Ray Tracing and Texturing (1)

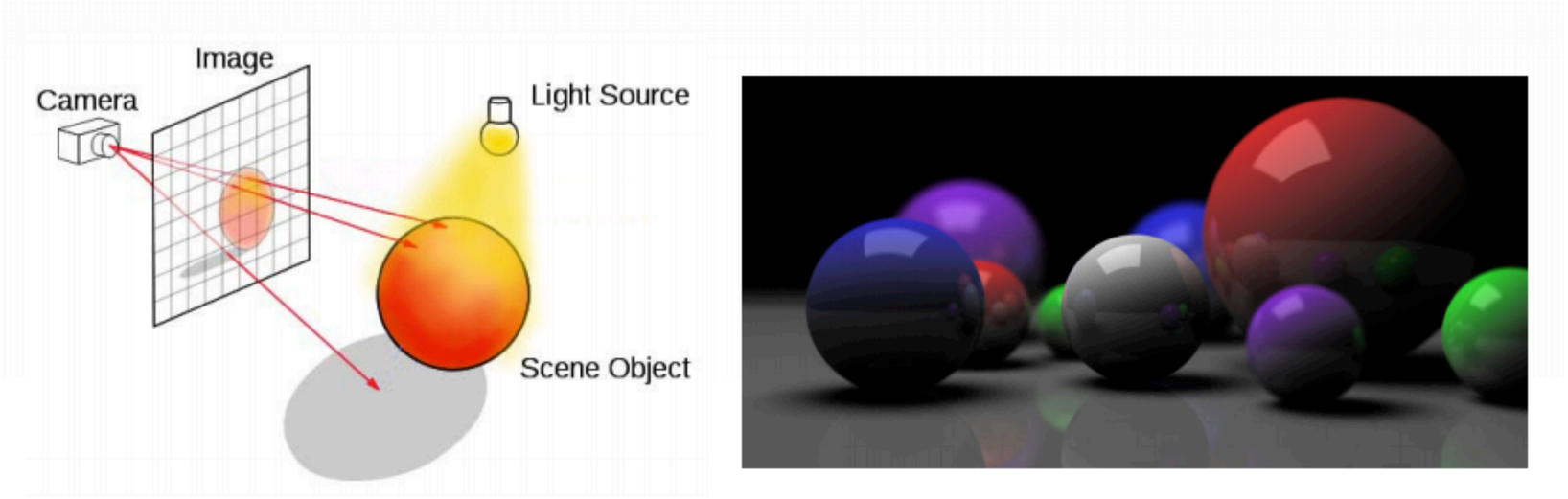
Ray Tracing & Direct Lighting

Yuehao Wang; Apr 9, 2021

Review Ray Tracing

Ray Tracing

- Simulation of the realistic imaging process.
- Recall the imaging process for pin-hole cameras.

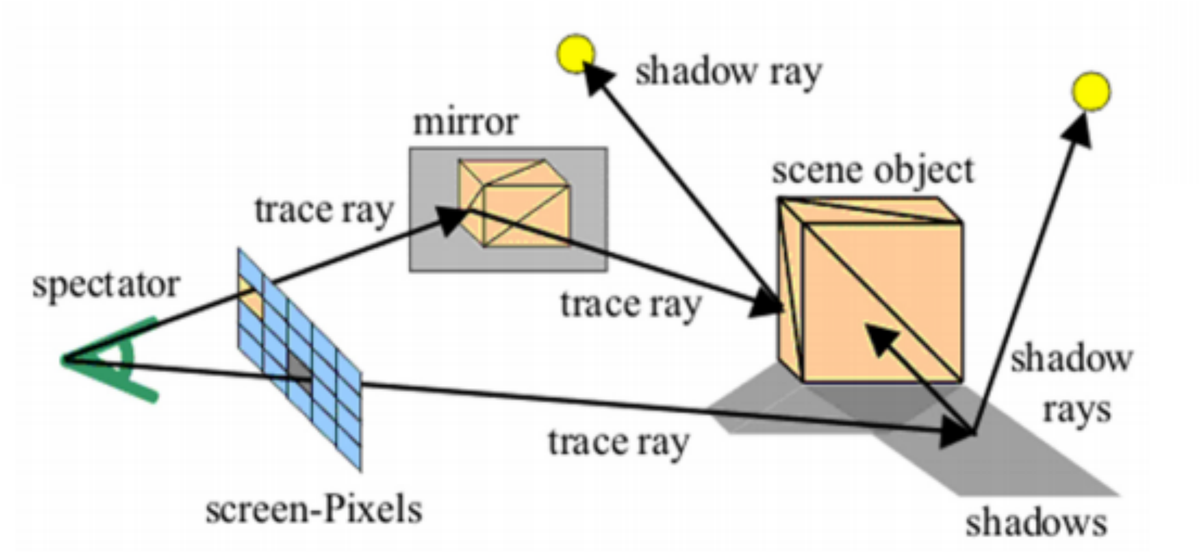


Ray Tracing

- Differing from projection-based rendering, e.g. OpenGL, we cast rays into the scene to evaluate the radiance brought back by ray tracing.
- Ray tracing routine
  - In general, starts from the camera’s pin-hole and directs to a point on the imaging plane.
  - When the ray intersects with some objects, perform reflections on the interaction point.
  - Repeat until the ray hits any light sources.
  - Traceback and compute the radiance brought back.

