Introduction to Computer Graphics (CS360A)

Instructor: Soumya Dutta

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur (IITK)

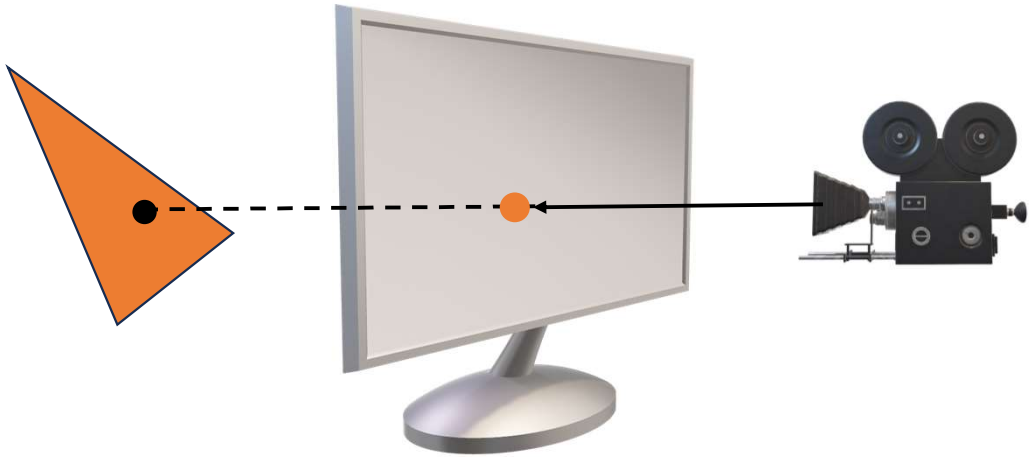
email: soumyad@cse.iitk.ac.in

Acknowledgements

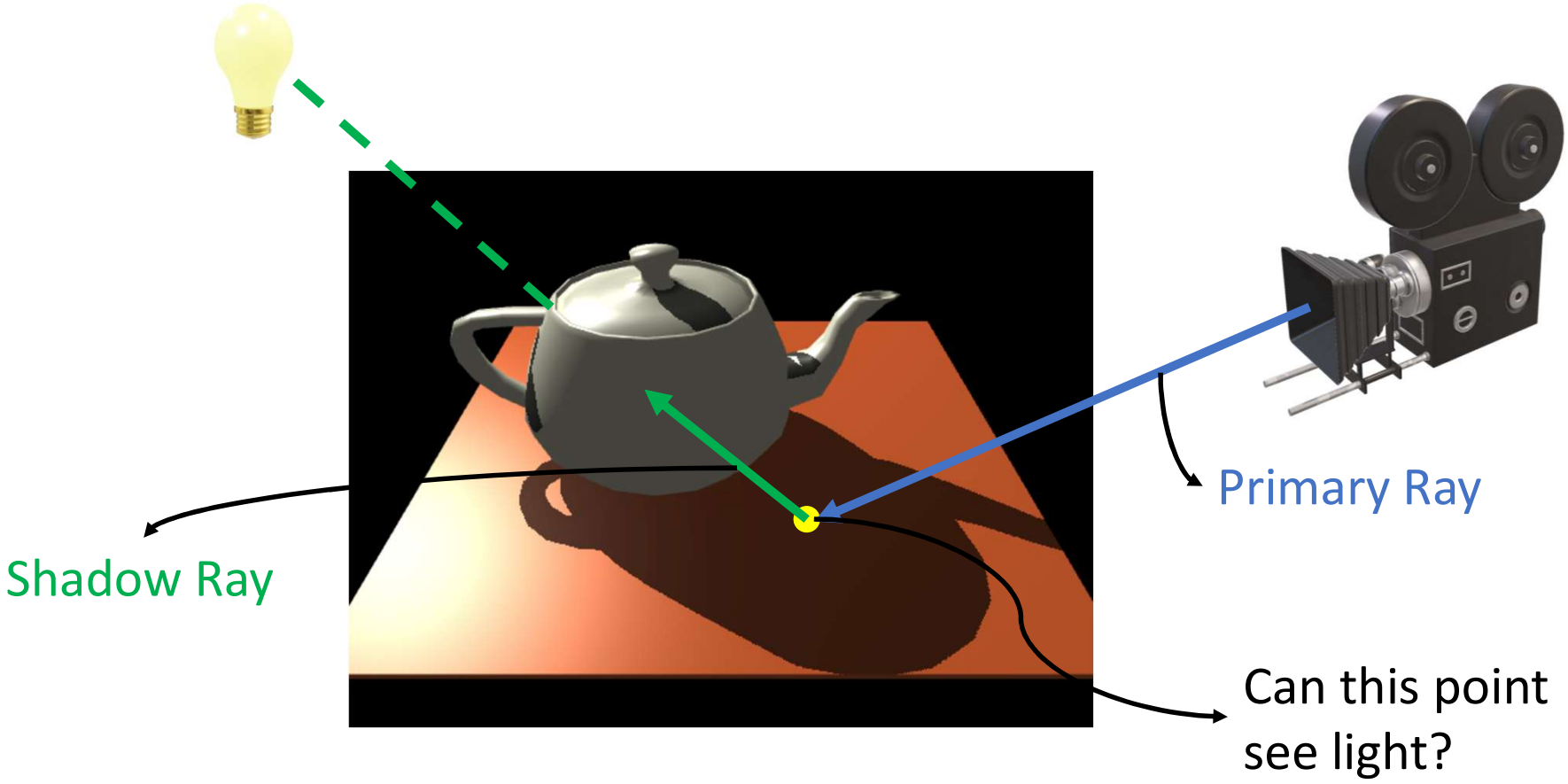
- A subset of the slides that I will present throughout the course are adapted/inspired by excellent courses on Computer Graphics offered by Prof. Han-Wei Shen, Prof. Wojciech Matusik, Prof. Frédo Durand, Prof. Abe Davis, and Prof. Cem Yuksel

Ray Tracing

- Start a ray from camera and find out which scene object the ray hits
- If a hit is found, set the fragment color with the primitive color at that point
- If no hit detected, set the fragment with background color



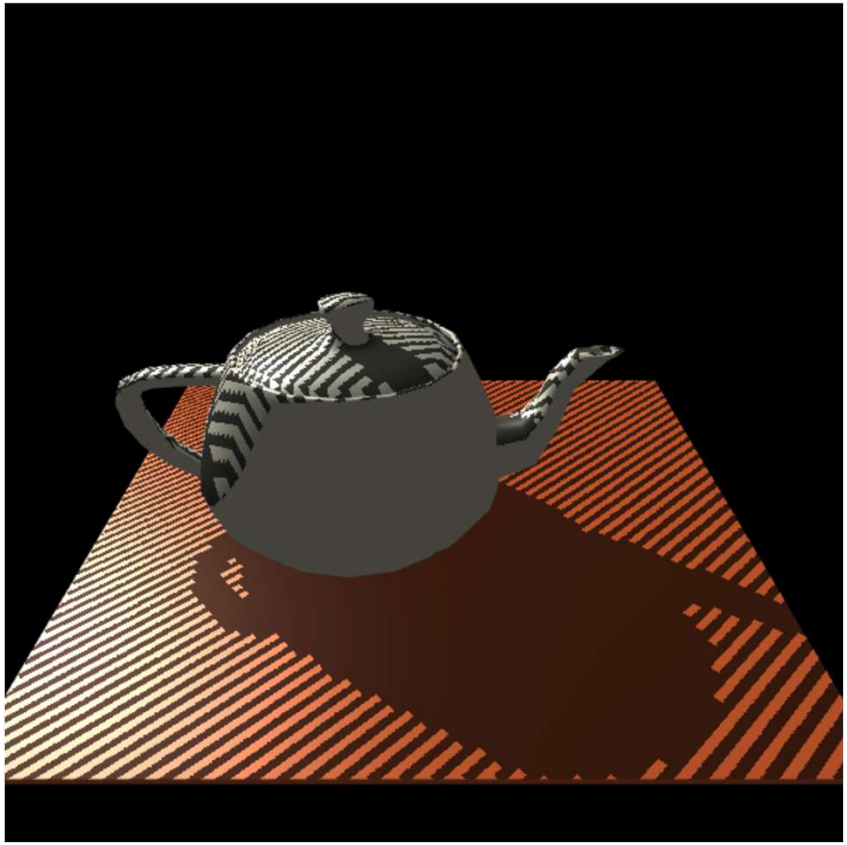
Ray Traced Shadow



Ray Traced Shadow

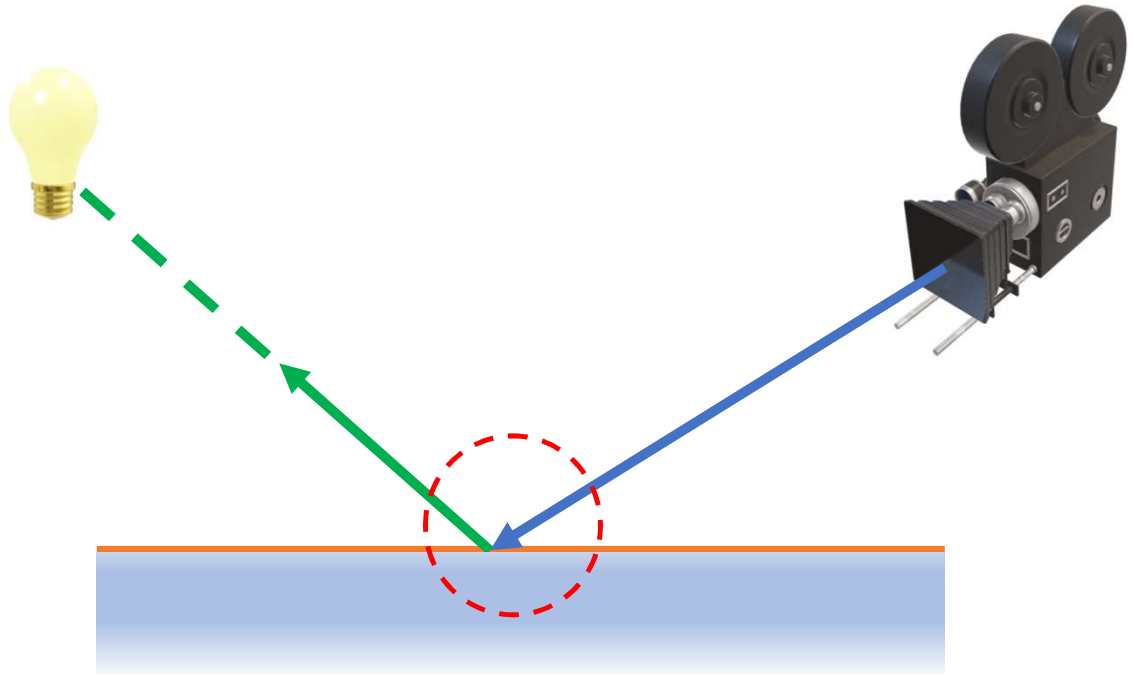
- Trace primary ray from camera/eye to compute the intersection point
- Start a secondary (shadow) ray towards the light
  - Ray origin: Intersection point
  - Ray direction: from intersection point to the light
- Test intersection of shadow ray with all objects in the scene
- If the shadow ray hits any object in the scene, then the point is in shadow
- The process should be repeated for each light source in the scene

