Ray Tracing

Radhika Mattoo, rm3485@nyu.edu

Overview

- I have split up some portions of the homework (sections 1.1 to 1.3) into separate functions in my *main.cpp* file, and concatenated sections 1.4 - 1.6 into one function via boolean parameters.
- To run the code for a section, simply uncomment the corresponding line in the *main()* function:

```
736
737  int main()
738  {
739      // part1();              // 1.1
740      // part2();              // 1.2
741      // part3();              // 1.3
742      // part4(false, false); // 1.4
743      // part4(true, false); // 1.5
744      // part4(false, true); // 1.6
745
746
747      return 0;
748  }
```

- The images created by when running my code are saved to the *build/* directory. For comparison, the images presented in this file are located in the *screenshots/* directory.

Setup

```
git clone --recursive https://github.com/NYUCG2017/assignment-1-radhikamattoo.git

mkdir build

cd build

cmake ../
```

Running

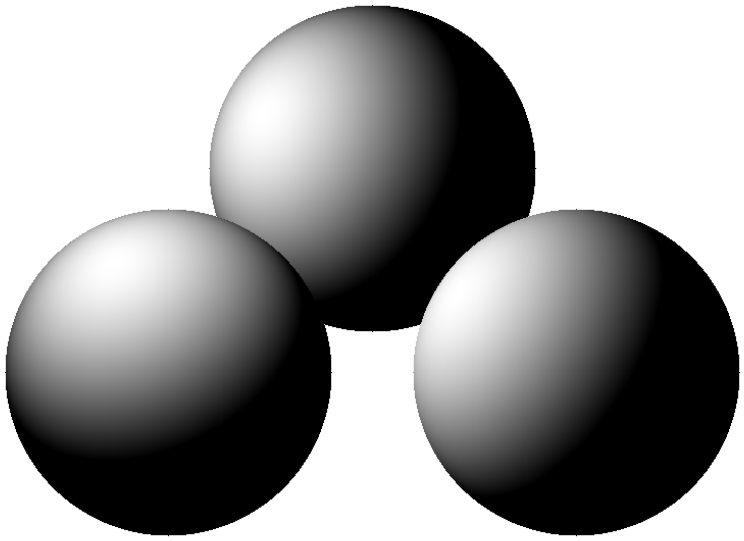
- In the *build/* directory:

```
make && ./Assignment1_bin
```

1.1 Ray Tracing Spheres

- This code is in the function called *part1()*
- I decided to render 3 spheres with **radius = 0.4** located at:
    - (-0.5, 0, 0)
    - (0.5, 0, 0)
    - (0, 0.5, 0)

  with the light positioned at:
    - (-1, 1, 1)
- Now you can see how the position of a given sphere affects its diffuse lighting
- **To run this code**: uncomment the call to *part1()* in *main()* at the bottom of the *main.cpp* file and in the *build/* directory run:
    - *make && ./Assignment1_bin*
- The output image will be saved to the *build/* directory as *part1.png*

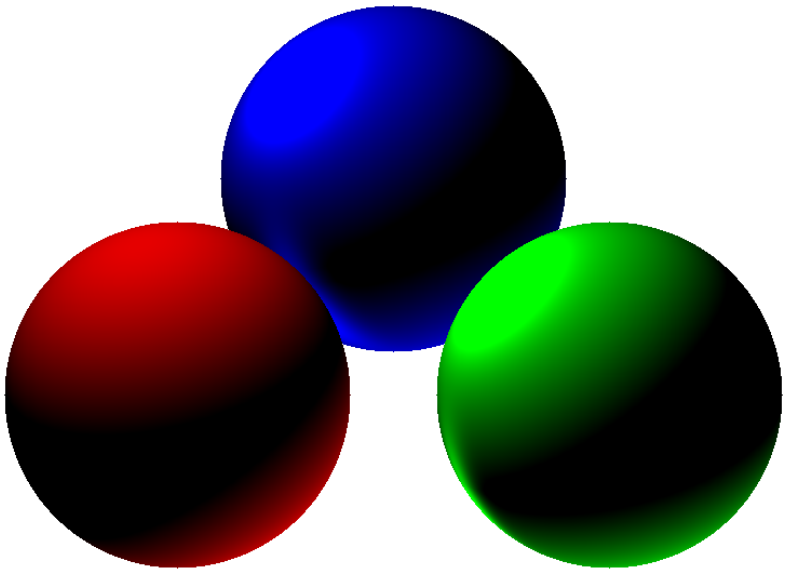


1.2 Shading

- This code is in the function called *part2()*
- I decided to render 3 spheres with **radius = 0.4** located at:
    - (-0.5, 0, 0)
    - (0.5, 0, 0)
    - (0, 0.5, 0)

  with 2 light sources positioned at:
    - (-1, 2, 1)
    - (1, -2, -1)
- The left red sphere has **only** diffuse lighting.
- The middle blue sphere and right green sphere have specular lighting as well, with the **Phong exponent = 10**, and **ambient lighting = 0.01**.