Ray Tracing

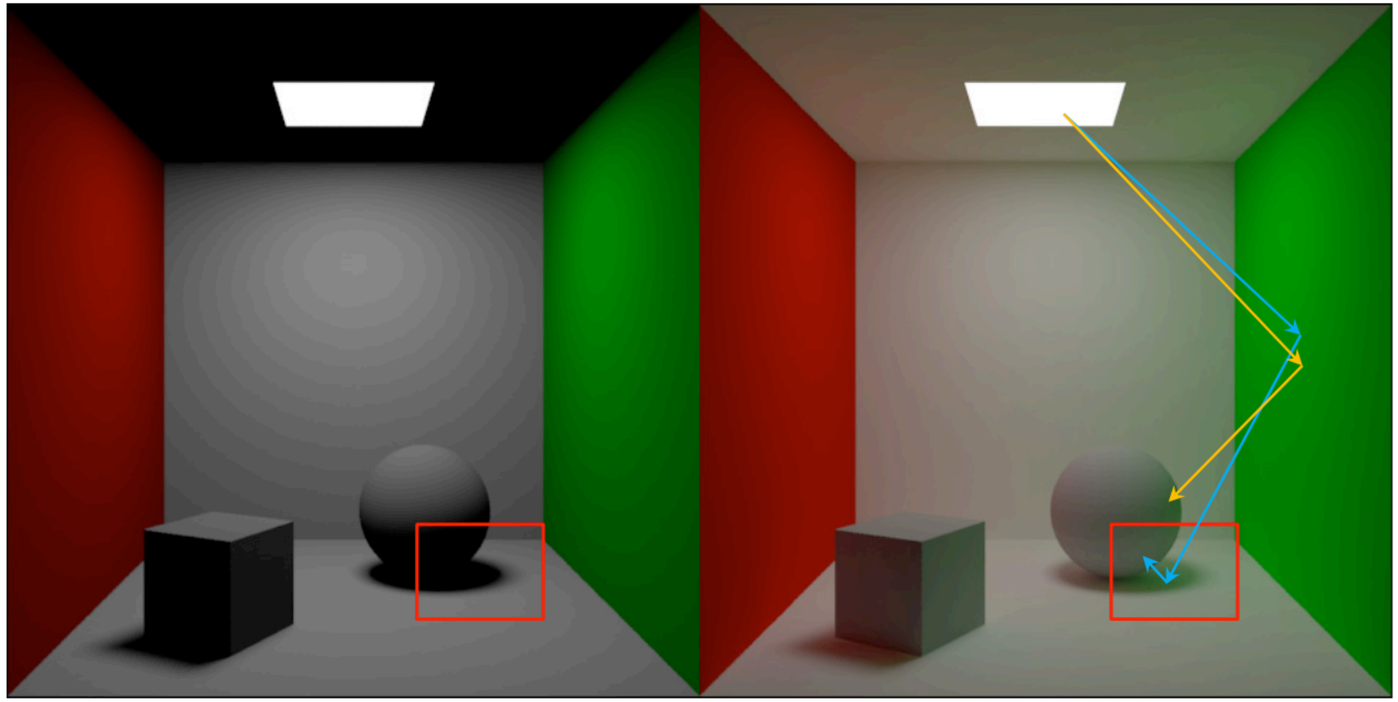
- For simplification, we consider merely two kinds of reflection:
  - Specular reflection (e.g. mirror, icy surface)
  - Diffusion reflection (e.g. rough walls)



Direct Lighting

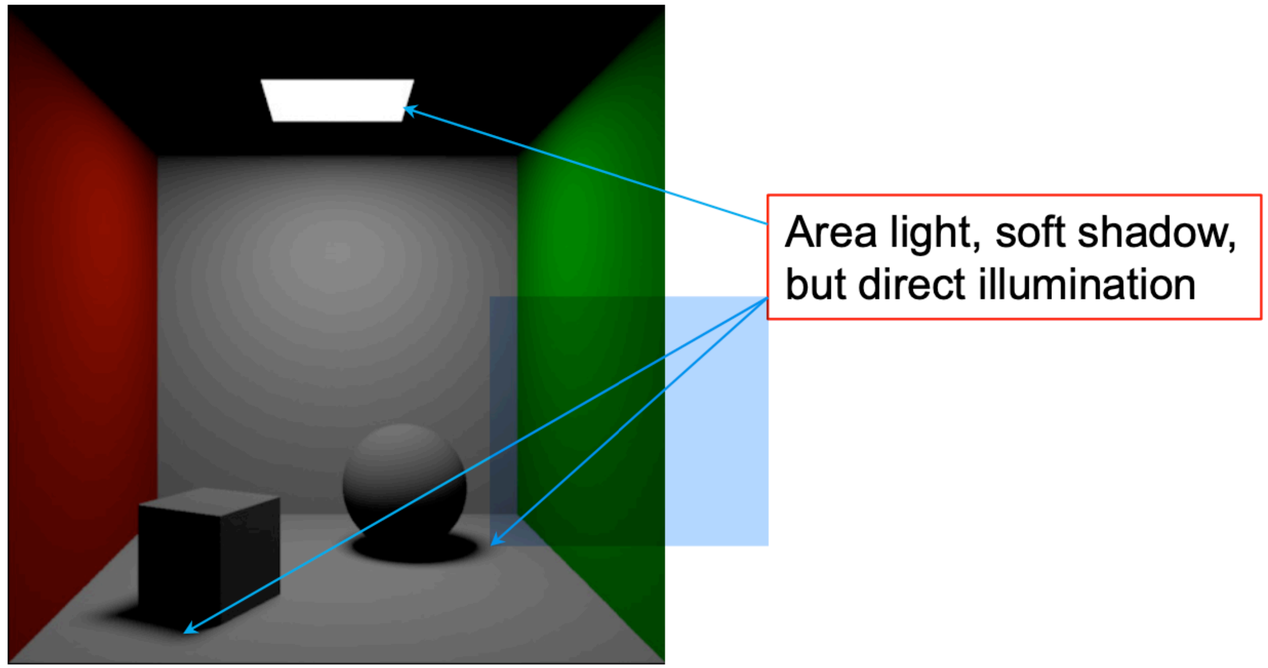
- This routine is somewhat problematic and inefficient since the ray could be reflected thousands of times until meeting any light source.