Ray Traced Shadow: Issue

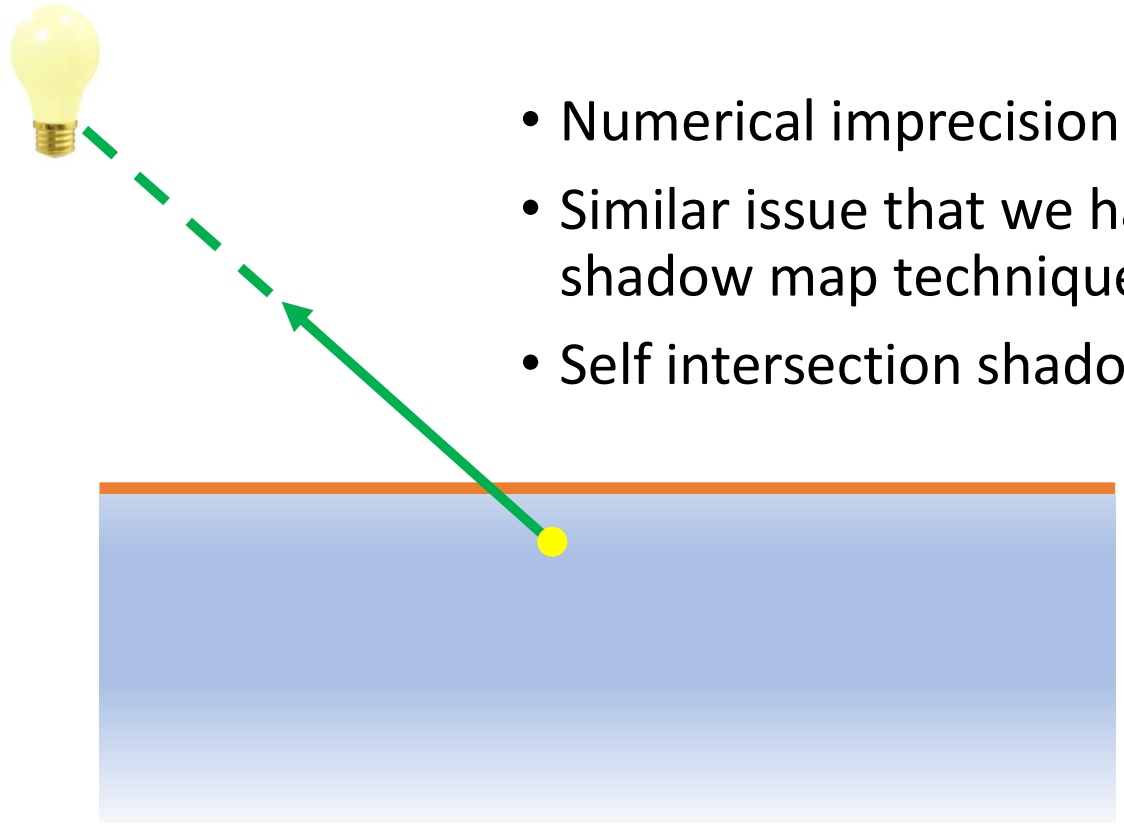


- You may see the that the output shadow has some artifacts as seen in the image left
- What is the problem?

Ray Traced Shadow: Issue

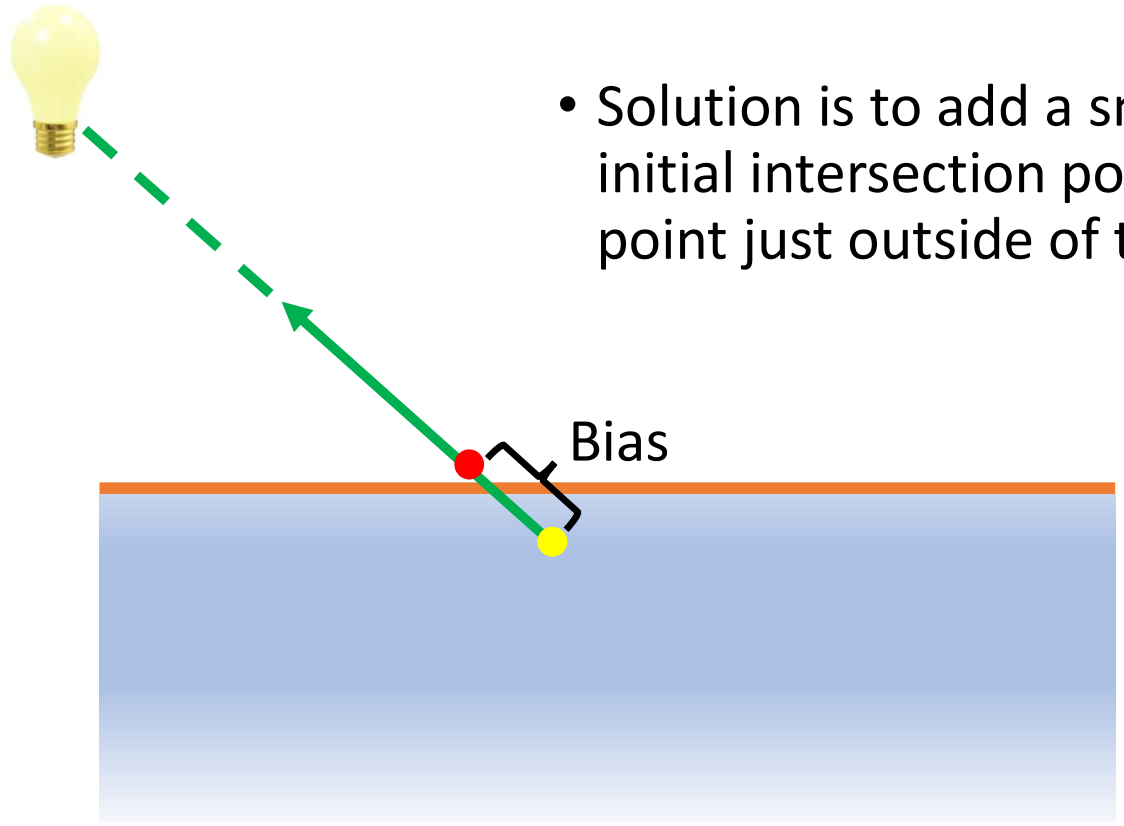


Ray Traced Shadow: Issue



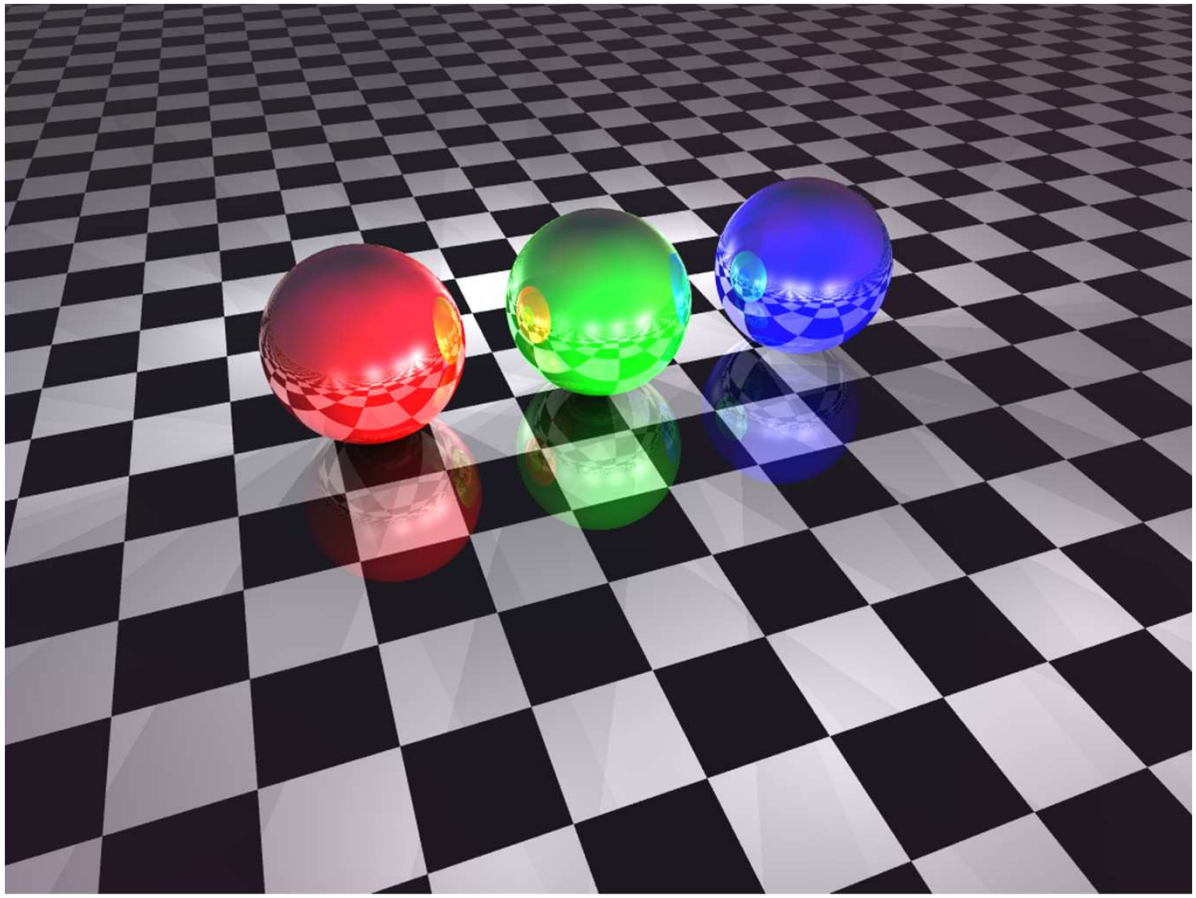
- Numerical imprecision
- Similar issue that we have seen before for shadow map technique
- Self intersection shadow