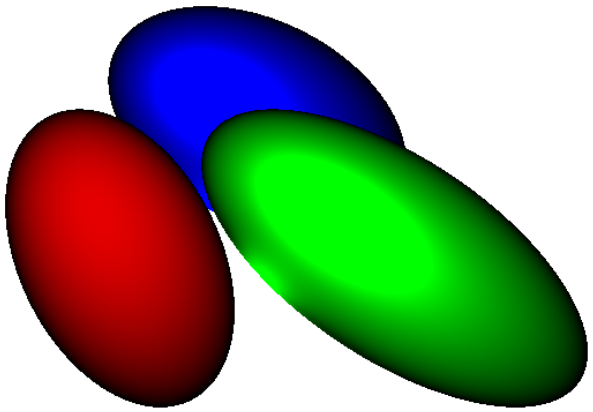
- **To run this code**: uncomment the call to *part2()* in *main()* at the bottom of the *main.cpp* file and in the *build/* directory run:
    - *make && ./Assignment1_bin*
- The output image will be saved to the *build/* directory as *part2.png*



1.3 Perspective Projection

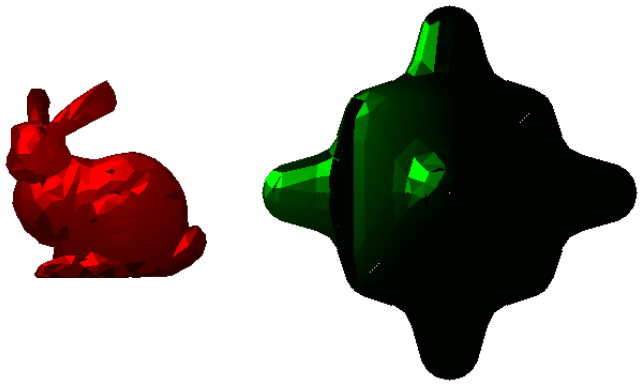
- This code is in the function called *part3()*
- The spheres and lighting setup is exactly the same as part 1.2
- The only difference with perspective projection in ray casting is how you construct the ray; thus, I simply changed the ray construction within my nested for loop, and the rest of the code stayed the same.
- I used a **focal length = 0.5**

- The output image makes sense, as perspective projection renders spheres as ellipses, and a relatively smaller focal length allows rays to be cast out wider than with a larger focal length. If I change the focal length variable in my code to something like 1.0, all of the spheres would be rendered closer than in this example image, as the rays are not cast out as far.
- **To run this code**: uncomment the call to *part3()* in *main()* at the bottom of the *main.cpp* file and in the *build/* directory run:
    - *make && ./Assignment1_bin*
- The output image will be saved to the *build/* directory as *part3.png*



1.4 Ray Tracing Triangle Meshes

- This code is in the function called *part4()* **and takes about 10 minutes to run**.
- To save time in running, I didn't put any background color on non-intersected pixels, like I have in previous sections with rendering spheres.
- The bumpy cube was scaled **down** by a factor of **10**, and the bunny was scaled **up** by a factor of **3**. The bunny was shifted left by **0.15** and down by **0.05**, and the cube was shifted right by **0.1** and up by **0.1**. This can be seen explicitly in the function called *read_off_data()*.
- The reason my green bumpy cube is dark is because of the light position – in the following section with shadows I move the light position to show that the cube is in fact rendered correctly, but looks dark because of the light position and scaling/shifting of the mesh.
- **To run this code**: uncomment the call to *part4(false, false)* in *main()* at the bottom of the *main.cpp* file and in the *build/* directory run:
    - *make && ./Assignment1_bin*
- The output image will be saved to the *build/* directory as *part4.png*