- A practical method is to decompose the lighting into **direct** and **indirect**.



Direct Lighting

- Direct lighting: once the shot rays hit any object, we directly reflect rays to the light sources.
- Shadow: But those rays may hit other objects when they are reflected to the light sources (sheltering/occlusion) => Results in shadow.
- If the light sources are area lights. Some shadow has chances to be lighted => Results in soft shadow.

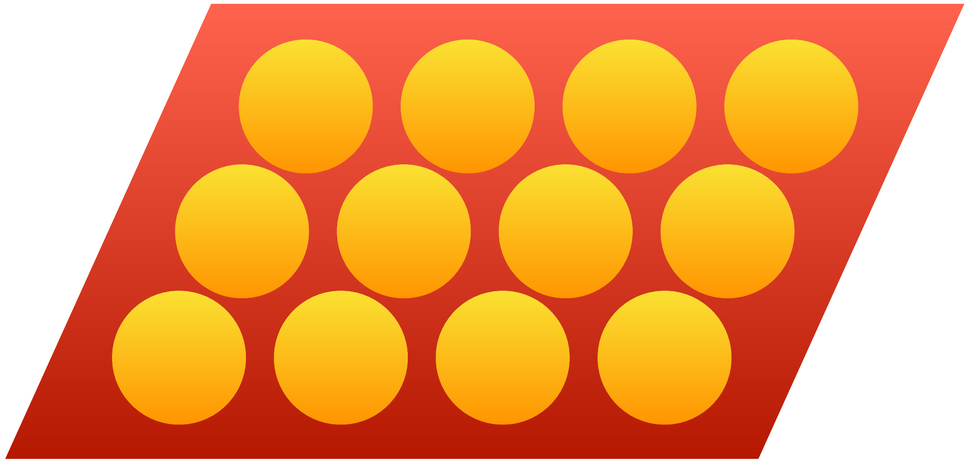


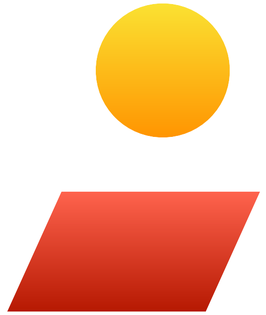
Direct Lighting

- Rendering equation: $L_o(p, \omega_o) = \int_S f(p, \omega_i, \omega_o) L_i \cos\theta_i d\omega_i$
- For discrete case, integral => summation.
- For Phong shading model => BRDF (f) is modeled as diffusion + specular.
- Discrete + Phong shading (diffusion):
  - $L_o(p, \omega_o) = \sum_{L_i} \text{diff} \cdot L_i \cos\theta_i$
    - Since direct lighting does not handle specular surface => we can approximate it by Phong model.

Area Light

- Recall the simplified rendering equation for direct Phong lighting: L_i represents all incoming lights.
- In direct lighting, L_i's are light sources. Basically they are points in the scene.
- Area light can be sampled as a collection of point lights:
  - Uniform area light: every point in the area light has the same power.
  - Total power: $\mathscr{L}$, samples: N, power of each point lights: $\mathscr{L}/N$.





A point light sample

Area light with 12 samples

Radiance

- Notice that what we consider now is radiance for each pixel on the imaging film. Not RGB color yet.
- Color represents waves with different wavelengths, similar to radiance. In computer area, lights are decomposed to 3 components: Red, Green, Blue.
  - Intuition: red, green, blue are bases to span a finite color space.
- For simplification in Phong lighting model, we can decompose radiance to RGB components and compute the radiance for the three channels, respectively.
- Finally, covert radiance to color via tone mapping (Gamma correction).

Homework 3

Task (tentative)

- Implement a pin-hole camera, which is able to generate rays.
- Implement ray-object intersection algorithms.
- Implement a Phong lighting integrator, which is supposed to compute radiance for each pixel.
- Implement texturing. You are required to attach a texture on objects.
- [optional] Implement mipmap/ripmap for texturing.
- [optional] Enable anti-aliasing for ray-based rendering.
- [optional] Implement environment map (cube map) for global lighting.

