

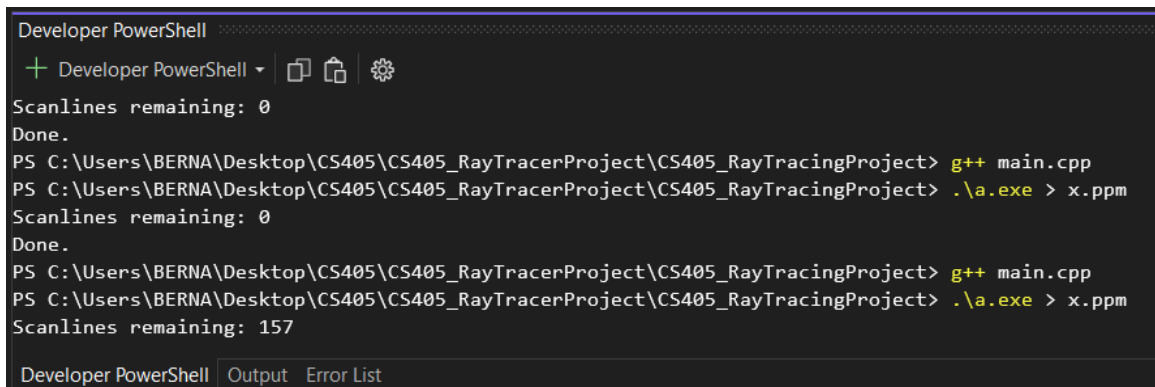
CS 405 Project #1: Ray Tracing

📅 Due Date	@November 6, 2022 11:59 PM
📄 Course	CS 405
☑ Done	<input type="checkbox"/>
📎 Files	Ray Tracing Project.pdf
☰ Notes	
☰ Task	Project

▼ Running the Project File

Open the project and write the following commands to the “Developers PowerShell”

- `g++ main.cpp`
- `.\a.exe > image.ppm`



```
Developer PowerShell
+ Developer PowerShell
Scanlines remaining: 0
Done.
PS C:\Users\BERNA\Desktop\CS405\CS405_RayTracerProject\CS405_RayTracingProject> g++ main.cpp
PS C:\Users\BERNA\Desktop\CS405\CS405_RayTracerProject\CS405_RayTracingProject> .\a.exe > x.ppm
Scanlines remaining: 0
Done.
PS C:\Users\BERNA\Desktop\CS405\CS405_RayTracerProject\CS405_RayTracingProject> g++ main.cpp
PS C:\Users\BERNA\Desktop\CS405\CS405_RayTracerProject\CS405_RayTracingProject> .\a.exe > x.ppm
Scanlines remaining: 157
Developer PowerShell Output Error List
```

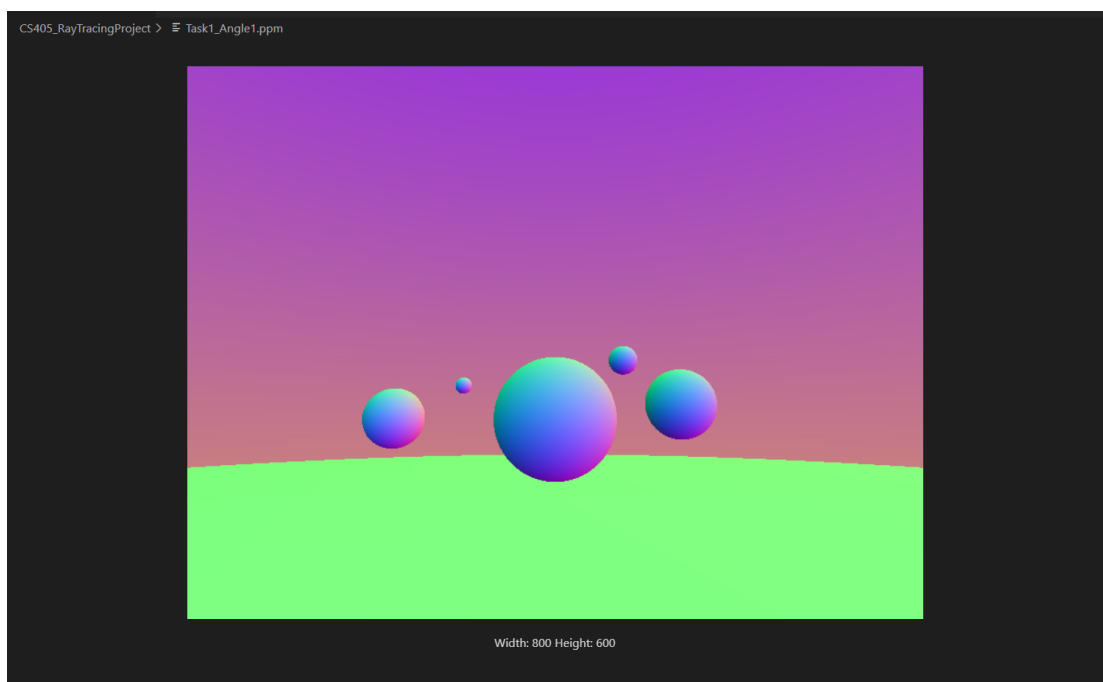
▼ Task 1 [Required]: Basic Scene (15Pt)

▼ Spheres' Positions & Radiuses

- Sphere 1
 - Coordinates: (-1.0, 0.0, -1.0)
 - Radius: 0.175
- Sphere 2

- Coordinates: $(-0.6, 0.2, -1.0)$
- Radius: 0.05
- Sphere 3
 - Coordinates: $(0.0, 0.0, -1.0)$
 - Radius: 0.4
- Sphere 4
 - Coordinates: $(0.5, 0.4, -1.0)$
 - Radius: 0.1
- Sphere 5
 - Coordinates: $(0.95, 0.1, -1.0)$
 - Radius: 0.25