Ray Traced Shadow: Issue



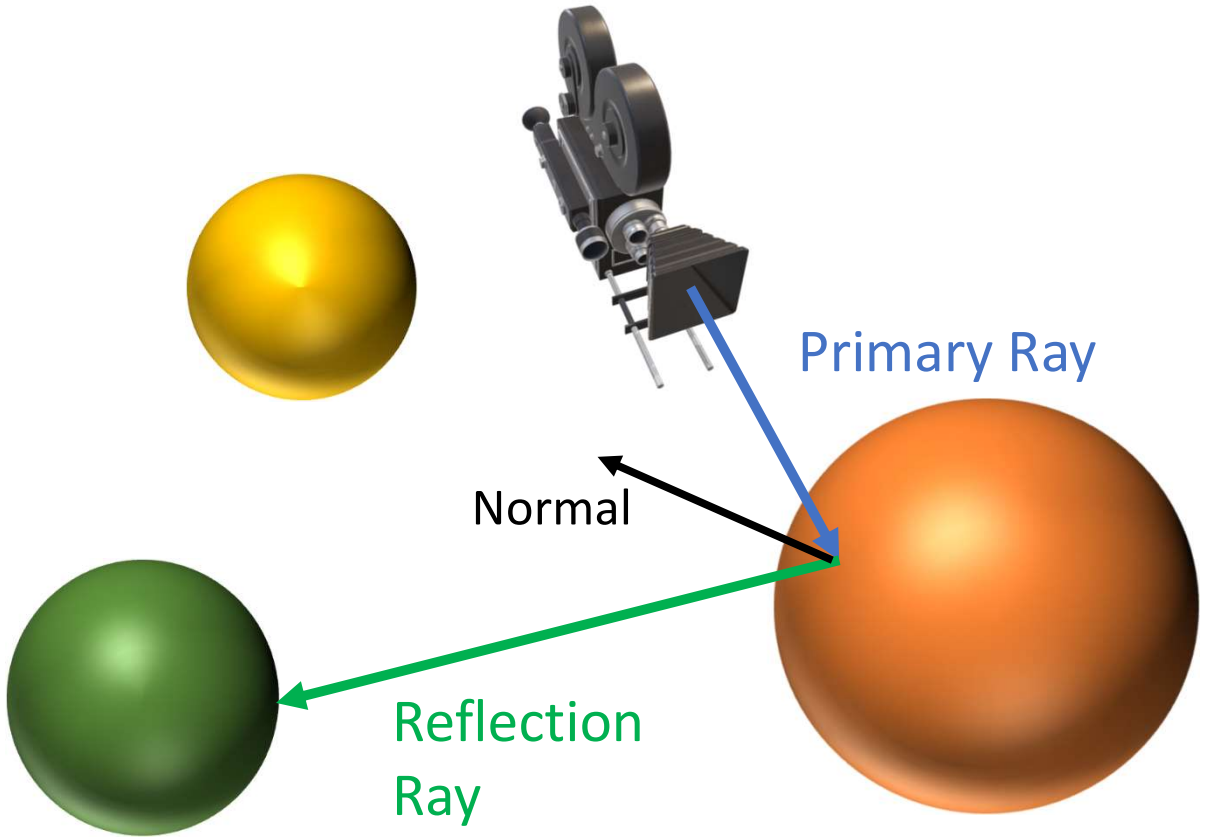
- Solution is to add a small bias with the initial intersection point to push the point just outside of the object