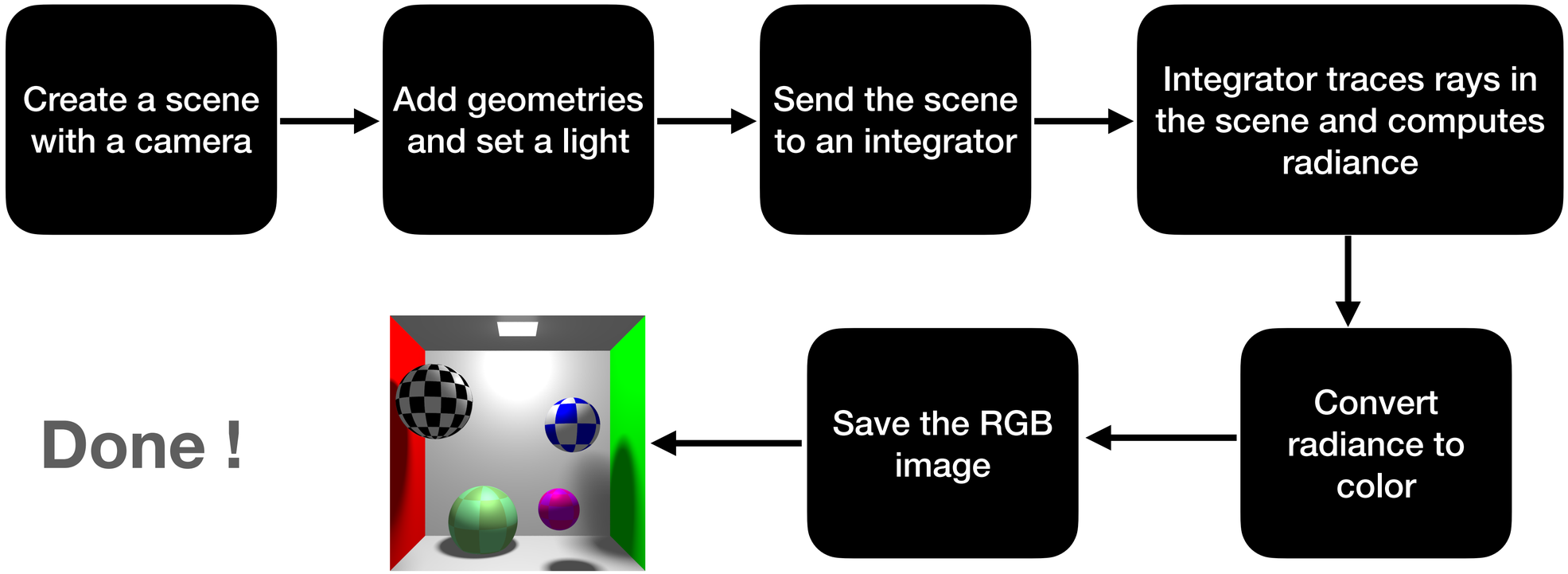
Notes

- Please be noticed that you are NOT allowed to use OpenGL.
- As the computation in this homework is quite heavy, we encourage you to use OpenMP or CUDA for operating-system level acceleration.
  - You may apply for the use of SIST HPC Cluster if necessary.
- A code skeleton will be offered to you. Please understand the functionality of each declared class in the code skeleton before you start.
- To verify your algorithm, you can use different camera/light settings.
- Please start as early as possible!