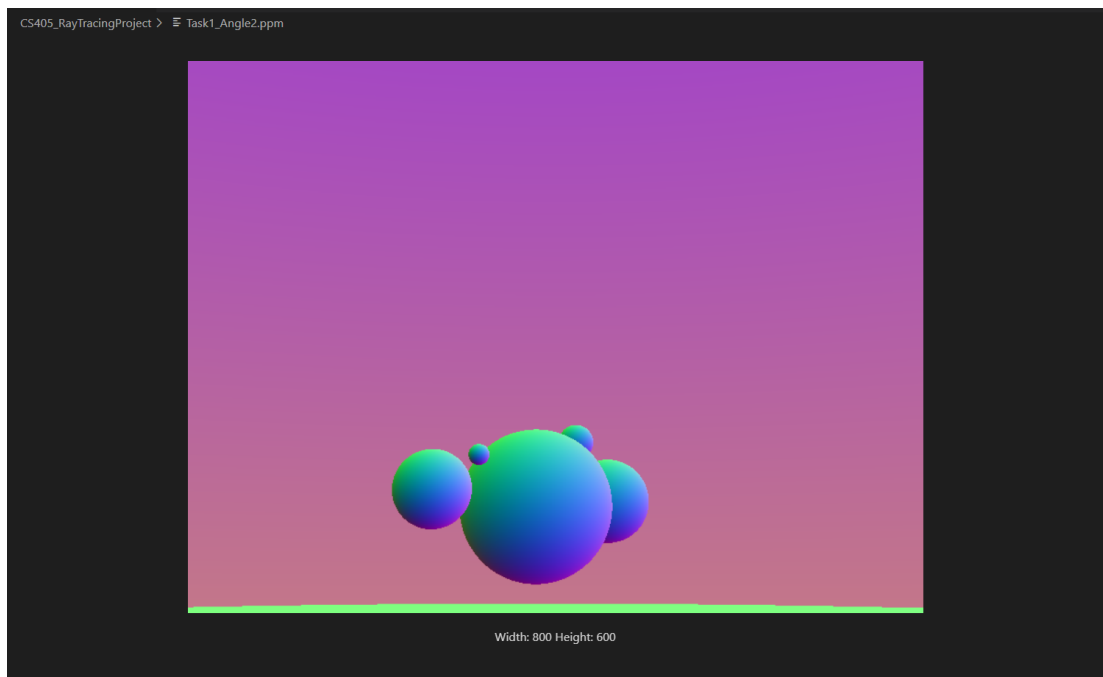
▼ Camera Angle #1



Looks from: $(-1, 0, 2)$

Looks at: $(0, 0.5, -1)$

▼ Camera Angle #2

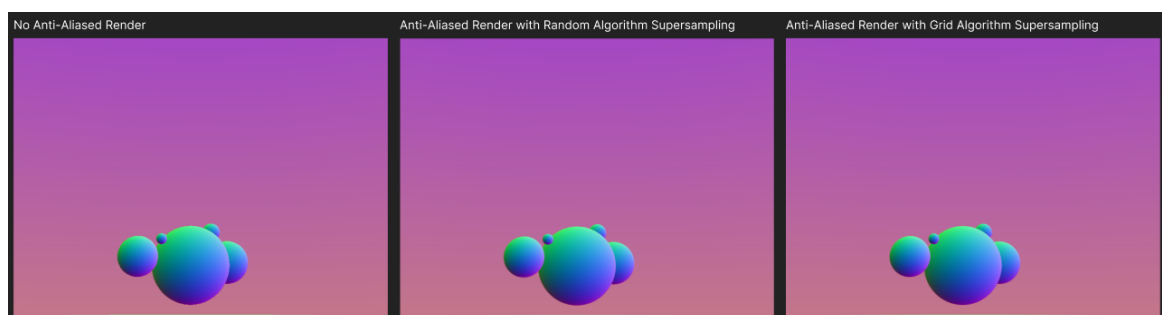


Looks from: $(-4.85, -0.45, 1.5)$

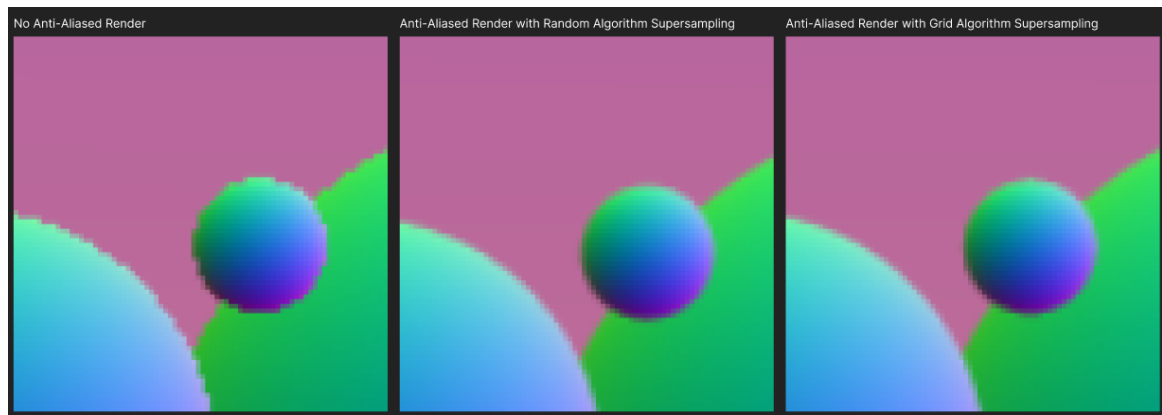
Looks at: $(1.25, 1.25, -1.5)$

▼ Task 2 [Optional]: Anti-Aliasing (10Pt)

Full Scenes



Close-Up Scenes



Anti-Aliasing Render Methods

1. No Anti-Aliasing
2. Anti-Aliasing with Random Algorithm
3. Anti-Aliasing with Grid Algorithm

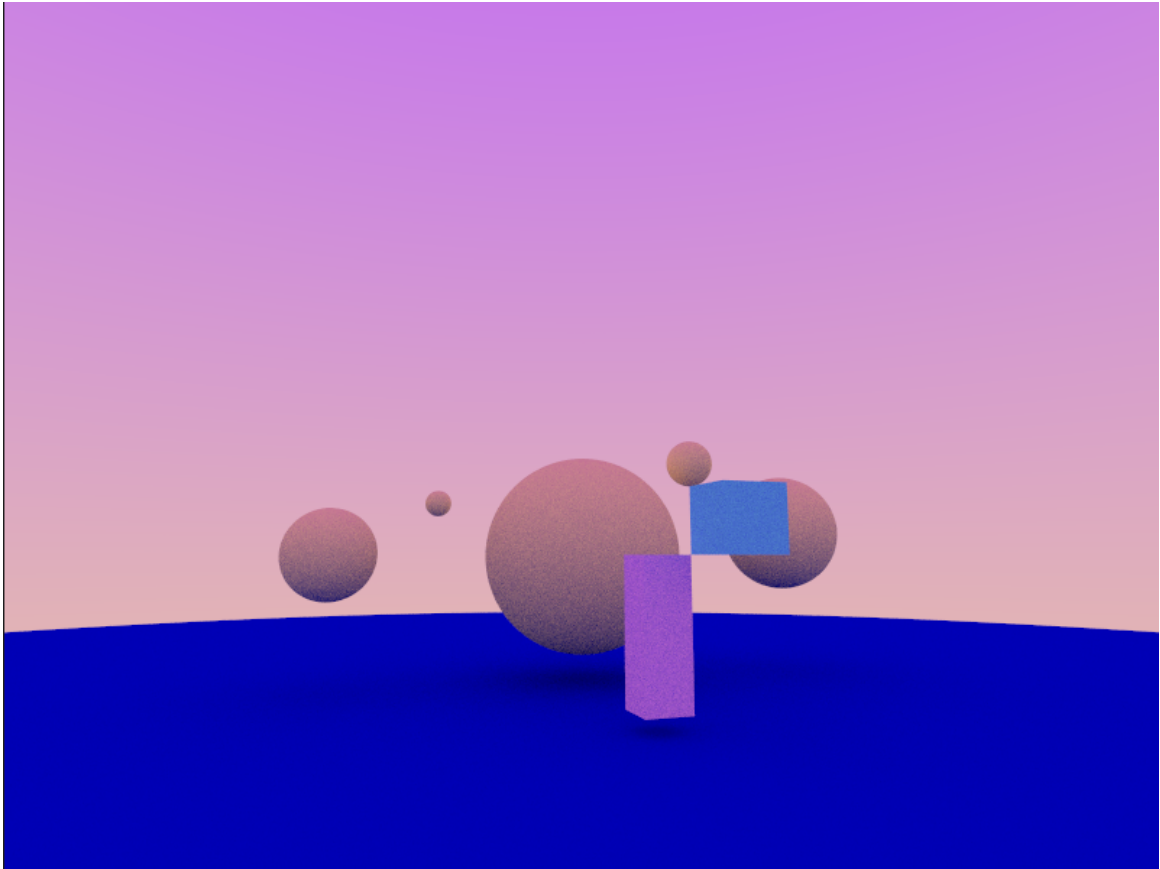
Observations

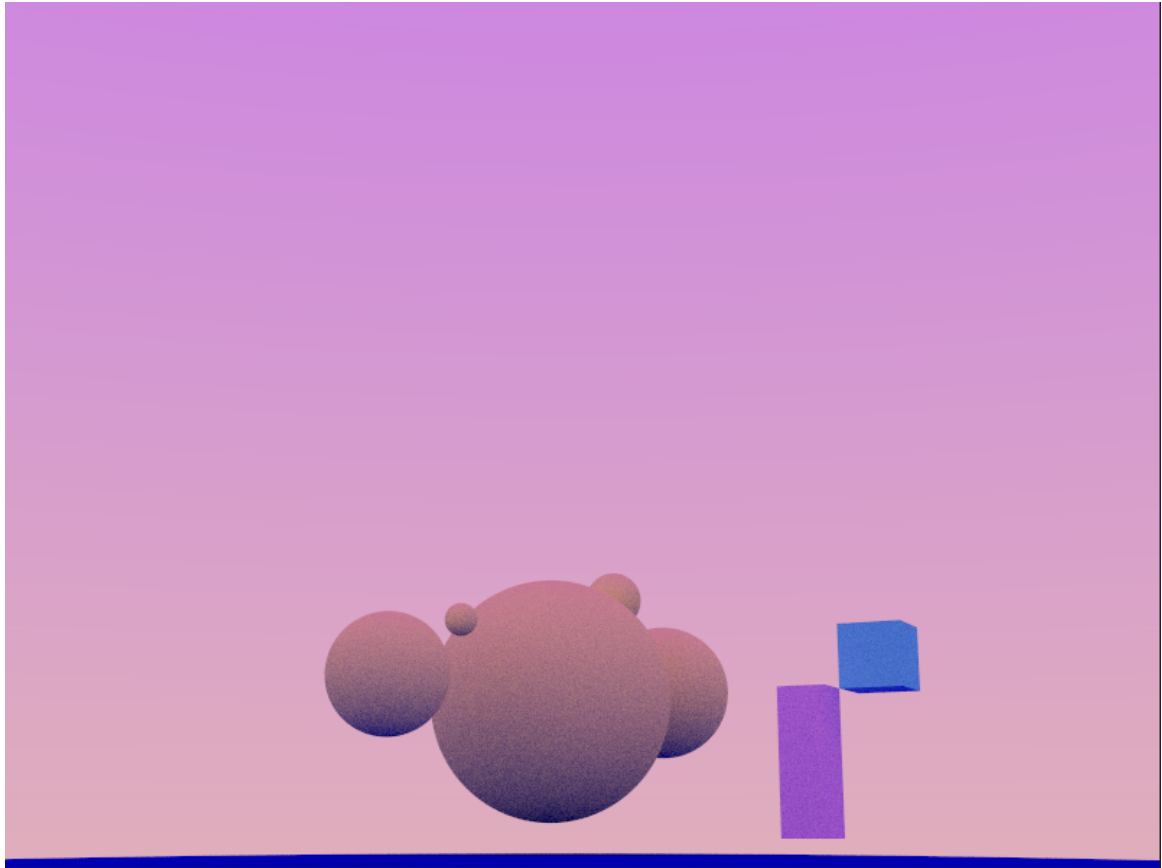
- Anti-Aliasing improves the render quality by smoothing the edges of the objects and make them blend together more with the whole scene.
- Anti-Aliased render provides much better results than the render with **no Anti-Aliasing**.
- **Anti-Aliasing using Grid Algorithm** provides the best results but on the other hand, the rendering time is by far the longest.
- **Anti-Aliasing using Random Algorithm** also provide good results when compared to the render with No Anti-Aliasing. But, its results are not as good as Grid Algorithm.

Conclusion

After making a benefit-cost analysis, **Anti-Aliasing with Random Algorithm** can be the most preferable. Since it takes less time and provides acceptable results. Due to its large render time, grid algorithm is not preferable despite its good results because as the render scenes get more complex the render time of the grid algorithm becomes much more longer.

▼ Task 3 [Optional]: More Shapes (5Pt)





I rendered a blue *cube and purple rectangle prism* besides the spheres for this task. I utilized the classes and code snippets from “Ray Tracing - The Next Week”.