Ray Traced Reflection



"perfect" specular reflection of objects

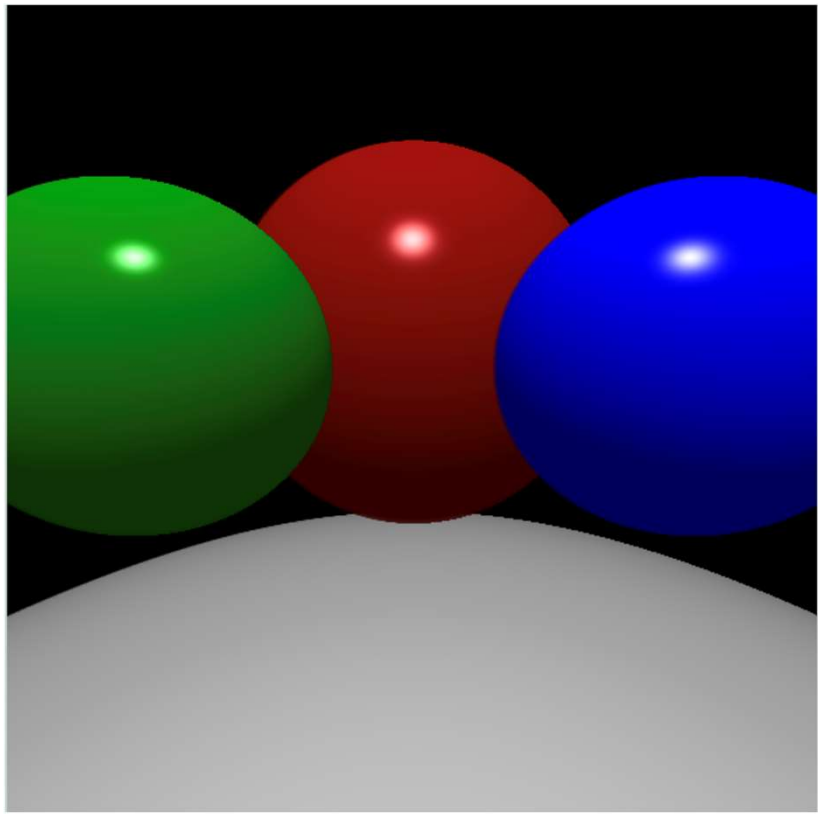
Ray Traced Reflection

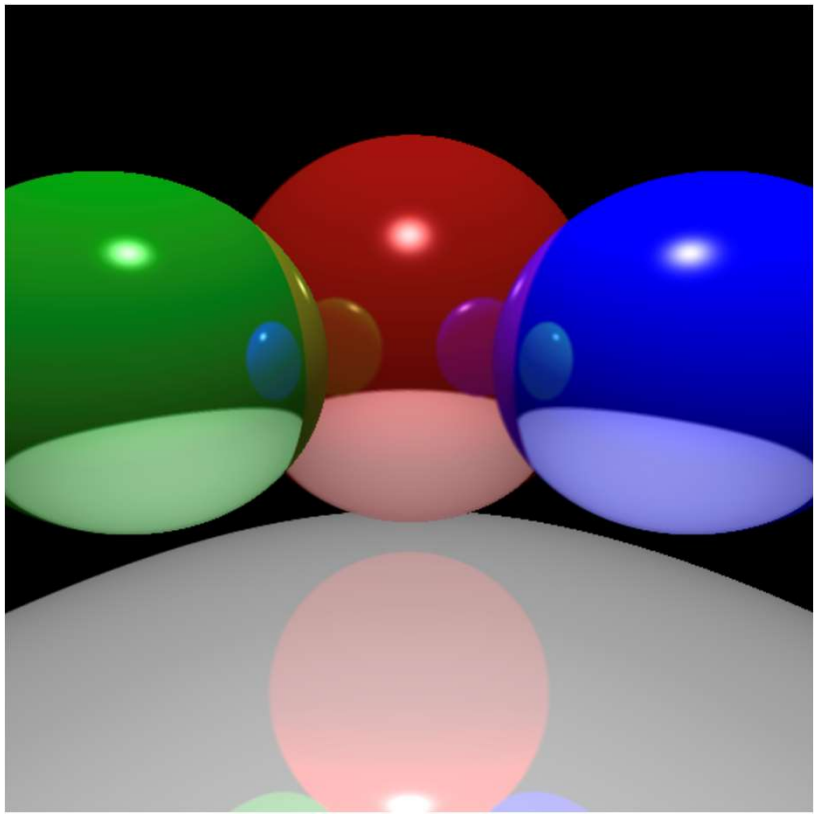


Ray Traced Reflection

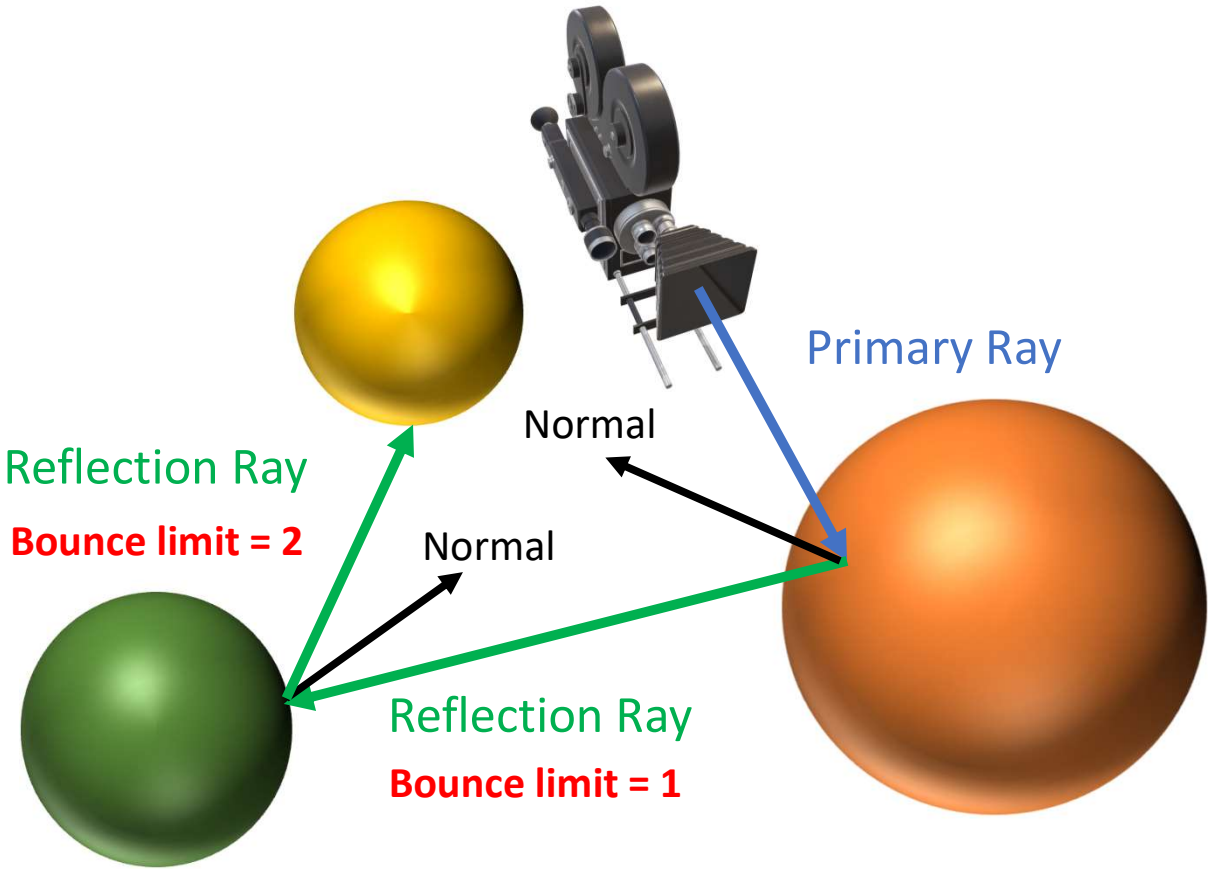
- Trace primary ray from camera/eye to compute the intersection point
- Compute the reflection vector
- Start a secondary (reflection) ray along reflection vector
  - Ray origin: Intersection point
  - Ray direction: from intersection point following the reflection vector
- Test intersection of reflection ray with all objects in the scene
- If the reflection ray hits any object in the scene, then the point will have reflection of the intersected object