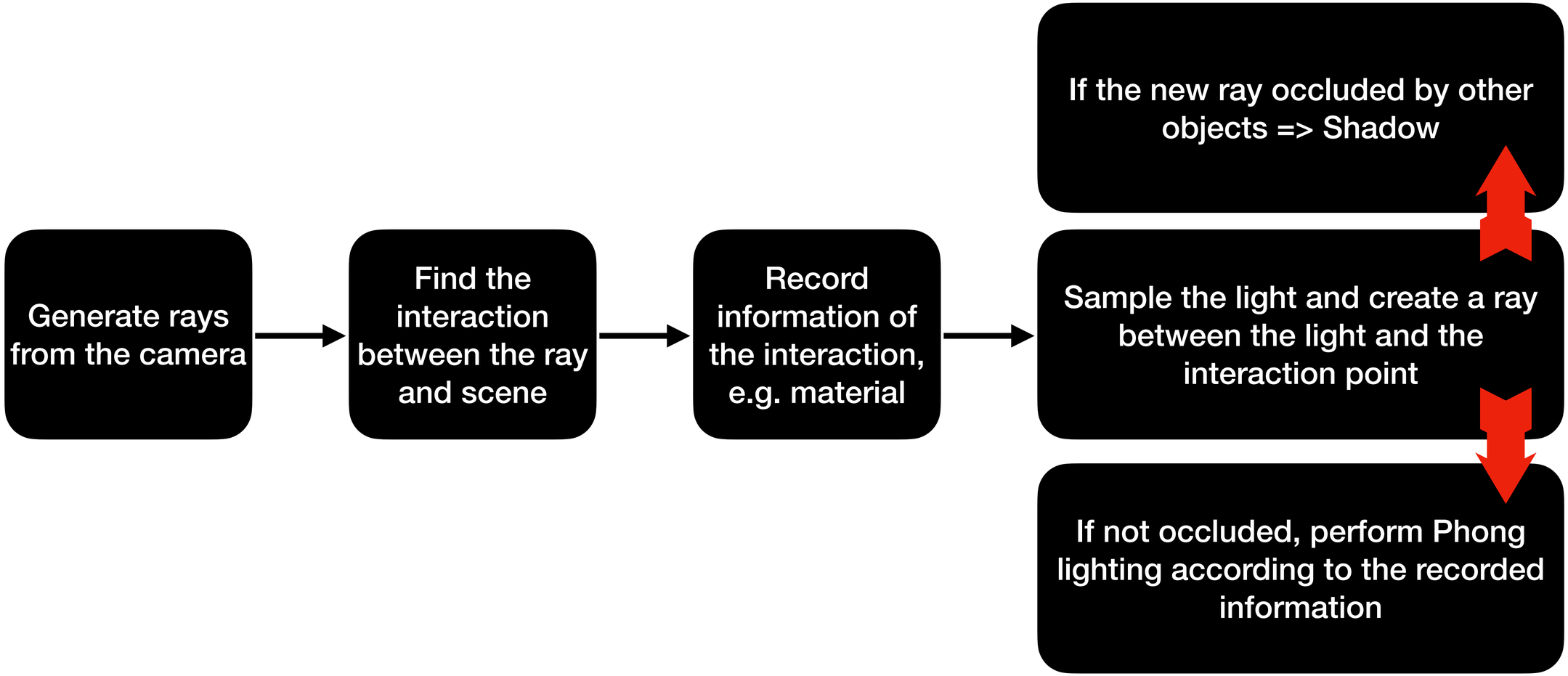
Code Components

- Camera: data structure and methods for manipulating cameras.
- Film: data structure storing each pixel’s radiance.
- Geometry: represents 3D geometries in the scene, e.g., box, sphere.
- Integrator: used to solve the rendering equation.
- Interaction and Ray: basic data structures for ray-object intersection
- Light: represents light sources in the scene.
- Material: stores Phong model and texture information for geometries.
- Scene: contains geometries and lights, with a camera attached.