▼ Task 4 [Required]: Diffuse and Metal Materials (25Pt)

▼ Material Properties

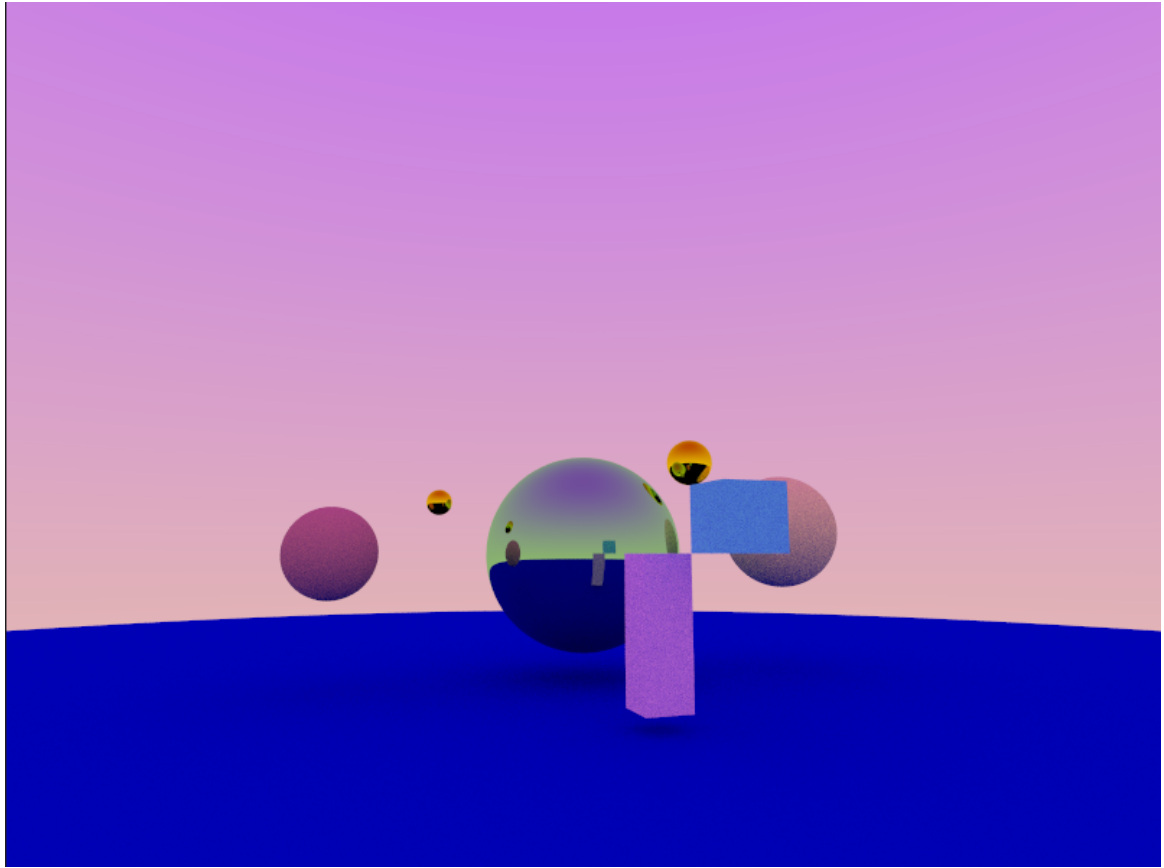
- **Ground Material**
 - material_ground
 - color(0.0, 0.0, 0.6)
- **Metals**
 - metal_gold
 - color(1.0, 0.95, 0.0)
 - metal_green
 - color(0.37, 1.0, 0.37)
- **Metals (With Fuzziness)**

- metal_gold
 - color(1.0, 0.95, 0.0)
 - fuzziness: 0.15
- metal_green
 - color(0.37, 1.0, 0.37)
 - fuzziness: 0.075
- **Lambertians**
 - material_orange
 - color(0.7, 0.3, 0.3)
 - material_aqua
 - color(0.2, 1.0, 1.0)
 - material_pink
 - color(1.0, 0.6, 1.0)
 - material_lambert
 - color(1.0, 1.0, 0.4)

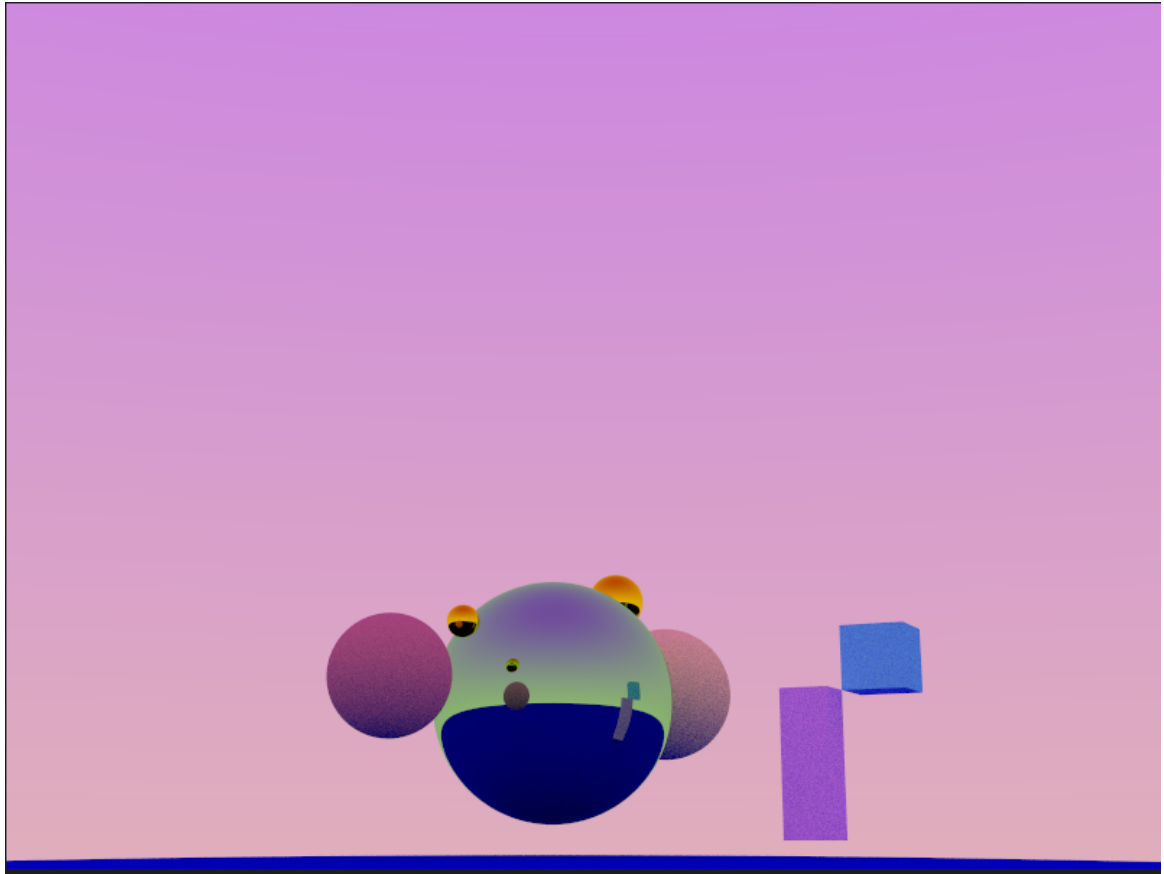
▼ **Object Properties**

- Ground
 - Material: material_ground
 - Coordinates: (0.0, -100.5, -1.0)
 - Radius: 100
- Sphere 1
 - Material: material_orange
 - Coordinates: (-1.0, 0.0, -1.0)
 - Radius: 0.175
- Sphere 2
 - Material: metal_gold

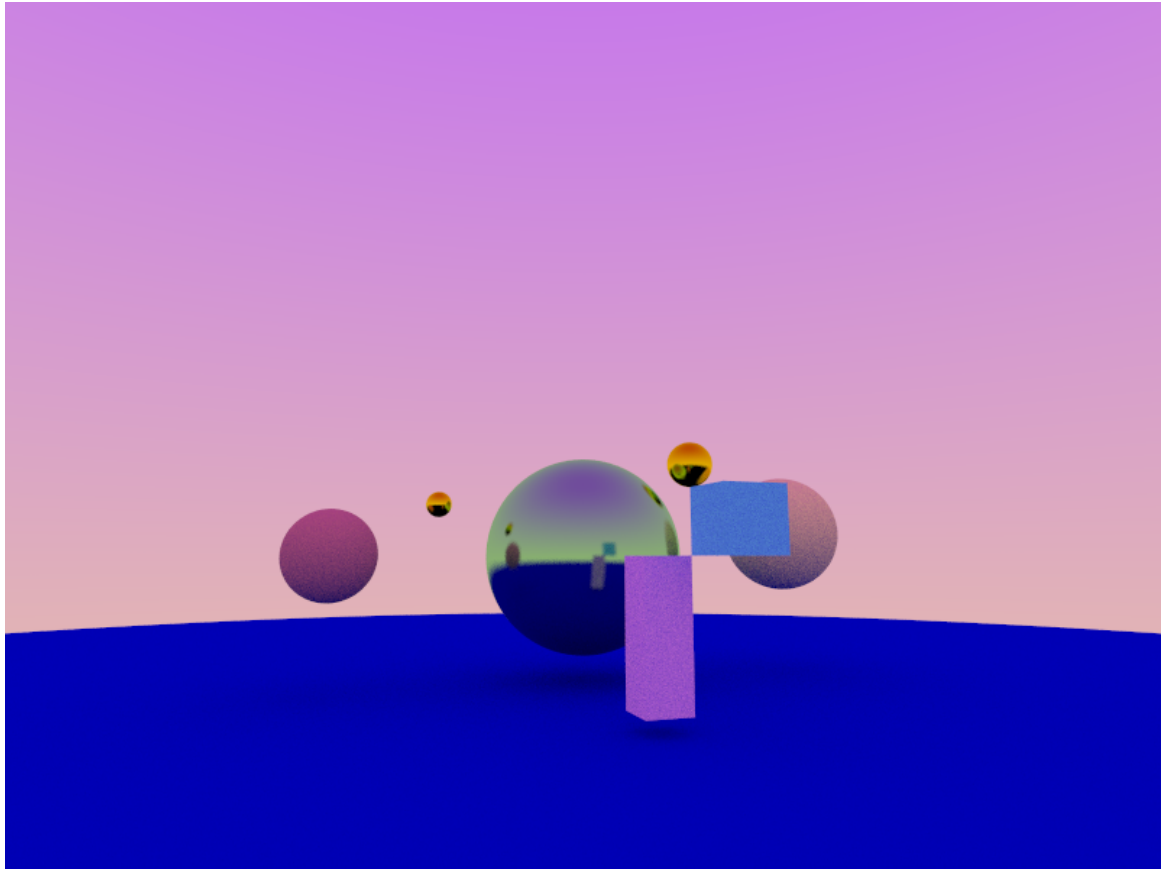
- Coordinates: (-0.6, 0.2, -1.0)
 - Radius: 0.05
- Sphere 3
 - Material: metal_green
 - Coordinates: (0.0, 0.0, -1.0)
 - Radius: 0.4
- Sphere 4
 - Material: metal_gold
 - Coordinates: (0.5, 0.4, -1.0)
 - Radius: 0.1
- Sphere 5
 - Material: material_lambert
 - Coordinates: (0.95, 0.1, -1.0)
 - Radius: 0.25
- Cube
 - Material: material_pink
- Rectangle Prism
 - Material: material_aqua