Ray Traced Reflection

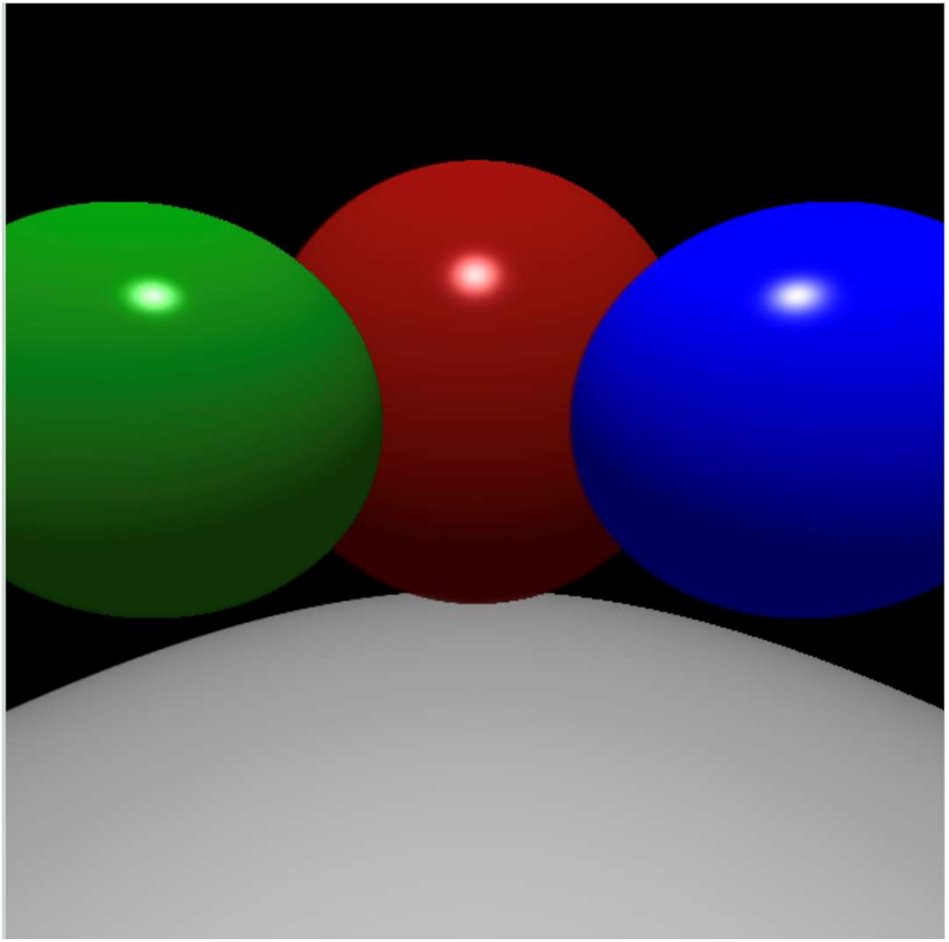




Reflection of Reflection

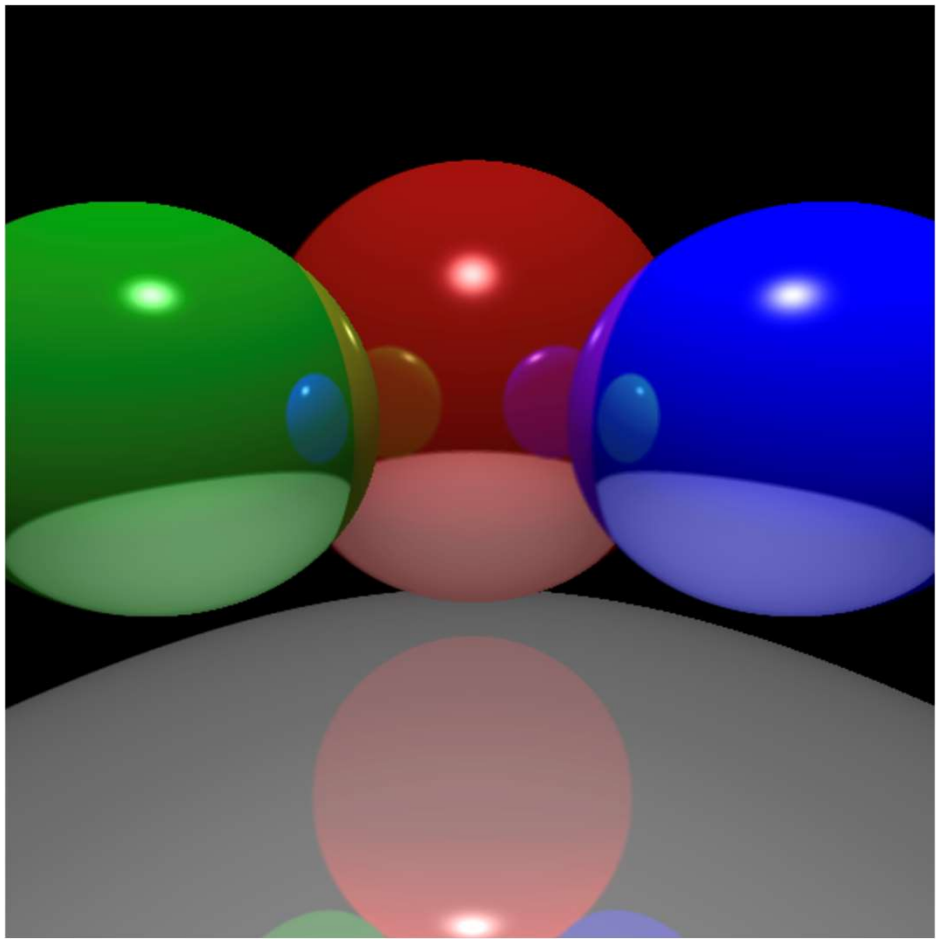


Reflection of Reflection of Reflection ……



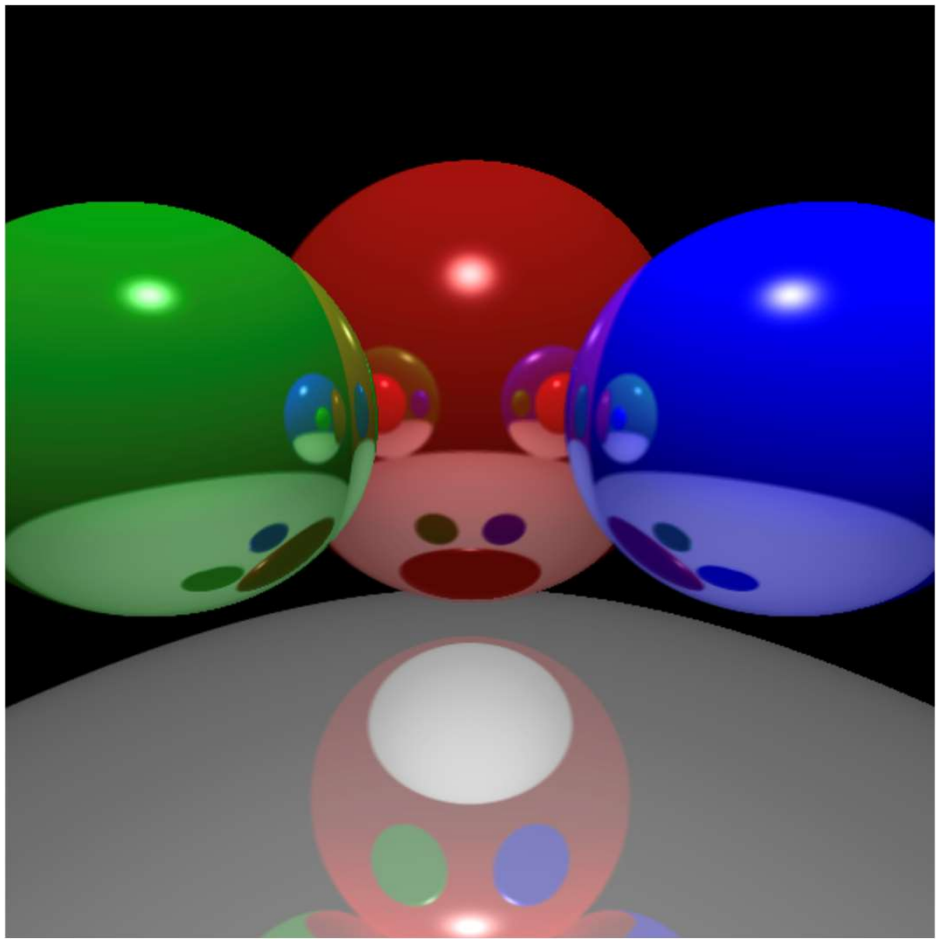
Bounce
limit = 0

Reflection of Reflection of Reflection ……



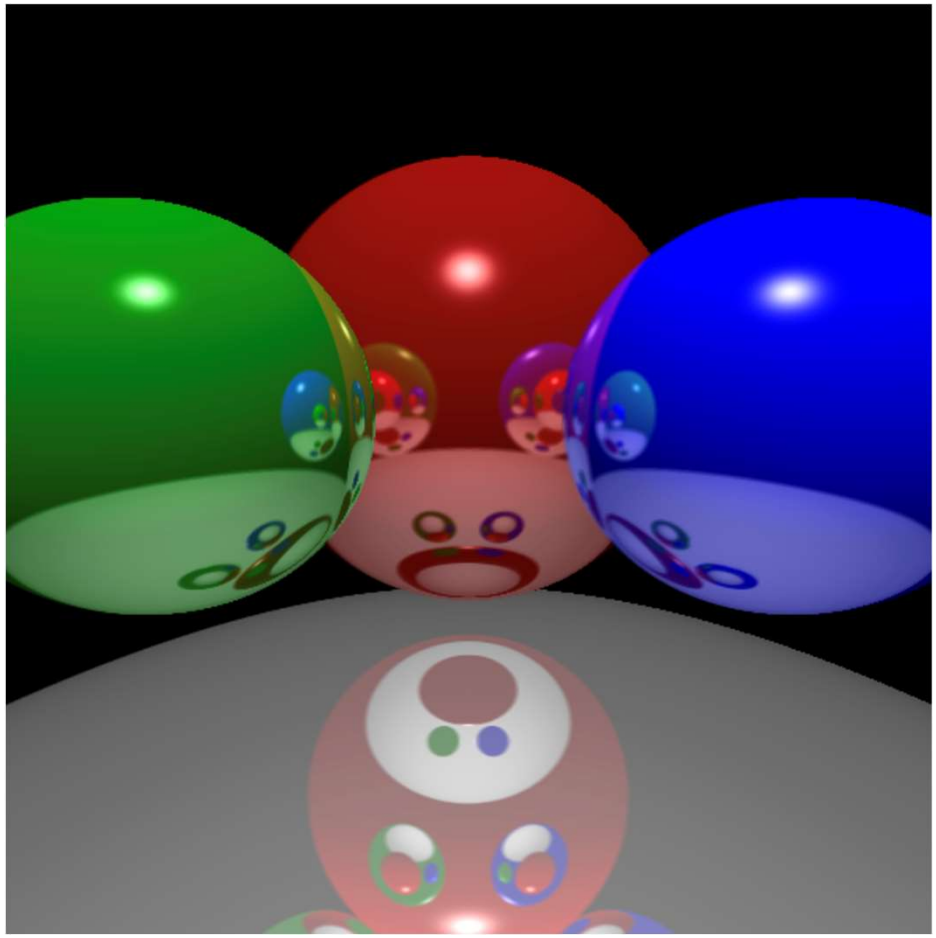
Bounce
limit = 1

Reflection of Reflection of Reflection ……



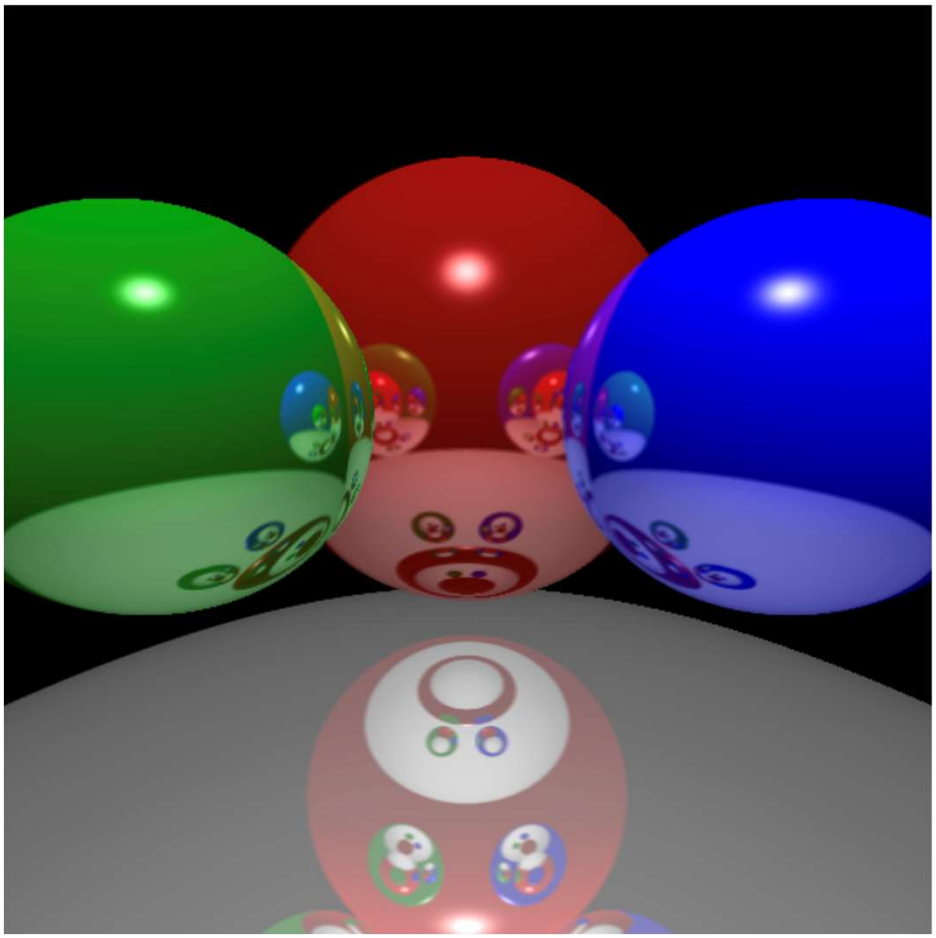
Bounce
limit = 2

Reflection of Reflection of Reflection ……



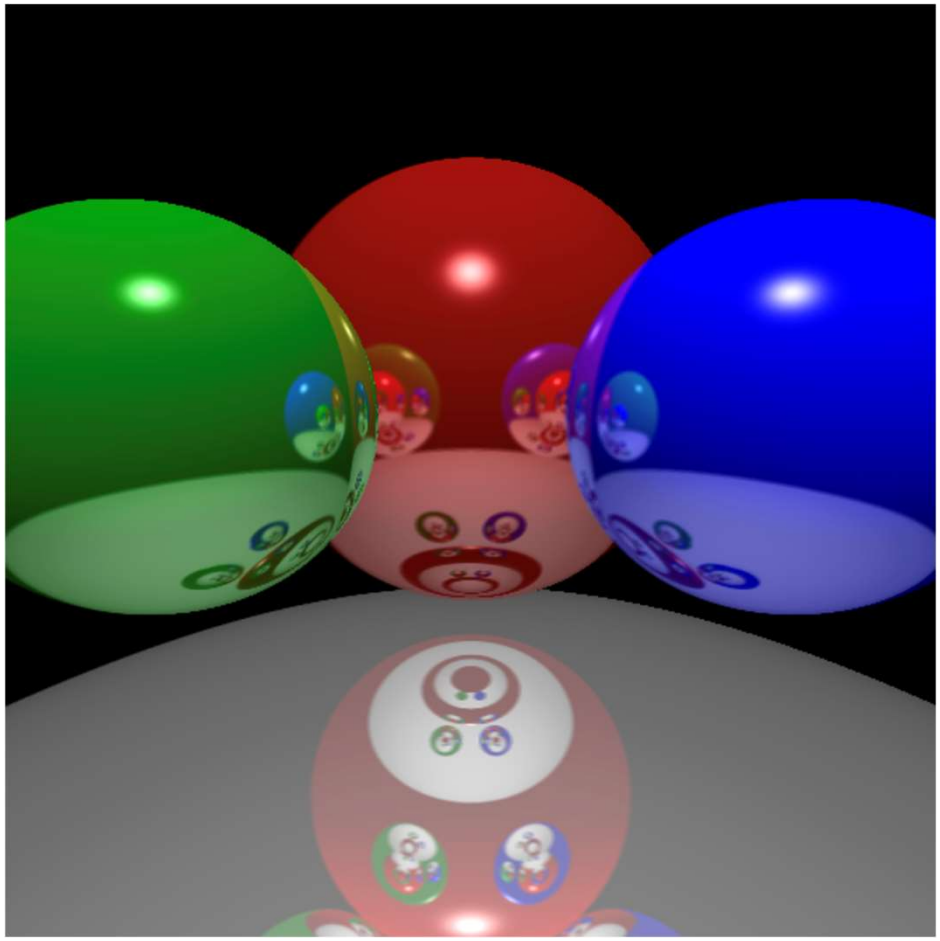
Bounce
limit = 3

Reflection of Reflection of Reflection ……



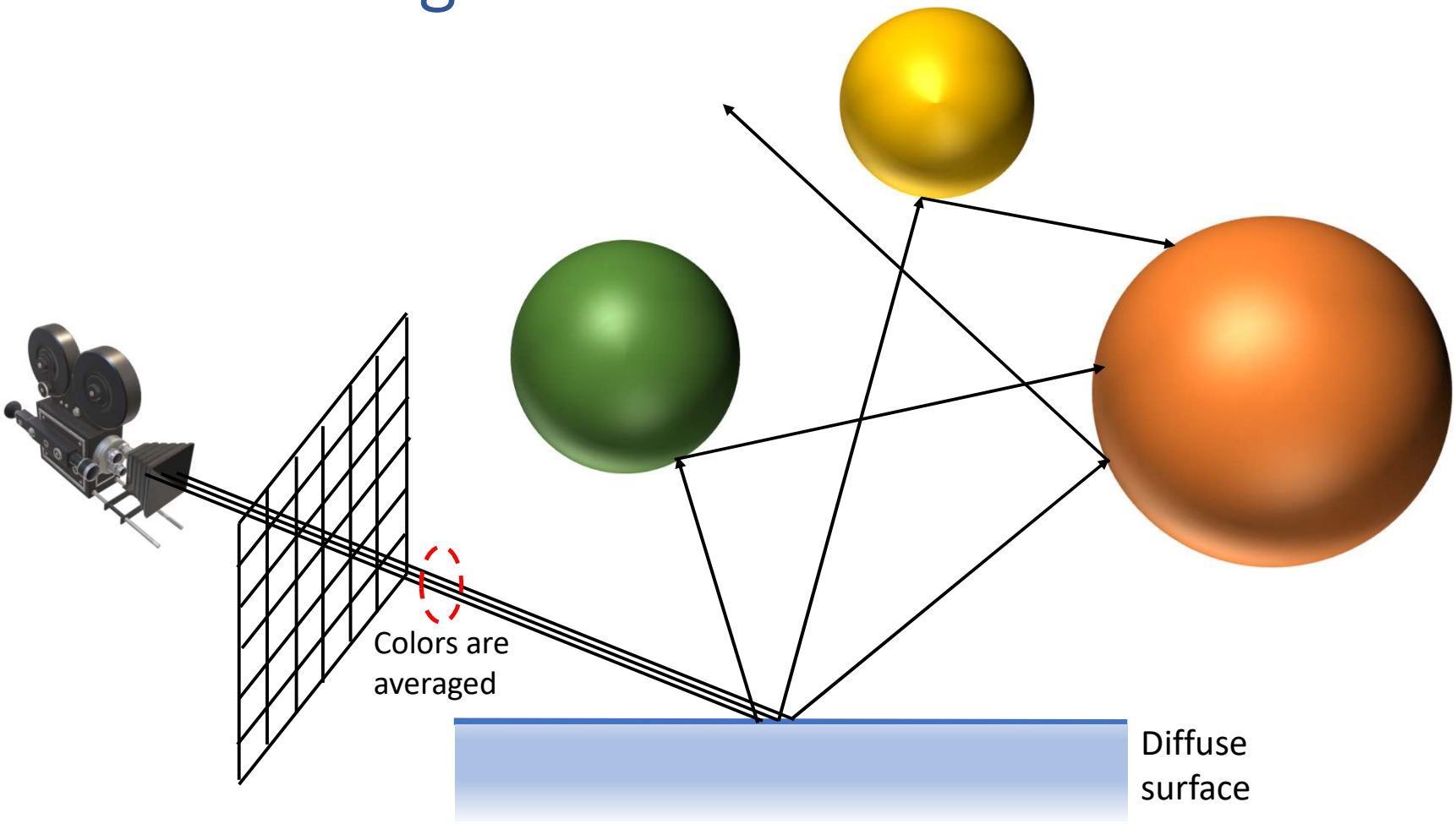
Bounce
limit = 4

Reflection of Reflection of Reflection ……



Bounce
limit = 5

Path Tracing: Global Illumination



Distributed Ray Tracing

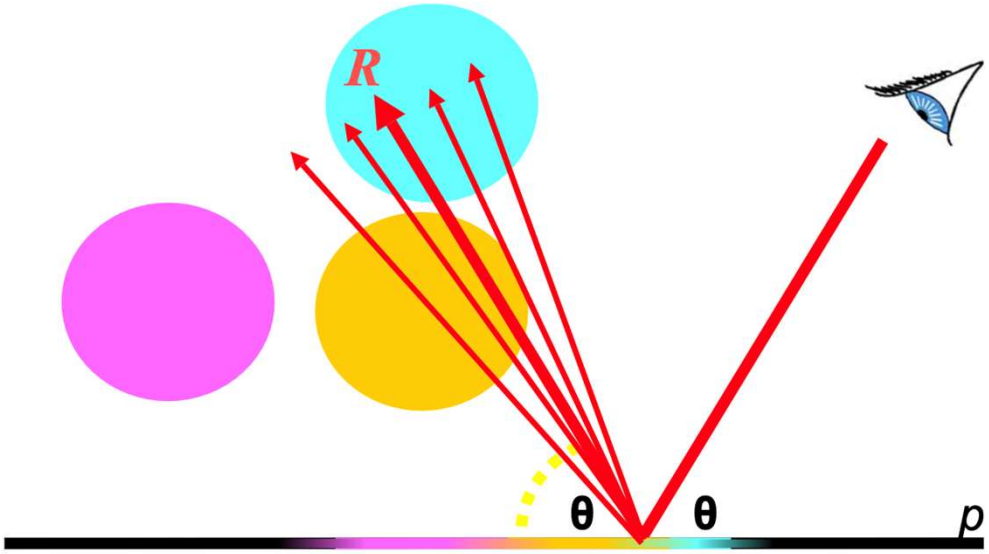
- Traditional ray tracing is limited to sharp shadow, sharp reflection, and sharp refraction