Working Flow

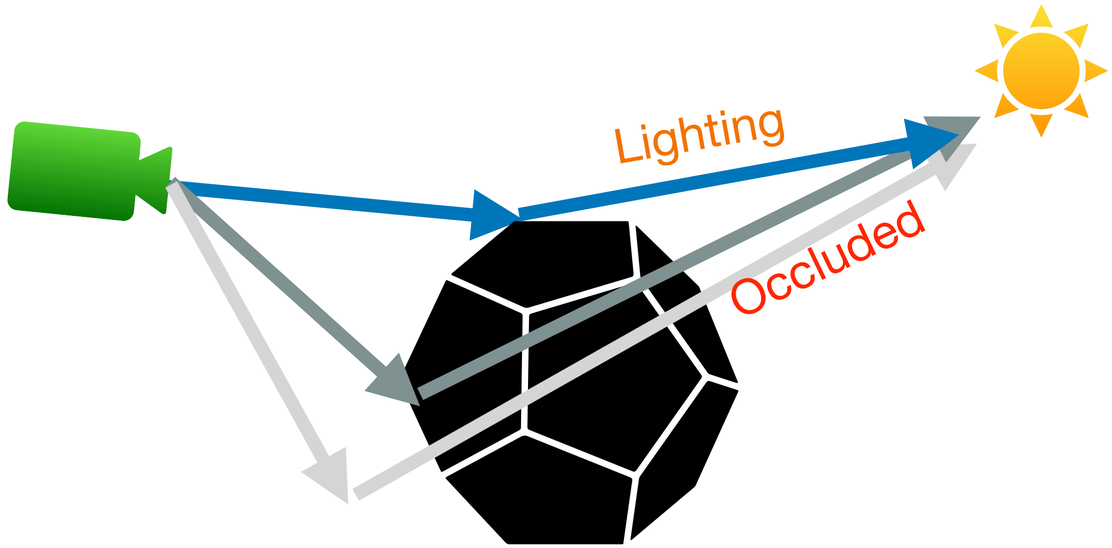


Working Flow in the Integrator

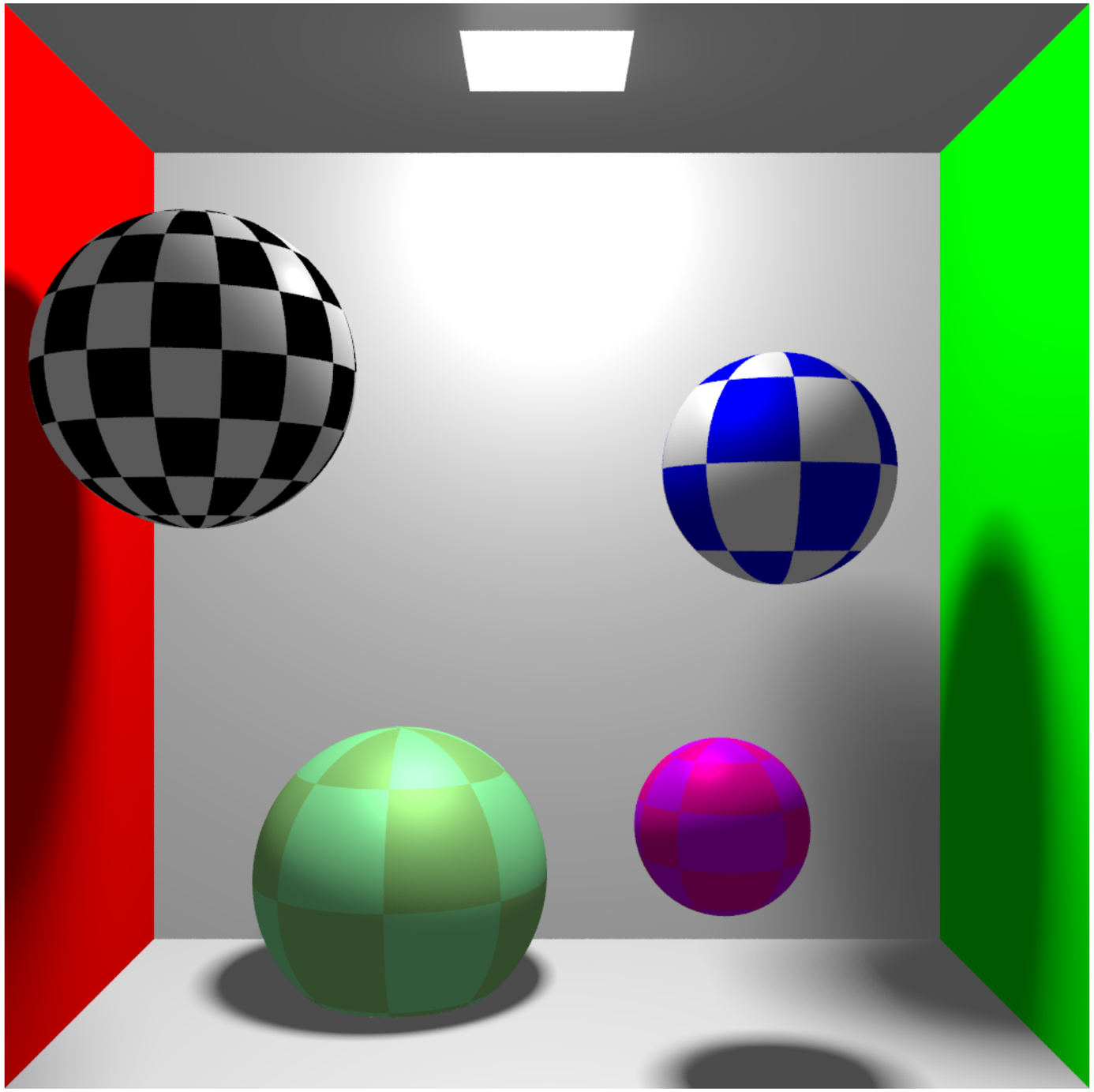


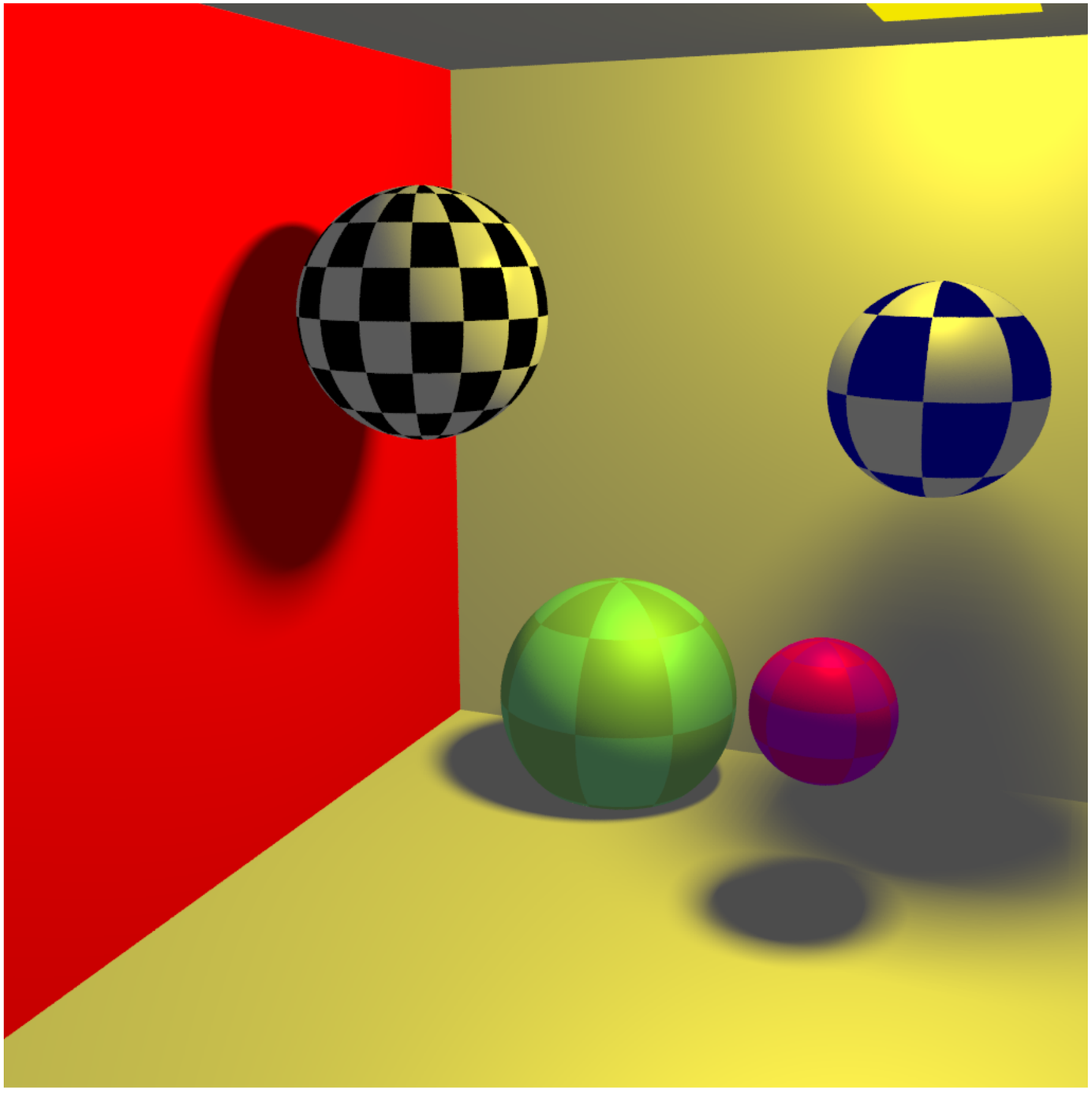
Algorithm for Direct Phong Lighting