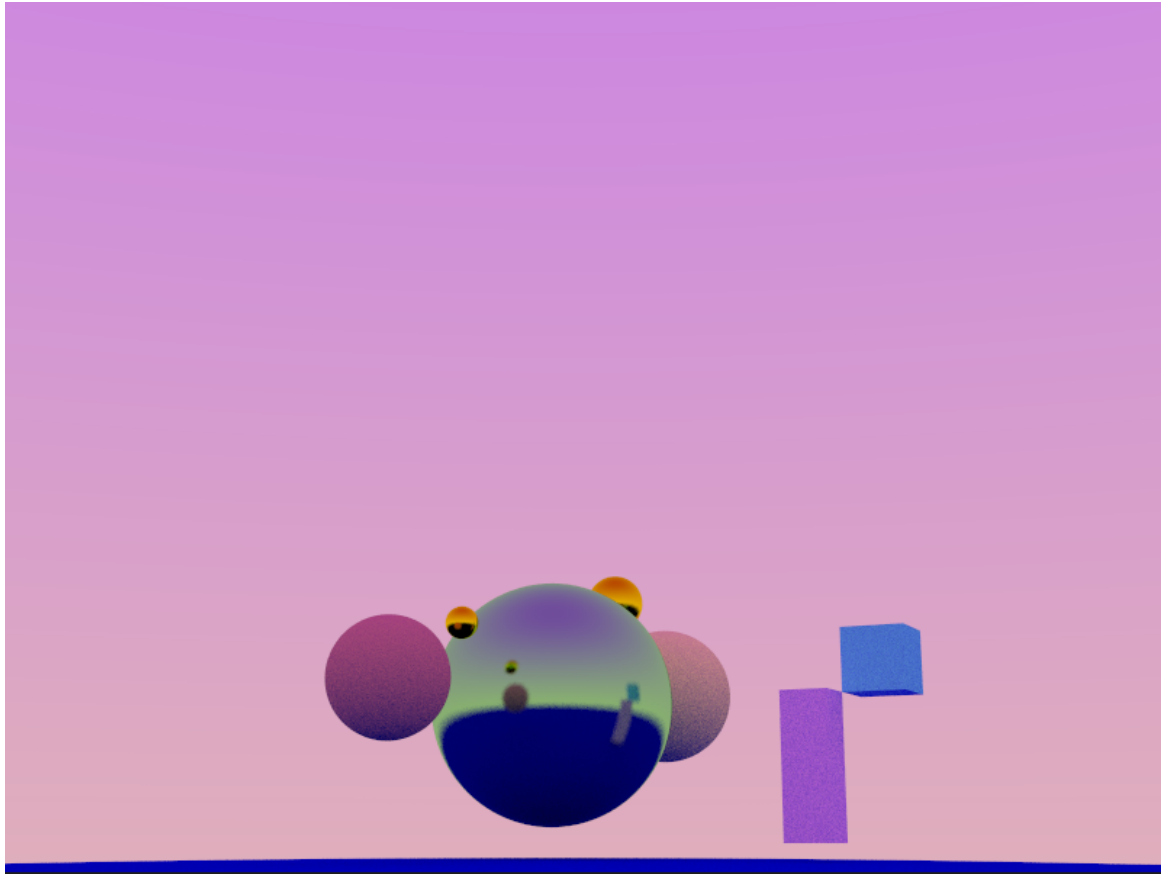
Camera Angle #1 // No Metal Fuzziness



Camera Angle #2 // No Metal Fuzziness



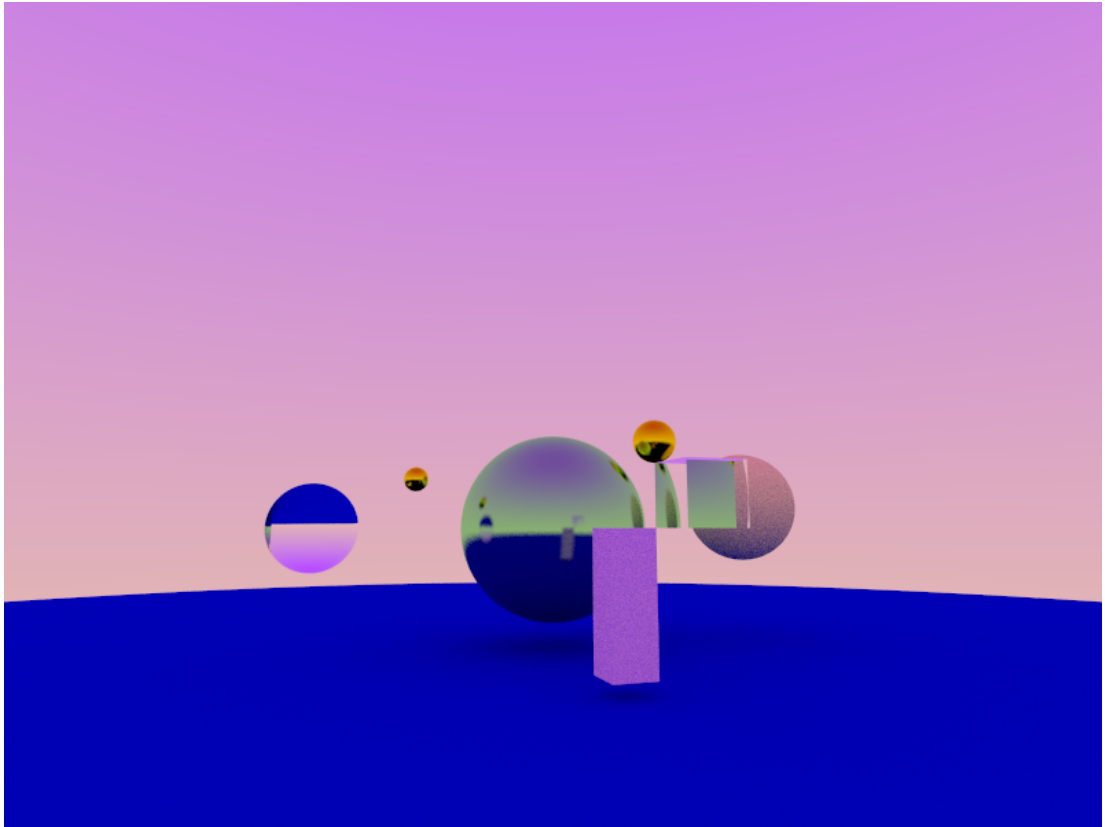
Camera Angle #1 // Metal Fuzziness



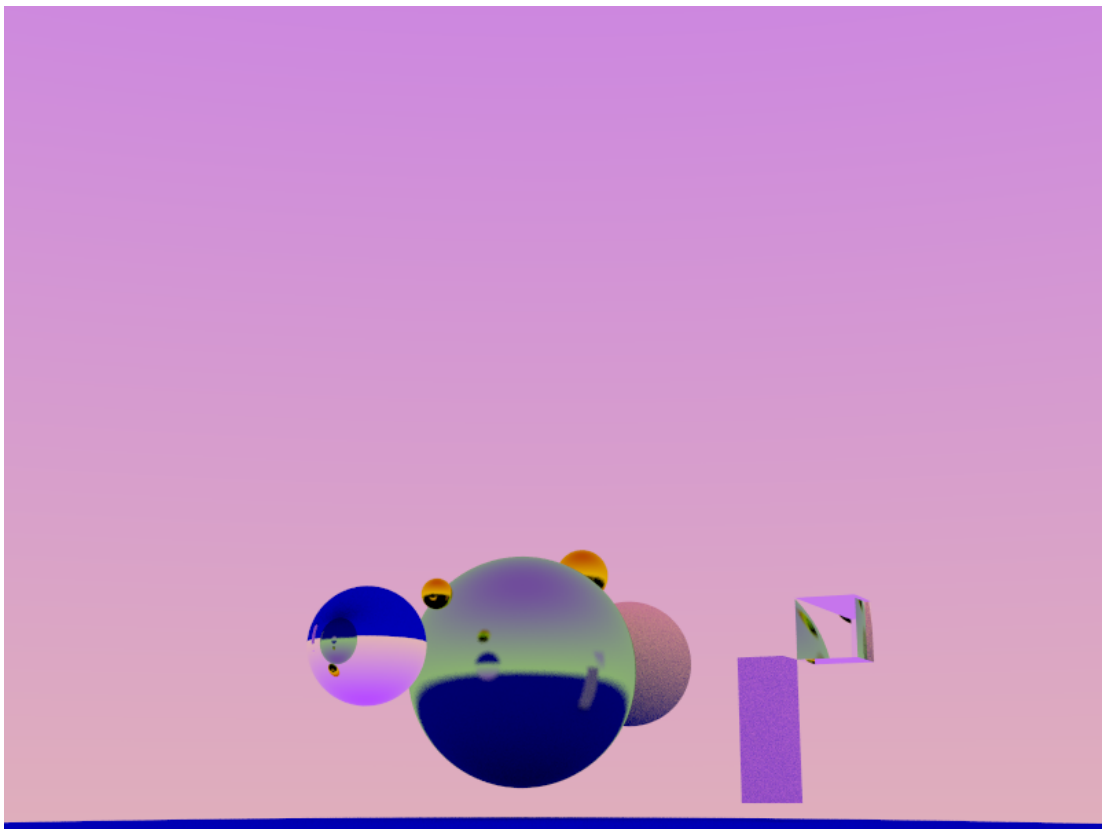
Camera Angle #2 // Metal Fuzziness

▼ **Task 5 [Optional]: Refraction (10Pt)**

▼ **Glass Material (Always Refracting)**

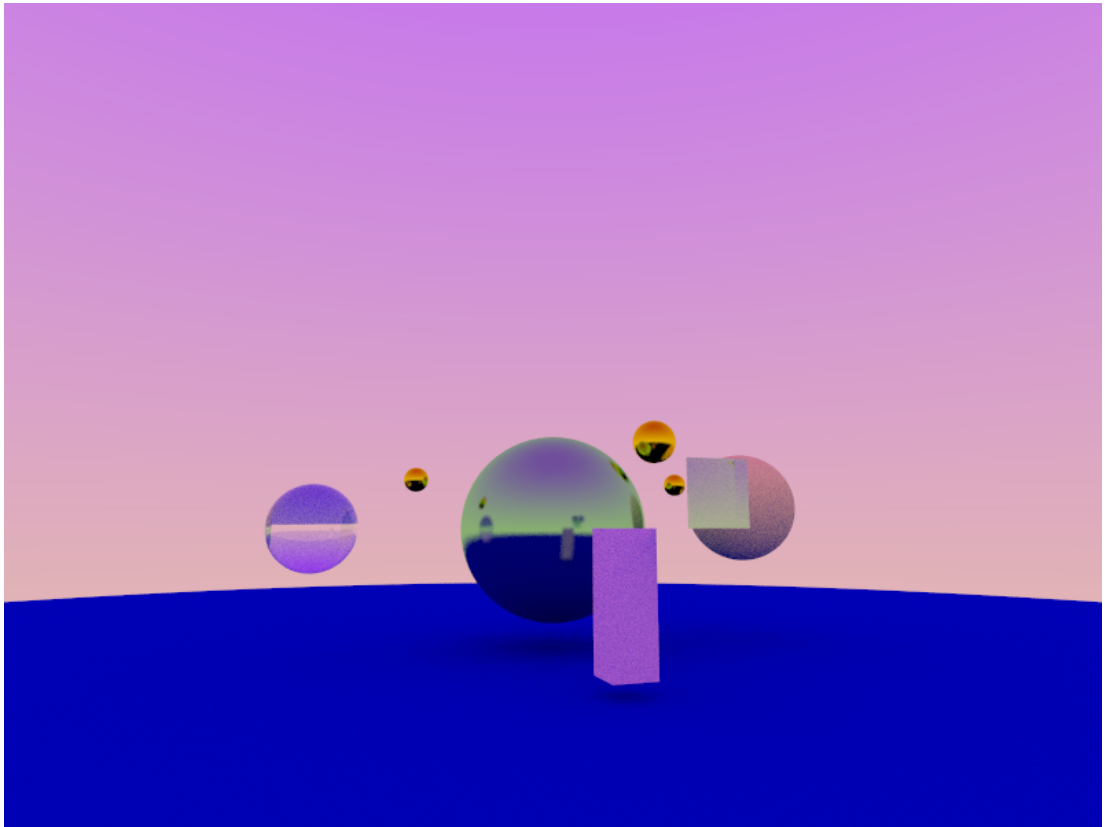


Camera Angle #1 // Always Refracting

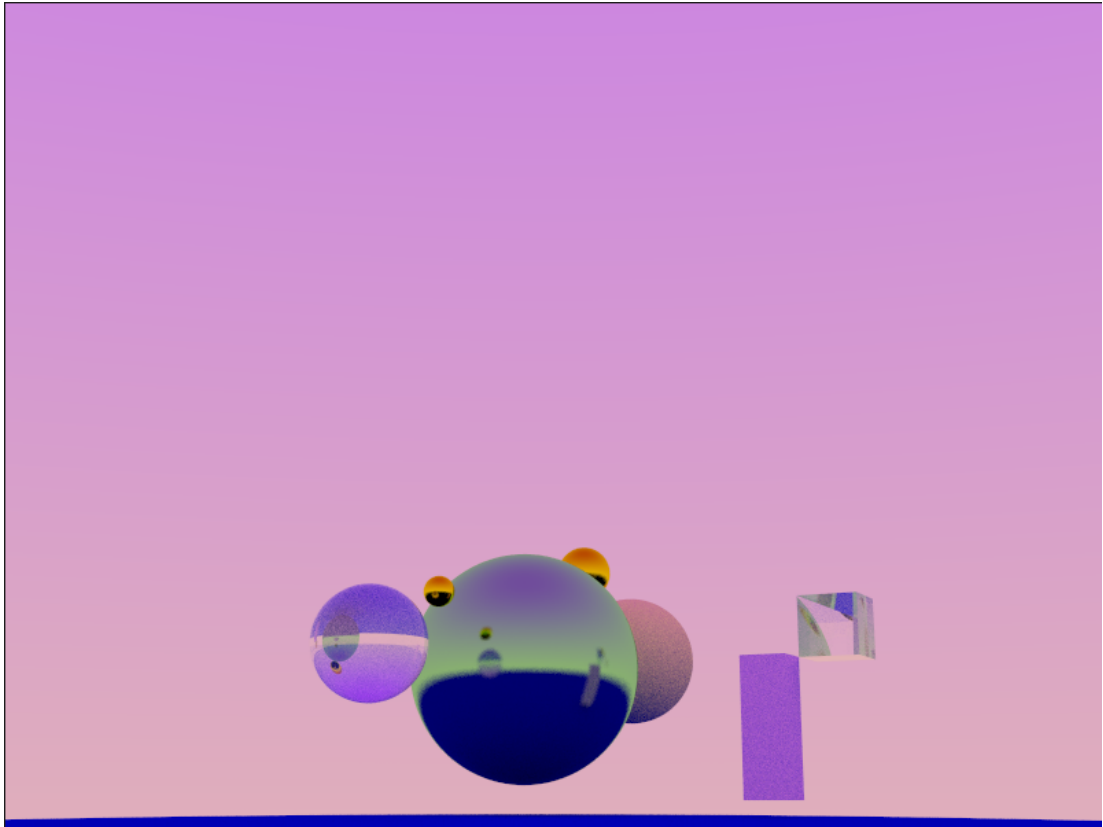


Camera Angle #2 // Always Refracting

▼ **Glass Material (Sometimes Refracting)**

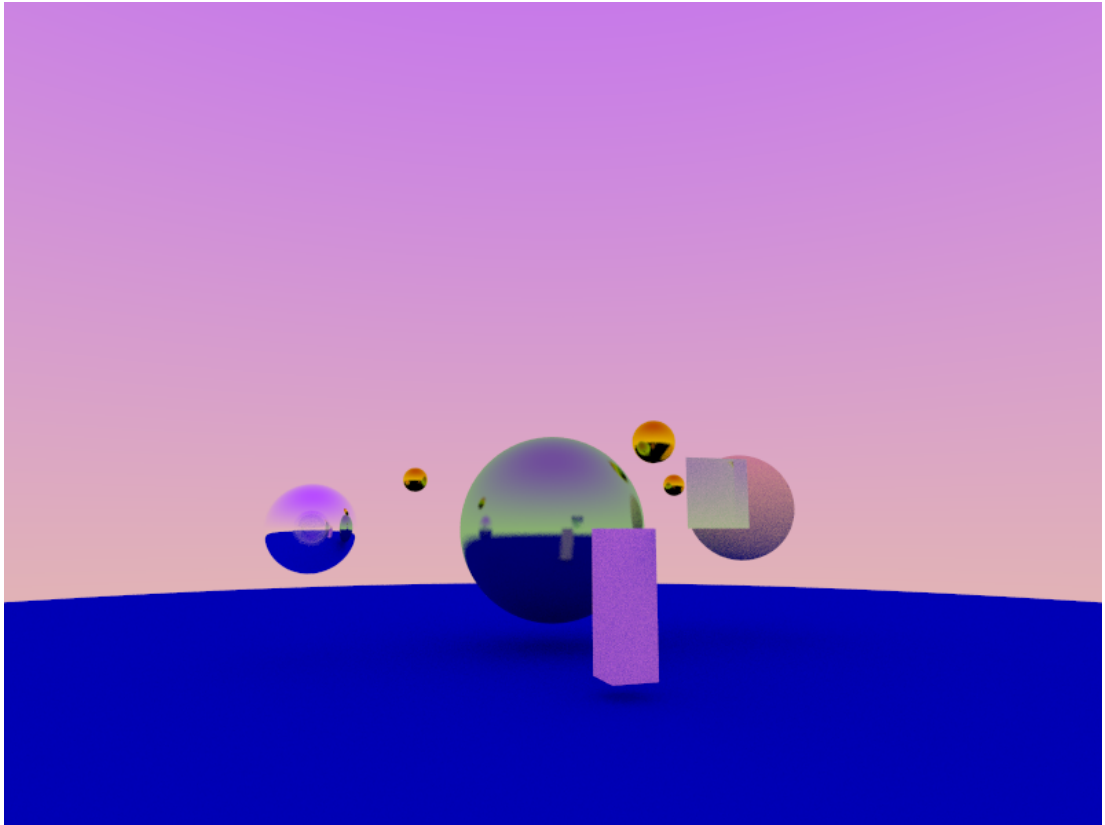


Camera Angle #1 // Sometimes Refracting

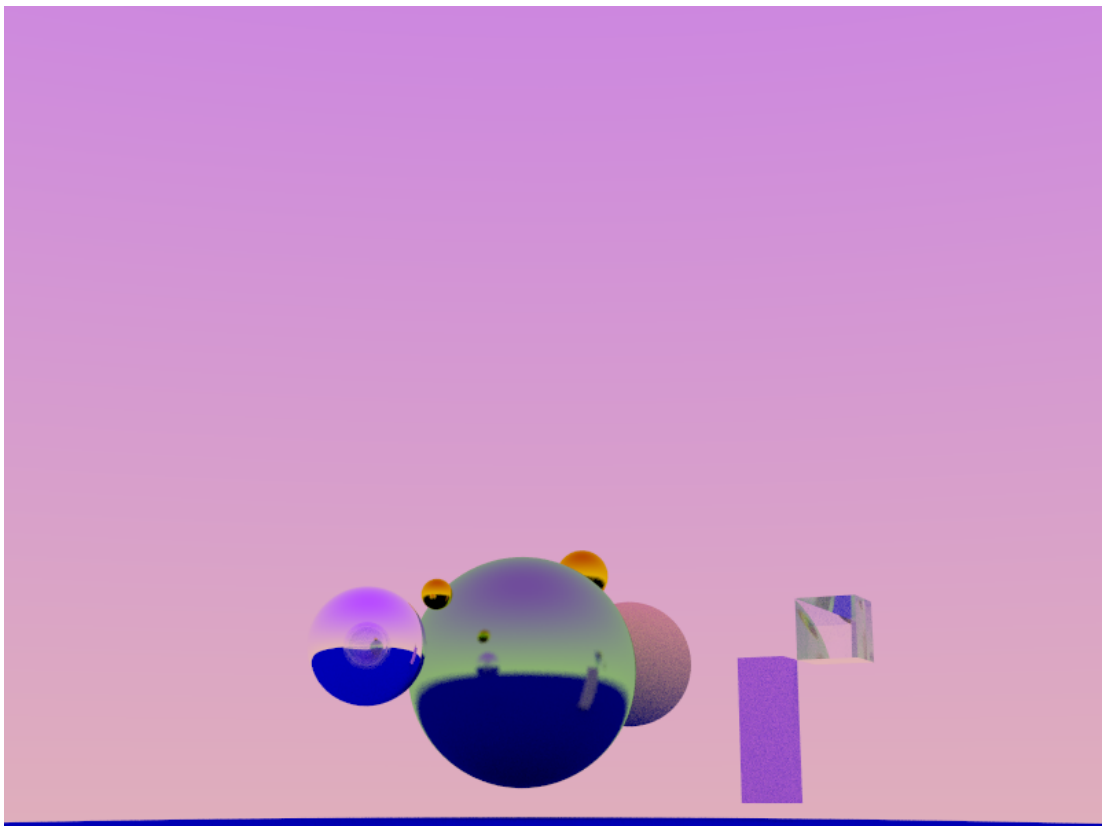