- Distributed ray tracing enables modeling “fuzzy” phenomena that we observe in nature
- Distributed ray tracing uses **oversampling** to reduce various aliasing artifacts and introduce fuzziness in the generated images

Distributed Ray Tracing

- Benefits of distributed ray tracing
  - Gloss (fuzzy reflections)
  - Soft shadows
  - Depth of field
  - Fuzzy translucency
  - Motion blur

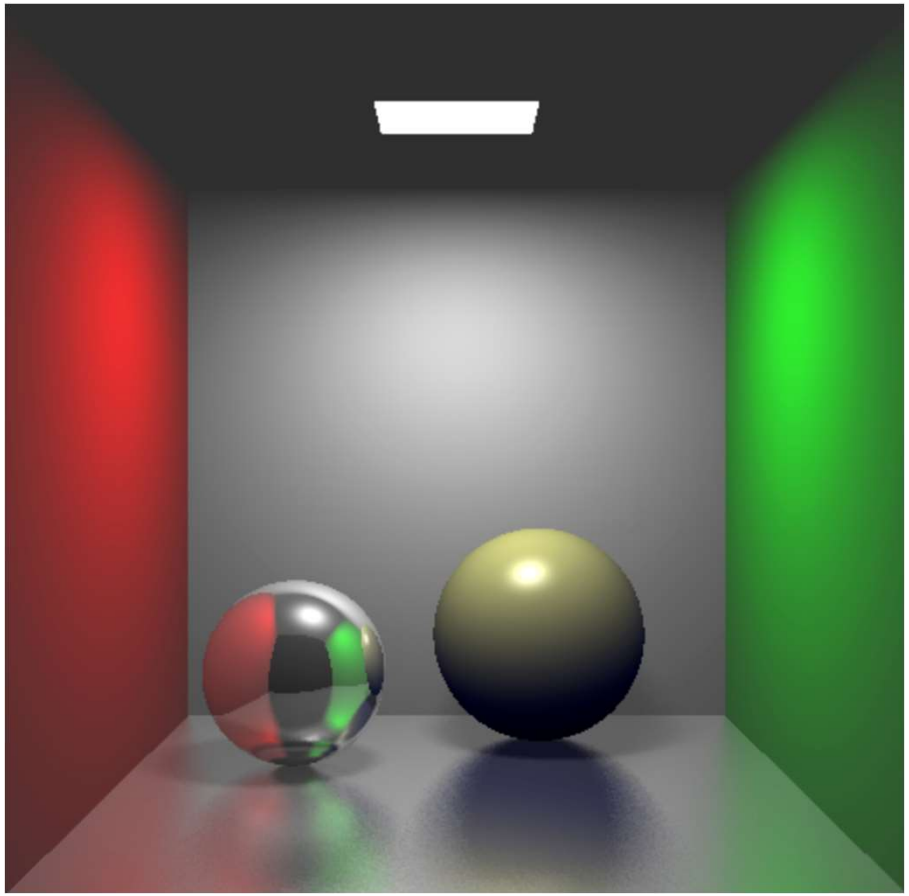
Gloss (Fuzzy Reflection)

- Randomly distribute many reflection rays from a surface point where reflection is being calculated
- Instead of casting a single ray out in the reflecting direction, a packet of rays are sent out around the reflecting direction
- Final value of reflection color can be found by taking the mean of the values returned by each of these rays



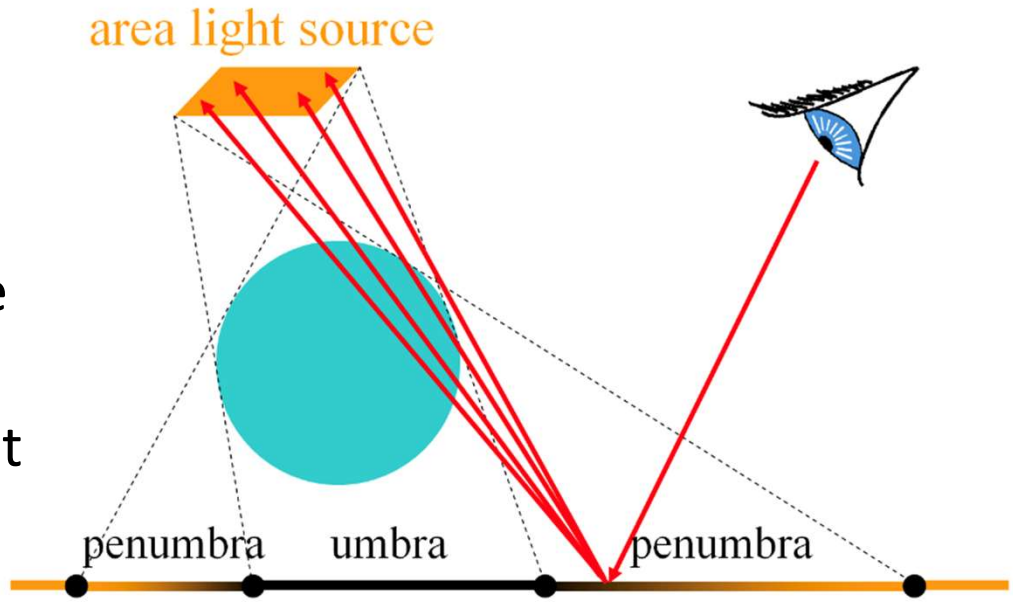
Gloss (Fuzzy Reflection)