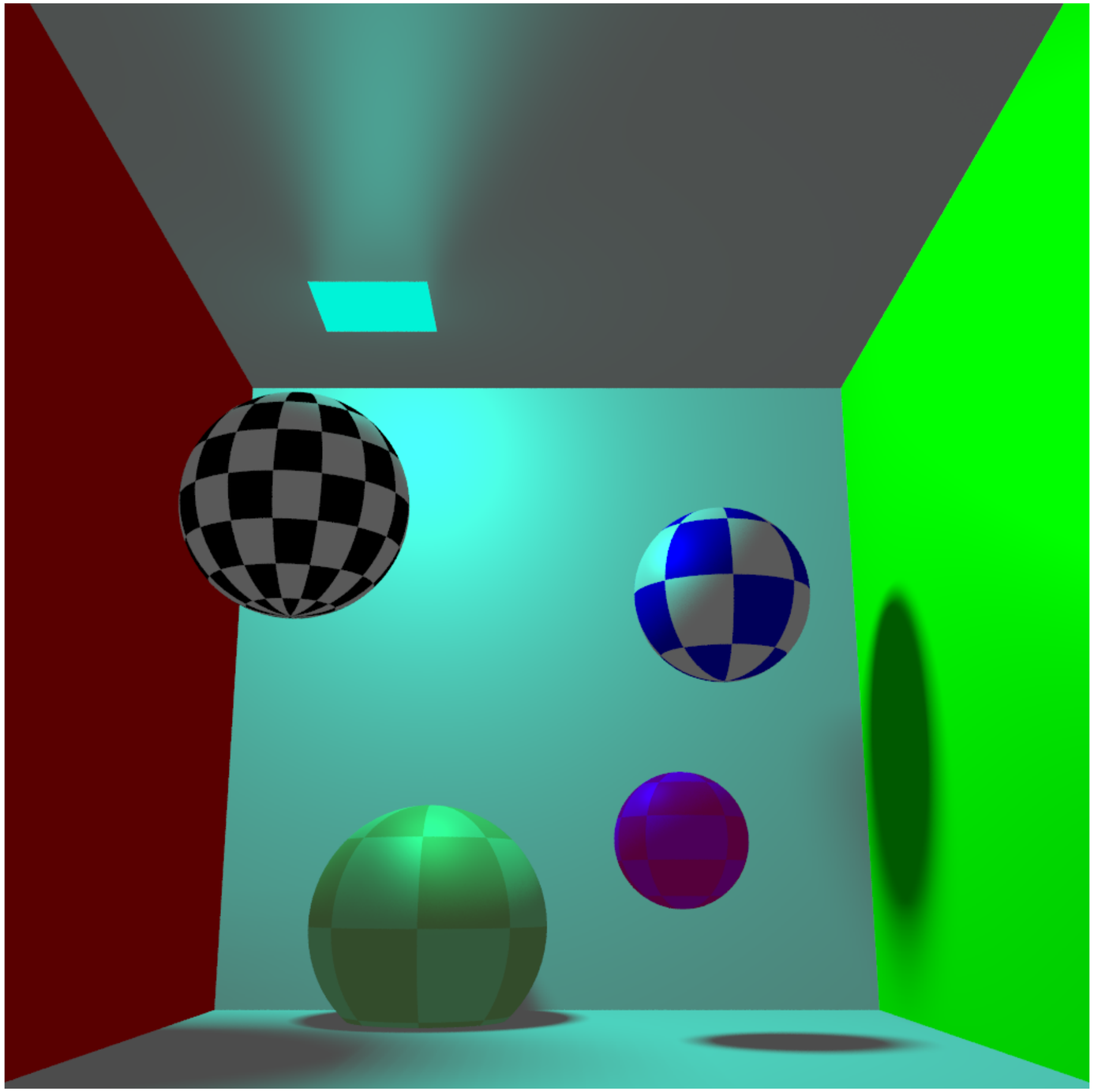
- If intersects(scene, camera ray) at P
  - Create a new ray from P to a light source.
  - If intersects(scene, new ray)
    - Return ambient lighting
  - Return diffusion + specular + ambient lighting