Camera Angle #2 // Sometimes Refracting

▼ **Hollow Glass with Negative Radius**



Camera Angle #1 // Hollow Glass 1



Camera Angle #2 // Hollow Glass 1

▼ Material Properties

- **Ground Material**

- material_ground
 - color(0.0, 0.0, 0.6)

- **Metals (With Fuzziness)**

- metal_gold
 - color(1.0, 0.95, 0.0)
 - fuzziness: 0.15
- metal_green
 - color(0.37, 1.0, 0.37)
 - fuzziness: 0.075

- **Lambertians**

- material_pink
 - color(1.0, 0.6, 1.0)
- material_lambert
 - color(1.0, 1.0, 0.4)

- **Glass**

- material_glass
 - refraction value: 2.5 (smaller values made the glass look more indistinguishable from the background)

▼ Object Properties

- **Ground**

- Material: material_ground
- Coordinates: (0.0, -100.5, -1.0)
- Radius: 100

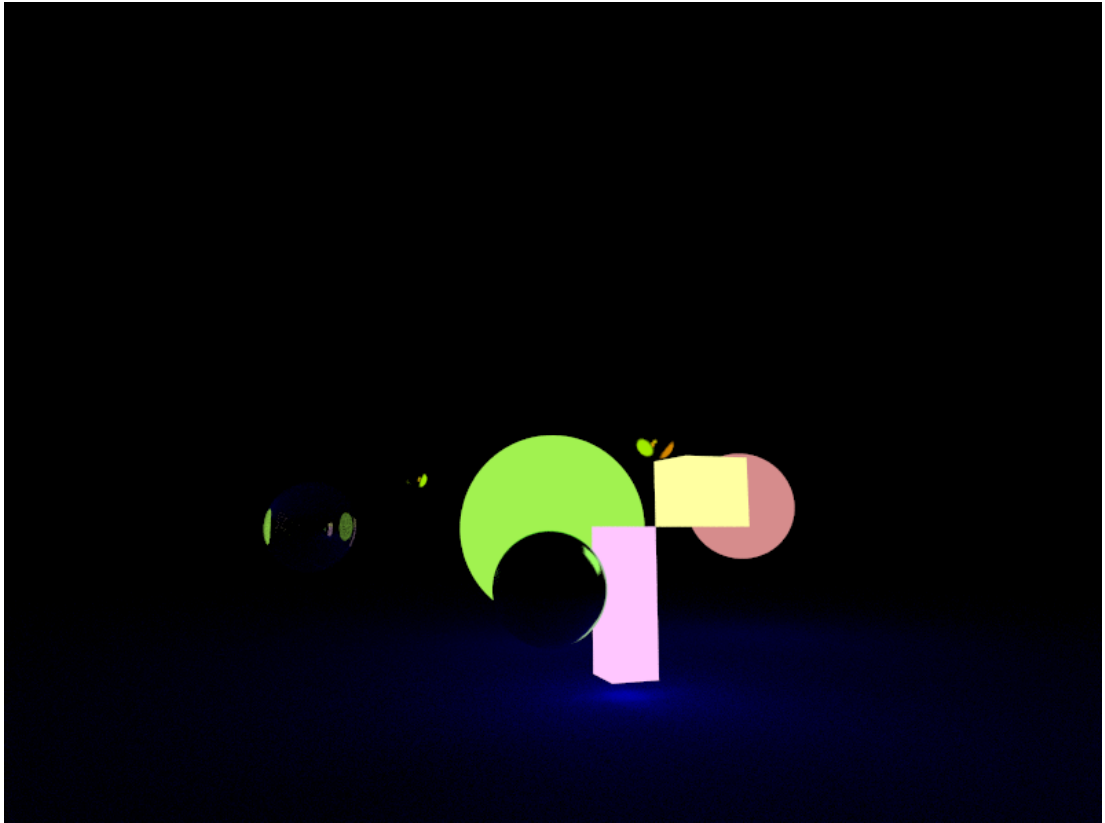
- **Sphere 1**

- Material: material_glass
- Coordinates: (-1.0, 0.0, -1.0)
- Radius: 0.175 (for hollow glass -0.175)
- Sphere 2
 - Material: metal_gold
 - Coordinates: (-0.6, 0.2, -1.0)
 - Radius: 0.05
- Sphere 3
 - Material: metal_green
 - Coordinates: (0.0, 0.0, -1.0)
 - Radius: 0.4
- Sphere 4
 - Material: metal_gold
 - Coordinates: (0.5, 0.4, -1.0)
 - Radius: 0.1
- Sphere 5
 - Material: material_lambert
 - Coordinates: (0.95, 0.1, -1.0)
 - Radius: 0.25
- Cube
 - Material: material_glass
- Rectangle Prism
 - Material: material_aqua

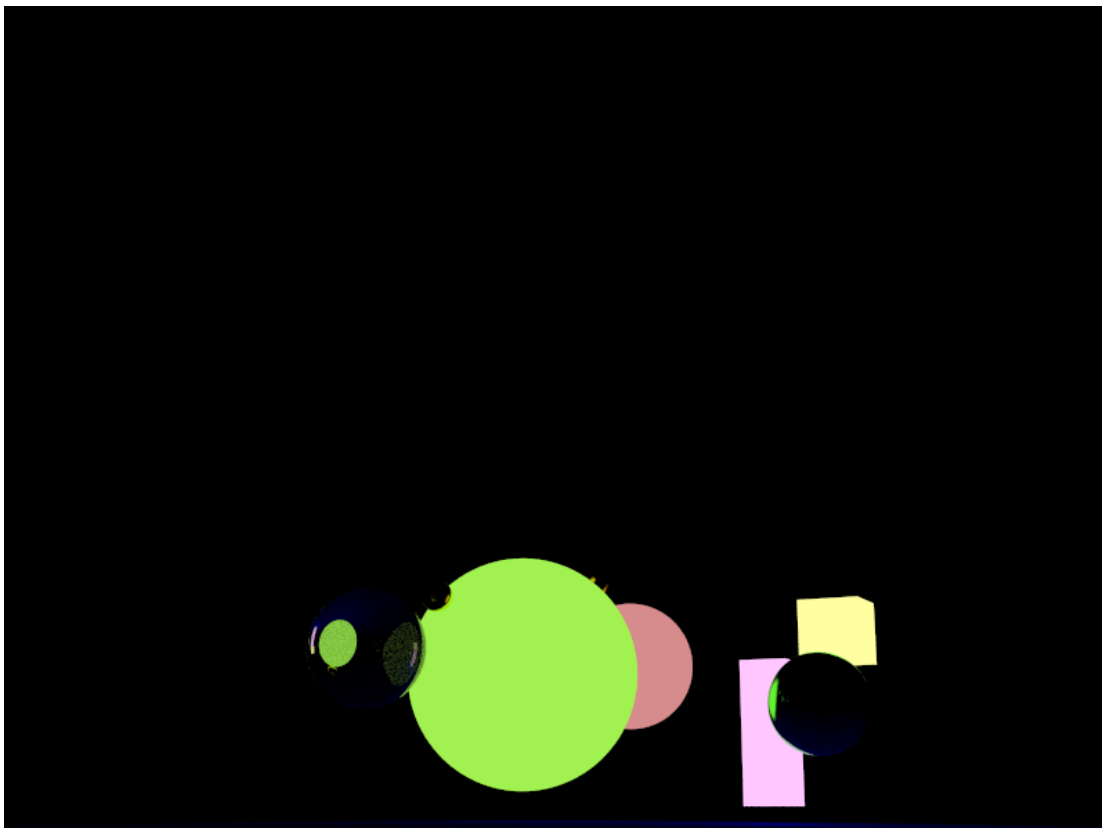
▼ Task 6 [Required]: Lights (15Pt)