- Randomly distribute many reflection rays from a surface point where reflection is being calculated
- Instead of casting a single ray out in the reflecting direction, a packet of rays are sent out around the reflecting direction
- Final value of reflection color can be found by taking the mean of the values returned by each of these rays



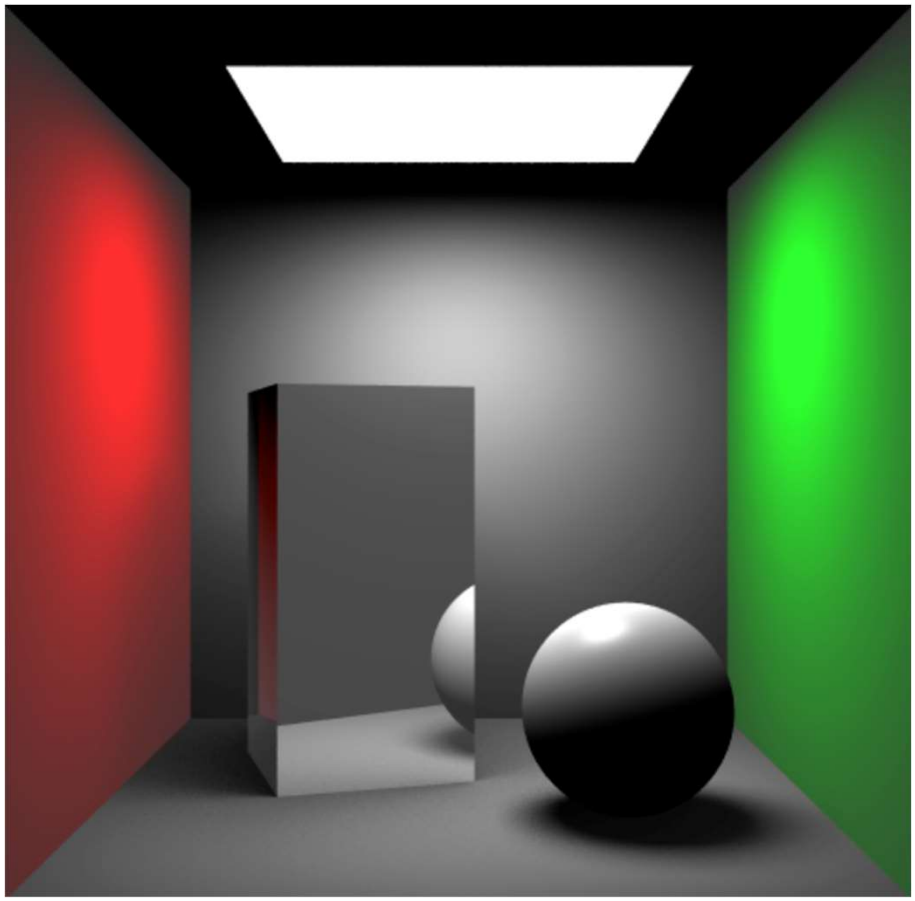
Soft Shadow

- Distributed ray tracing attempts to approximate soft shadows by modeling light sources as area source instead of point source
- From each point, a set of rays are cast to the area of the light source
- Final decision is made using the ratio of the number of rays that hit the light source to the number of rays cast

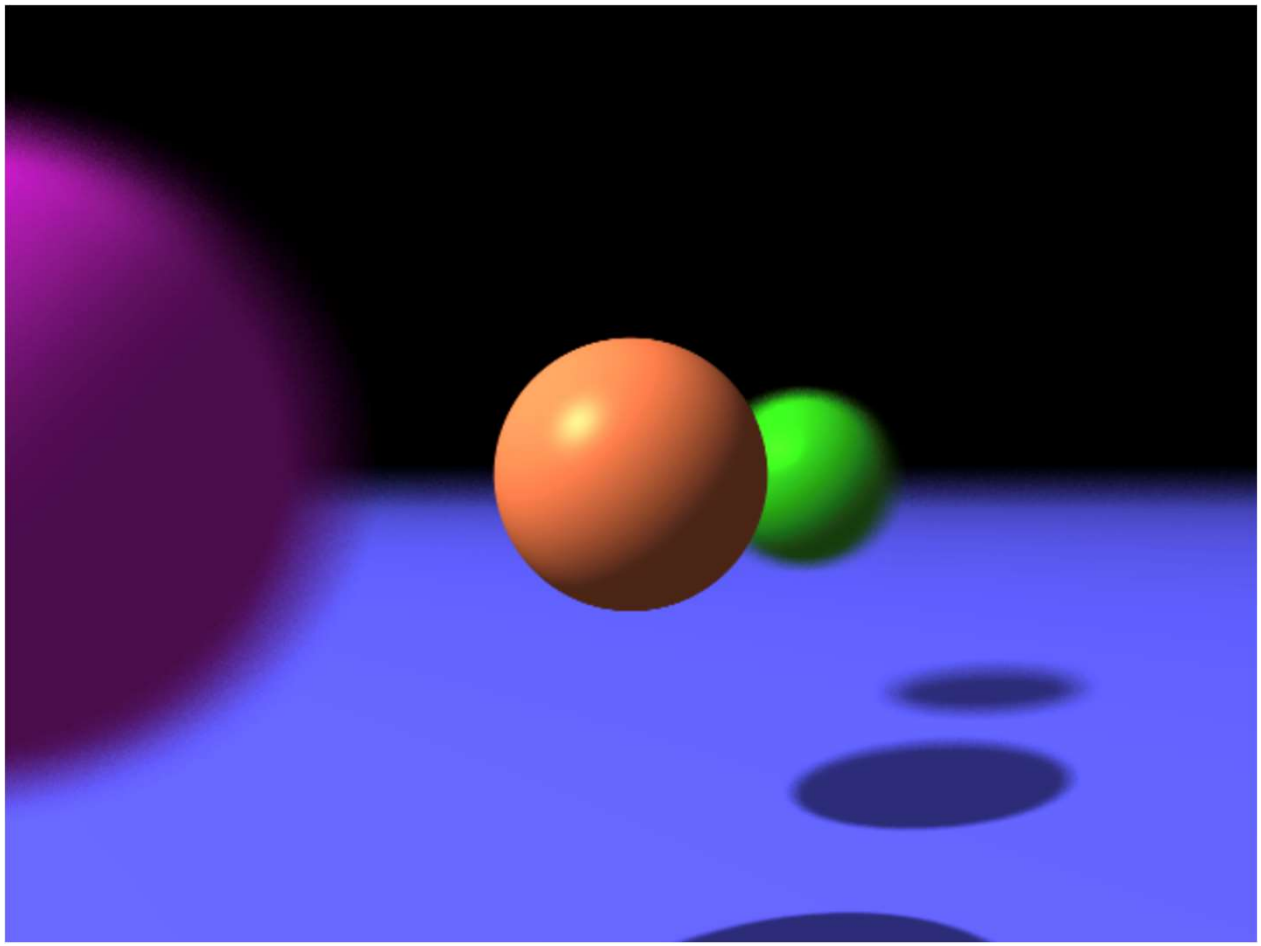


Soft Shadow

- Distributed ray tracing attempts to approximate soft shadows by modeling light sources as area source instead of point source
- From each point, a set of rays are cast to the area of the light source
- Final decision is made using the ratio of the number of rays that hit the light source to the number of rays cast