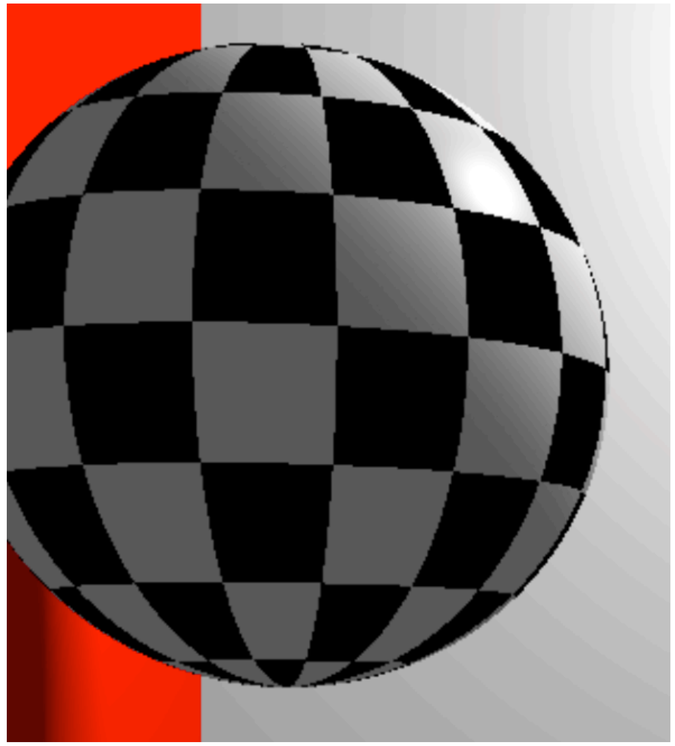
Demonstration



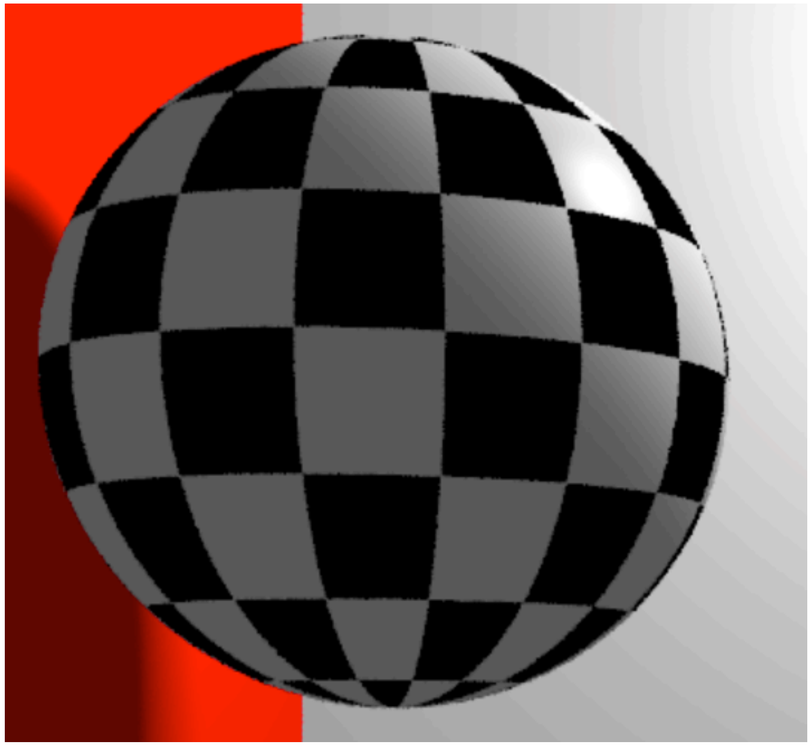




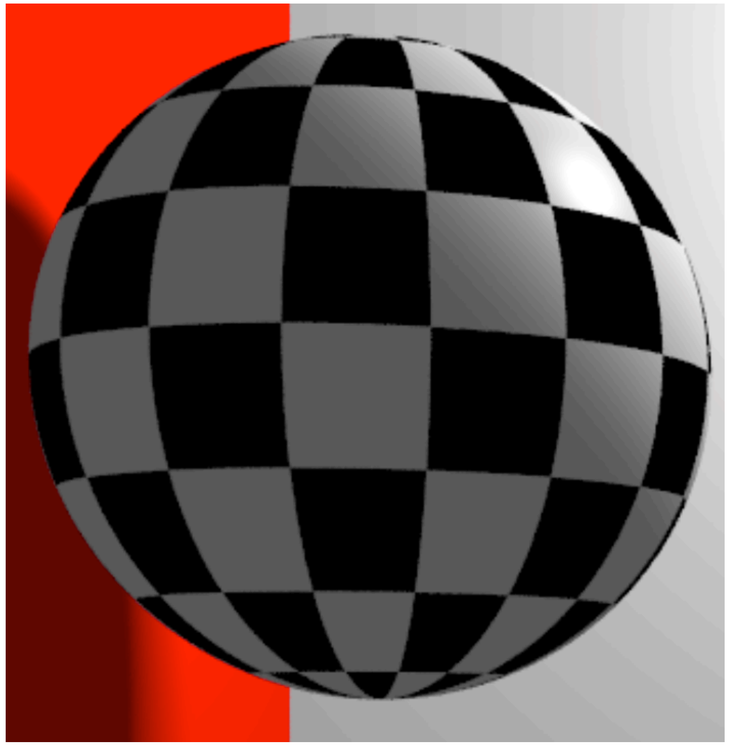
Multi-Sampling Anti-Aliasing



No MSAA



Samples=4



Samples=16

References

- *Physically Based Rendering: From Theory to Implementation*. Matt Pharr, Wenzel Jakob, and Greg Humphreys. http://www.pbr-book.org/.

- *Computer Graphics I, Lecture 9-12*. Prof. Xiaopei Liu.

Q&A