▼ Blackout Renders

- Background Color: (0, 0, 0) // Blackout



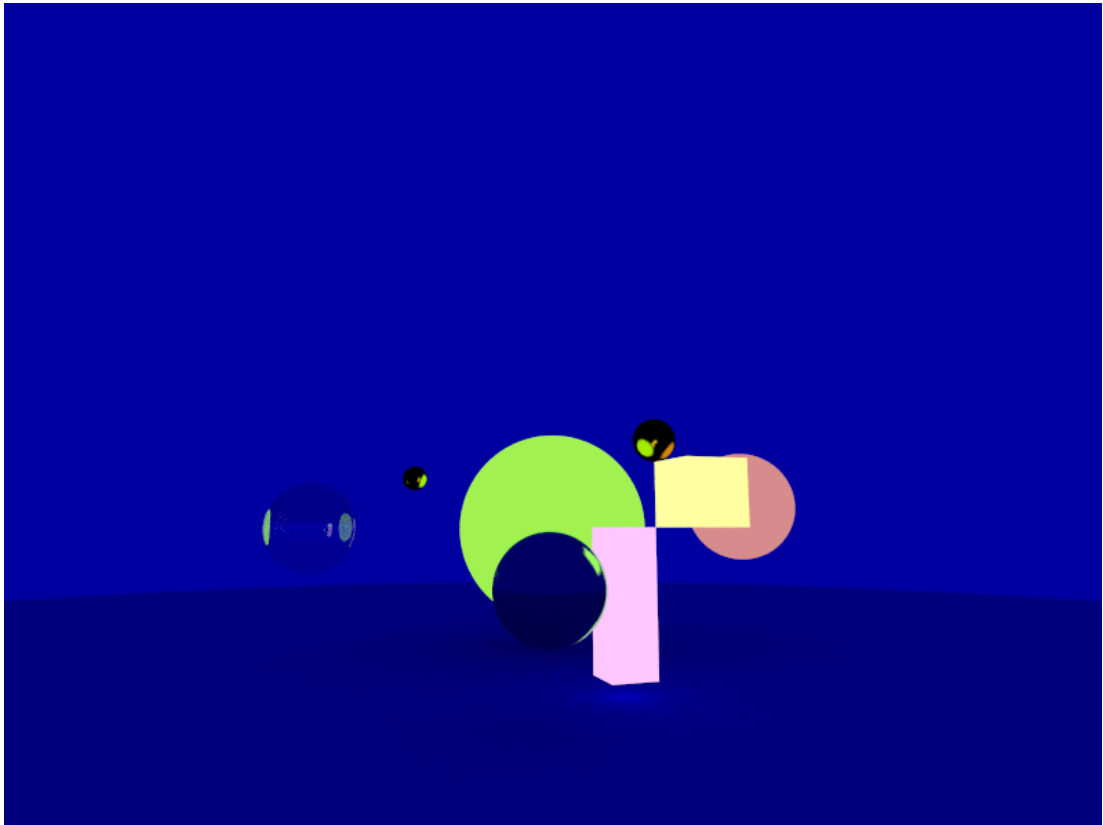
Camera Angle #1 // Blackout



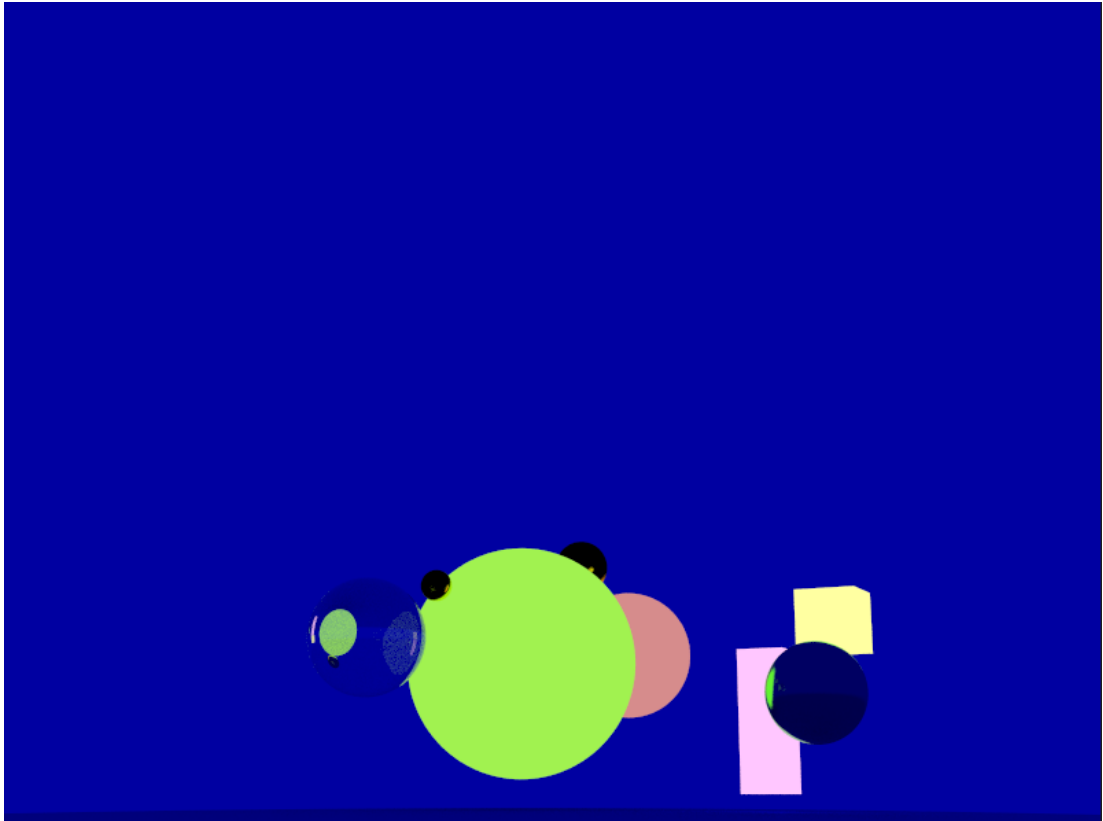
Camera Angle #2 // Blackout

▼ Renders With Background

- Background Color: (0.0, 0.0, 0.4) // Navy Room



Camera Angle #1 // Navy Room



Camera Angle #2 // Navy Room

▼ Material Properties

- **Ground Material**
 - material_ground
 - color(0.0, 0.0, 0.6)
- **Metals (With Fuzziness)**
 - metal_gold
 - color(1.0, 0.95, 0.0)
 - fuzziness: 0.15
 - metal_green
 - color(0.37, 1.0, 0.37)
 - fuzziness: 0.075
- **Glass**
 - material_glass

- refraction value: 2.5 (smaller values made the glass look more indistinguishable from the background)

- **Diffuse Light**

- light_green
 - color(0.4, 0.9, 0.1)
- light_moon
 - color(1.0, 1.0, 0.4)
- light_orange
 - color(0.7, 0.3, 0.3)
- light_pink
 - color(1.0, 0.6, 1.0)

