project I can graduate. Many thanks to the Professor who teaches this wonderful course, helping me learn various topics in computer graphics. Many thanks to TAs who support the course and share the answers to homework. Many thanks to my classmates who celebrate each step of this computer graphics journey with me. Many thanks to all of you!! See you in the future!!!

That's all.