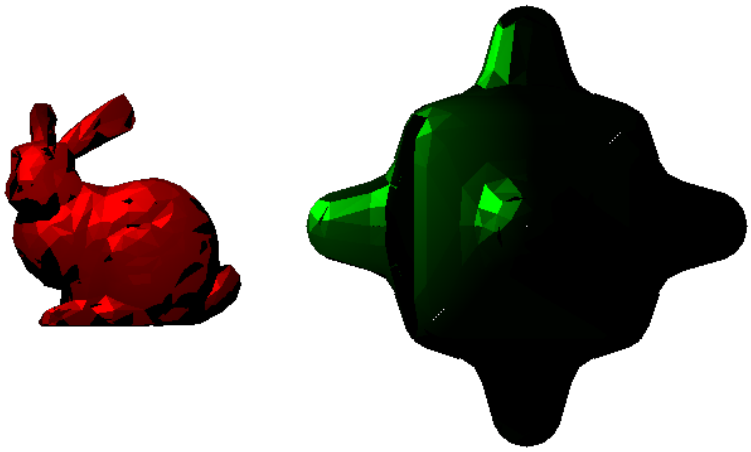


1.5 Shadows

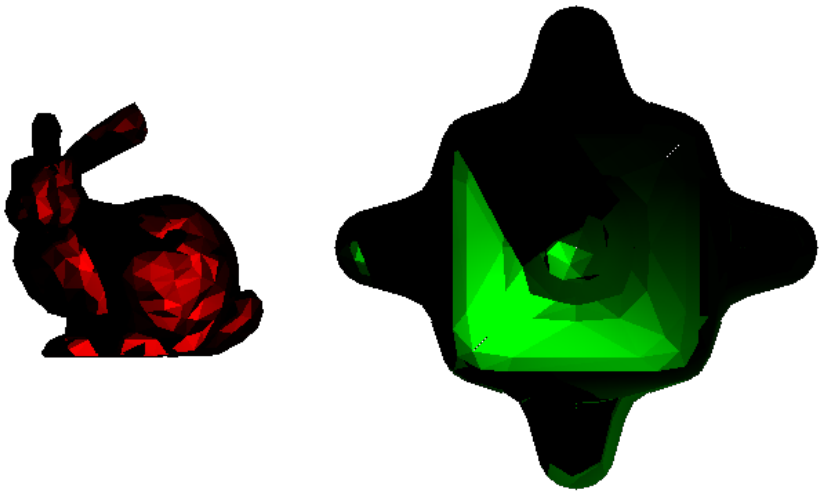
- This code is in the function called *part4()* **and takes 10-15 minutes to run**.
- I have included **2** screenshots with the light positioned at different locations to showcase shadows specifically on the bumpy cube. I'm assuming it looks dark when rendered because of how I scaled/shifted it; it's placement in the z-plane seems to be much closer to the image plane than the bunny, so most of the light hits the bunny instead of the cube.
- The first image below has a light positioned at: *(-1, 1, 1)*. The second image below has a light positioned at *(1, -1, 1)*.
    - By default, the light is positioned at *(-1, 1, 1)*. To produce the second image, simply comment out the first light position and uncomment the second light position within the *part4* function.

```
618     Vector3d light_position(-1,1,1); // first light
619     // Vector3d light_position(1,-1,1); // second light
```

- **To run this code**: uncomment the call to *part4(true, false)* in *main()* at the bottom of the *main.cpp* file and in the *build/* directory run:
    - *make && ./Assignment1_bin*
- The output image will be saved to the *build/* directory as *part5.png*
- Light positioned at *(-1, 1, 1)* (Default) :



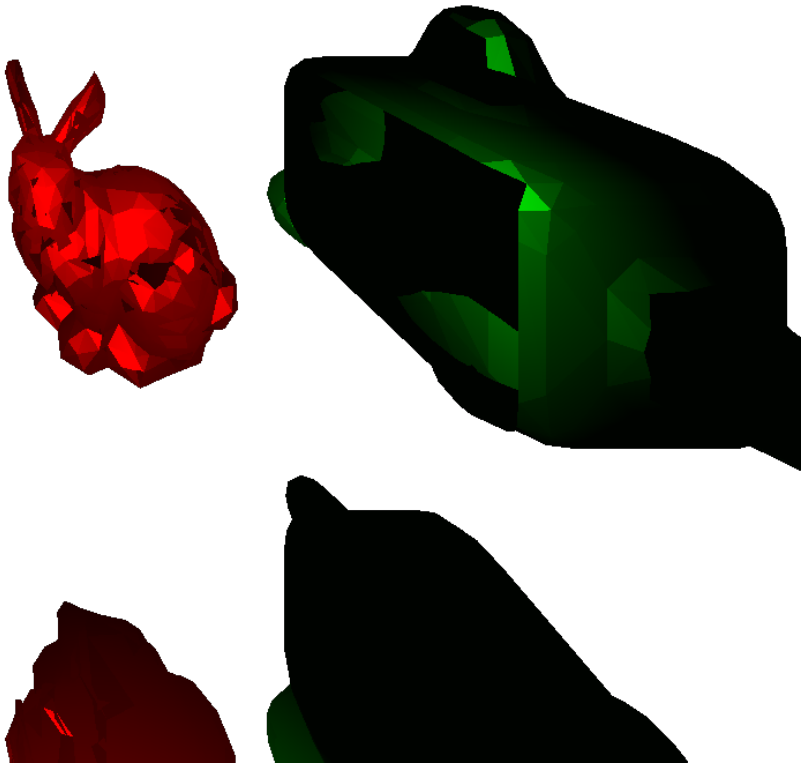
- Light positioned at *(1, -1, 1)* (Need to uncomment line to render):



1.6 Reflections on the floor

- This code is in the function called *part4()* **and takes 15 minutes to run**.
- I've included 2 screenshots using different light positions, just like in the previous section. To render the second screenshot, follow the instructions from the previous section.
- You need to use perspective projection for reflections, because orthographic rays are always parallel to whatever 'floor' you define, so there will never be an intersection. Perspective rays are 'sprayed out' from the image plane, so they will intersect with the 'floor'.
- I decided not to render this section with shadows, as it adds on quite a bit of runtime.
- **To run this code**: uncomment the call to *part4(true, false)* in *main()* at the bottom of the *main.cpp* file and in the *build/* directory run:
    - *make && ./Assignment1_bin*
- The output image will be saved to the *build/* directory as *part5.png*

- Light positioned at *(-1, 1, 1)* (Default) :



- Light positioned at *(1, -1, 1)* (Need to uncomment line to render):