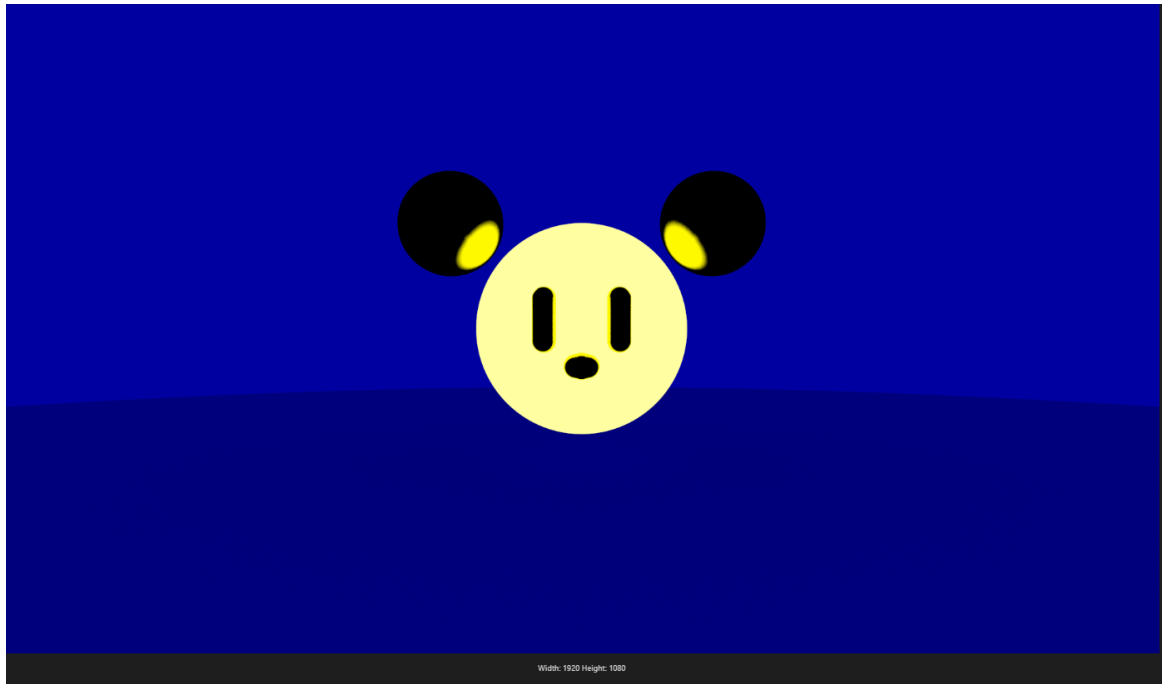
- ▼ **Object Properties**

- Ground
 - Material: material_ground
 - Coordinates: (0.0, -100.5, -1.0)
 - Radius: 100
- Sphere 1
 - Material: material_glass
 - Coordinates: (-1.0, 0.0, -1.0)
 - Radius: 0.175 (for hollow glass -0.175)
- Sphere 2
 - Material: metal_gold
 - Coordinates: (-0.6, 0.2, -1.0)
 - Radius: 0.05
- Sphere 3
 - Material: light_green
 - Coordinates: (0.0, 0.0, -1.0)

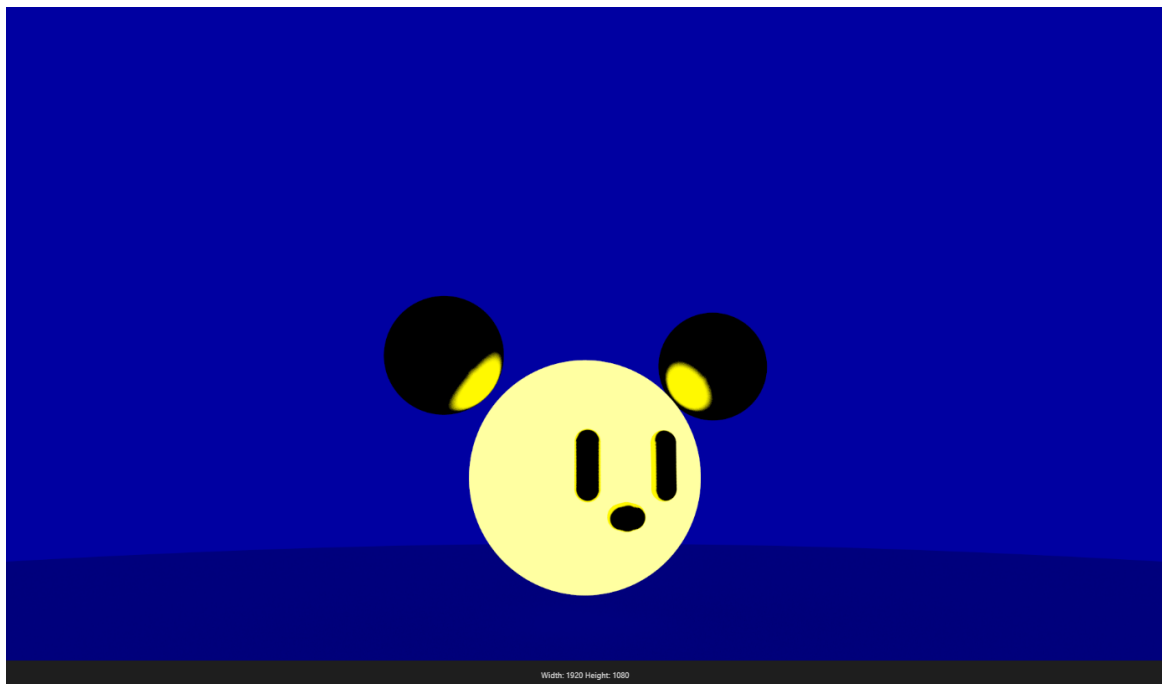
- Radius: 0.4
- Sphere 4
 - Material: metal_gold
 - Coordinates: (0.5, 0.4, -1.0)
 - Radius: 0.1
- Sphere 5
 - Material: light_orange
 - Coordinates: (0.95, 0.1, -1.0)
 - Radius: 0.25
- Sphere 6
 - Material: metal_green
 - Coordinates: ()
 - Radius: 0.15
- Cube
 - Material: light_moon
- Rectangle Prism
 - Material: light_pink

▼ Task 7 [Optional]: Let's get creative! (10Pt)

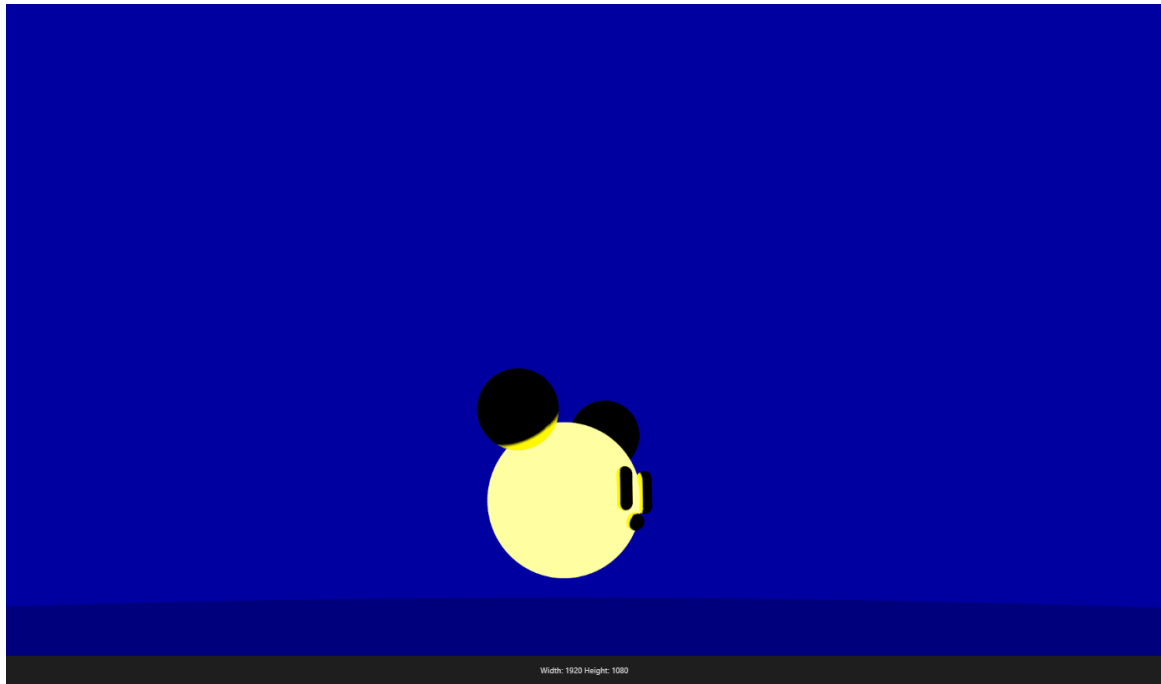
I created a Mickey Mouse head just by using 38 spheres. I used 39 spheres in total to create this render.



Camera #0 // Front



Camera #1 // Angle #1



Camera #2 // Angle #2

▼ Task 8 [Required]: Questions (10Pt)

▼ **Question #1:** The approximate time complexity of ray tracing on models with triangles.

For each pixel of the render scene, ray tracing for models with triangles requires that:

- a ray is created for determining the closest triangles.
- the distance between the triangle and the image plane is calculated for each triangle.

Each pixel is considered and look for the corresponding triangle. In order to determine the closest triangle, all of the triangles should be considered.

Together with having a good data structures and algorithms, the complexity $O(\log n)$ can be obtained, where n is the number of triangles.

▼ **Question #2:** What is the difference between preprocessing and computing the image? Why?

Image preprocessing refers to the process that prepares an image for rendering. Adjusting the brightness of the image in order to obtain accurate

colors or general normalizations for pixels are methods that are used for preprocessing an image.

Computing an image refers to generating colors for each pixel based on the rendering method. These rendering methods can be:

1. Rasterization (including scanline rendering)
2. Ray Casting
3. Ray Tracing

▼ **Question #3:** What are the critical parameters for the ray tracer algorithm's performance?

- **"Sample Per Pixel"** is one of the parameters that has major impact on the performance of the algorithm. As the sample size increases, the rendering quality is also increases but this situation causes rendering time to get very long.
- ***Different algorithms used for implementing different features.*** When implementing features like anti-aliasing or blurring, the efficiency of the algorithms used for these features plays an important role for evaluating the performance of the overall ray tracer program.

▼ **Question #4:** Imagine you are a systems engineer at Pixar. There is a new super-resolution in 6000×6000 pixels and you have to estimate the maximum rendering time per frame. Assume that the scenes are static, so you are not going to spend any CPU on animation, scene hierarchy, etc. The average scene has 500 objects with a total of 5.000.000 triangles to check intersection with.

- Triangle Ray Tracer
 - Time Complexity: $O(\log n)$
 - 6000 * 6000 pixels
 - $5 * 10^6$ triangles
 - Maximum Rendering Time:
 - $6000 * 6000 * \log(5 * 10^6)$