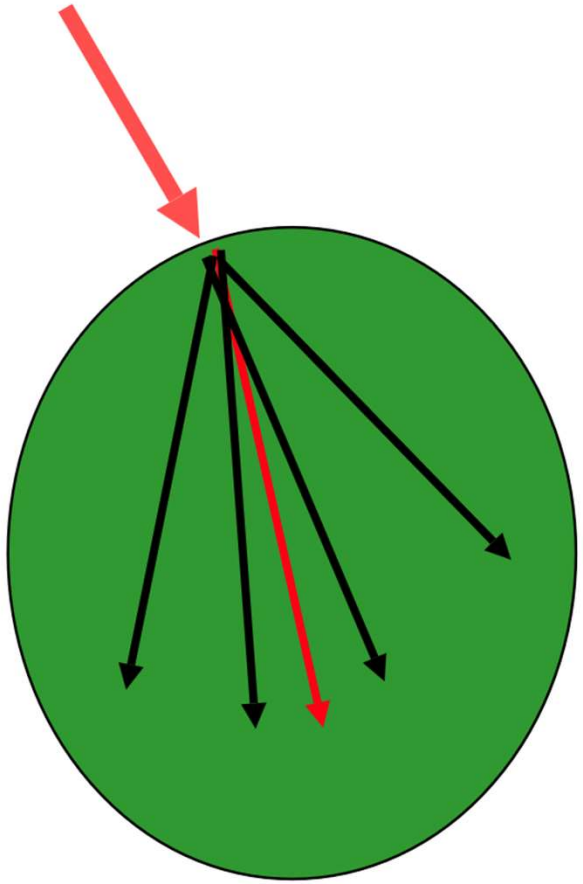


Depth of field

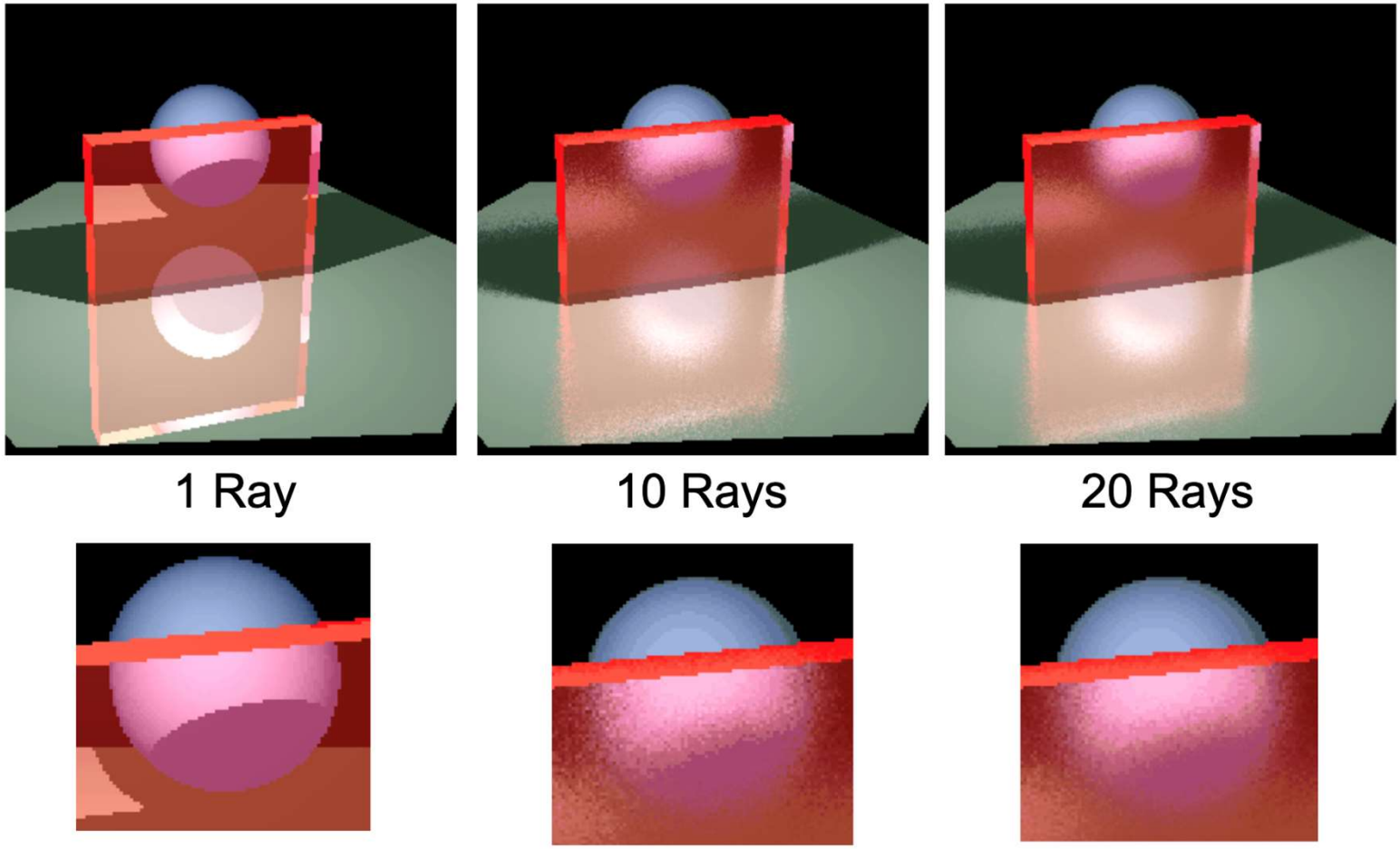


Fuzzy Translucency

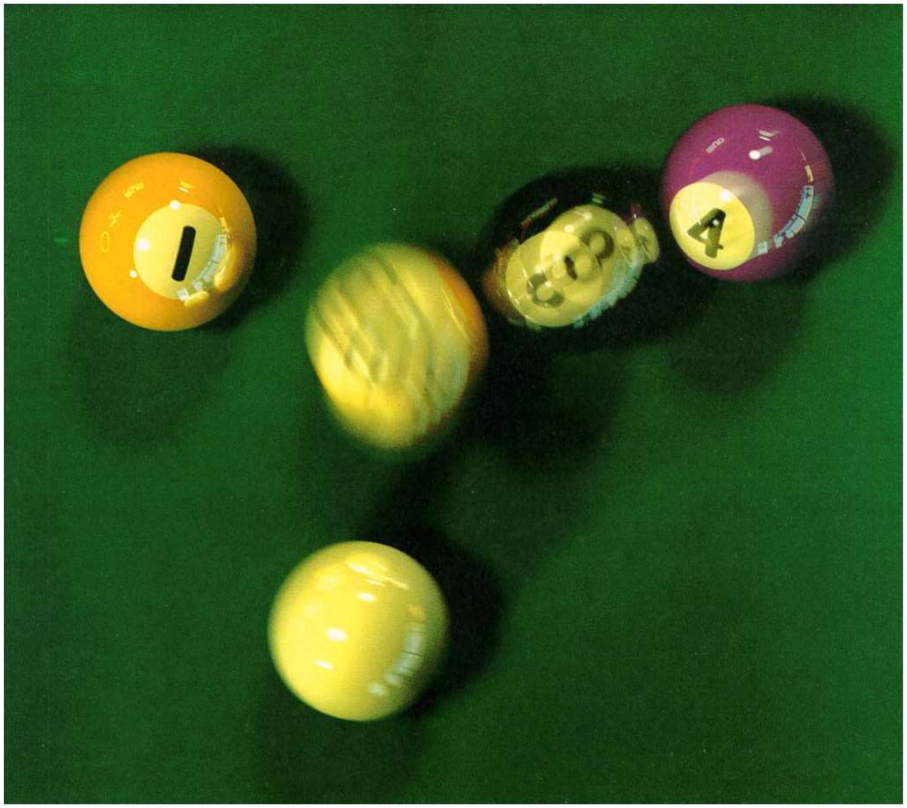
- Same as fuzzy reflection
- Instead of casting a single refracted ray, a packet of refraction rays are sent through the object
- Final value of refraction color can be found by taking the mean of the values returned by each of these rays

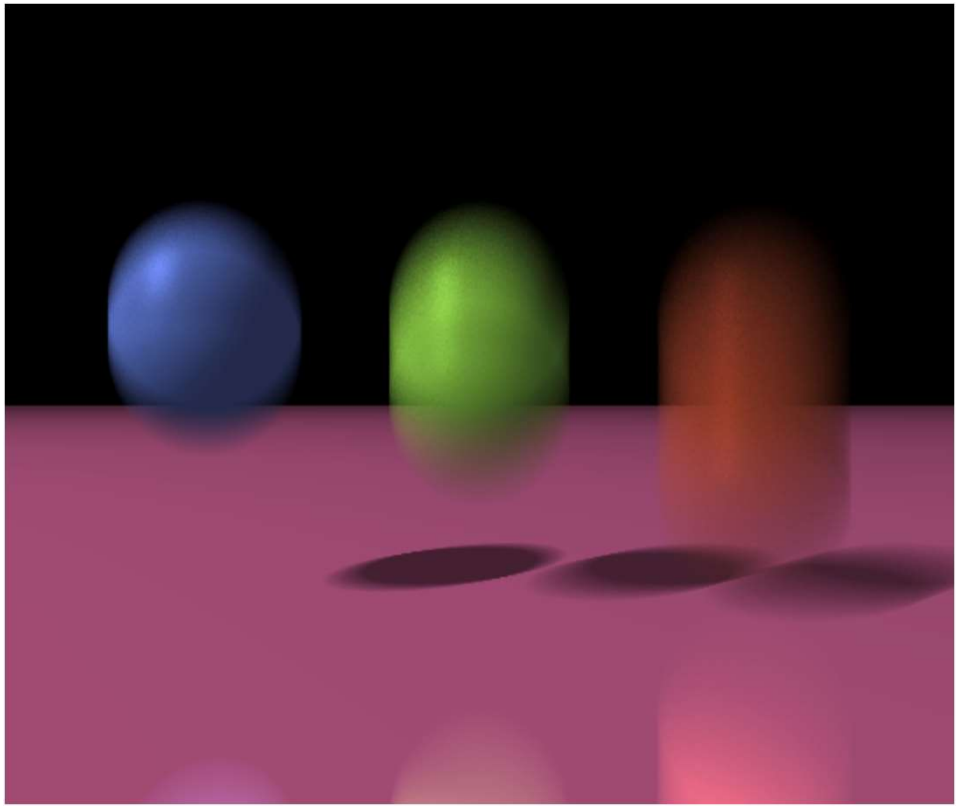


Fuzzy Translucency



Motion Blur





Implementation of Ray Tracer in Shader

- Draw a square with coordinates defined directly in clip space

Inside drawScene():

// buffer for the square to cover entire viewport [-1,1]

```
const bufData = new Float32Array([
-1, 1, 0, 1, 1, 0, -1, -1, 0, -1, -1, 0, 1, 1, 0, 1, -1, 0,]);
```

// Draw the square

```
gl.drawArrays(gl.TRIANGLES, 0, 6);
```

Implementation of Ray Tracer in Shader

- **Vertex shader:**

```
const vertexShaderCode = `#version 300 es
in vec3 aPosition;
void main() {
gl_Position = vec4(aPosition,1.0);
}`;
```

Implementation of Ray Tracer in Shader

- **Fragment shader:**

```
struct Sphere {
vec3 center;
float radius;
vec3 color;
};

struct Ray {
vec3 origin;
vec3 direction;
};
```

Implementation of Ray Tracer in Shader

- **Fragment shader:**

```
Sphere sphere;

// Create a sphere
sphere.center = vec3(0.0, 0.0, 0.0);
sphere.radius = 1.0;
sphere.color = vec3(0.0, 0.8, 0.8);
```

Implementation of Ray Tracer in Shader

- **Fragment shader:**

```
Ray ray;

// create the ray for current frag
ray.origin = cameraPos;

// direction is through each screen fragment in negative z direction
vec2 screenPos = gl_FragCoord.xy/vec2(canvWidth, canvHeight);
ray.direction = normalize(vec3(screenPos * 2.0 - 1.0, -1.0));
```