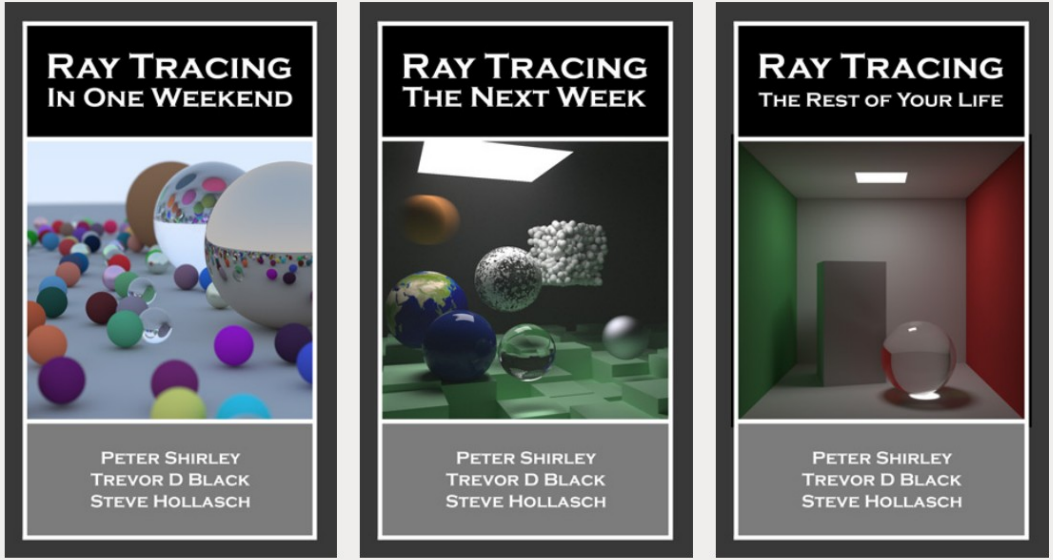
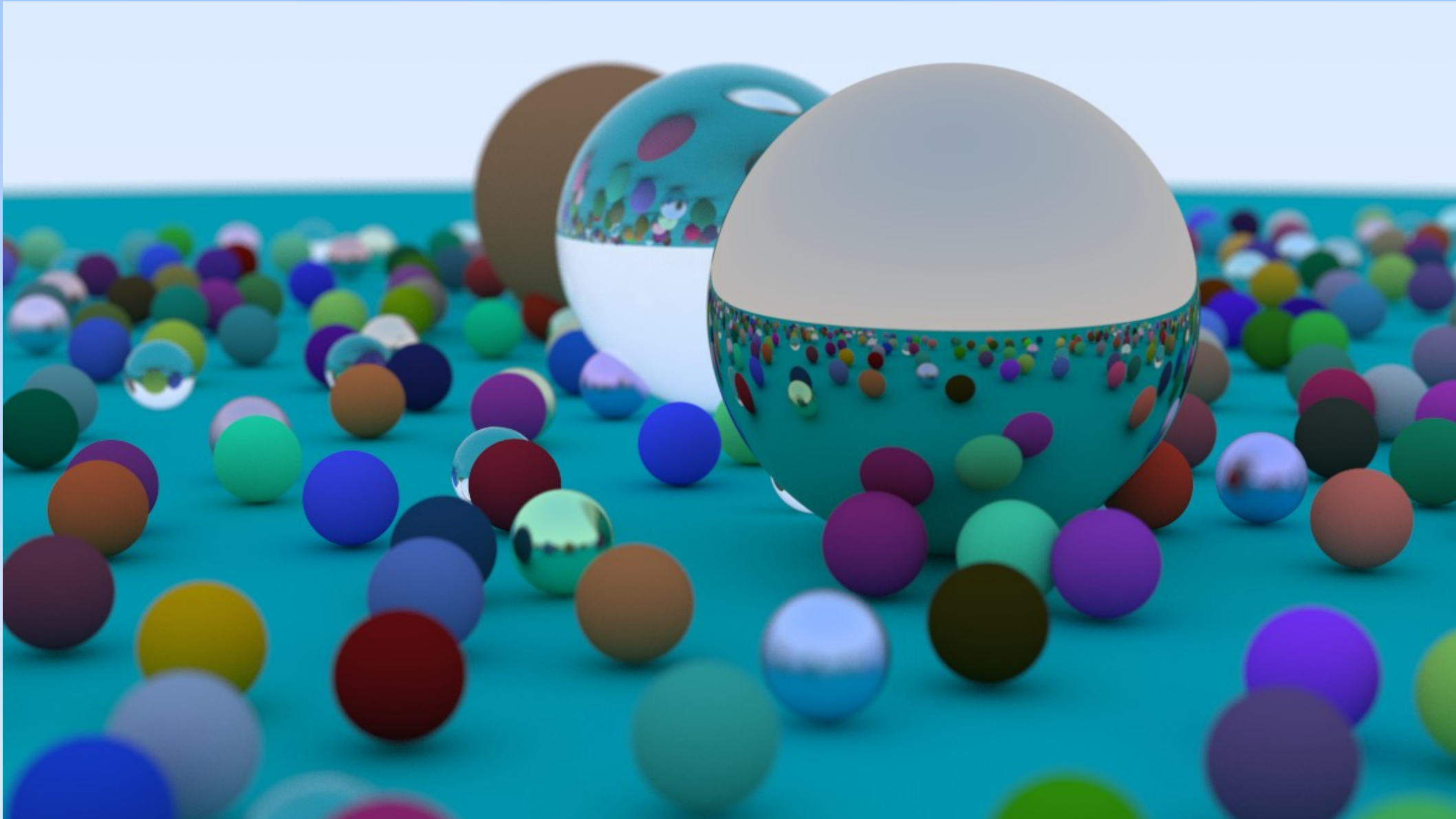
Writing Your Own Ray Tracer From Scratch Is Wasting Your Time

Ray Tracing in One Weekend

—

The Book Series

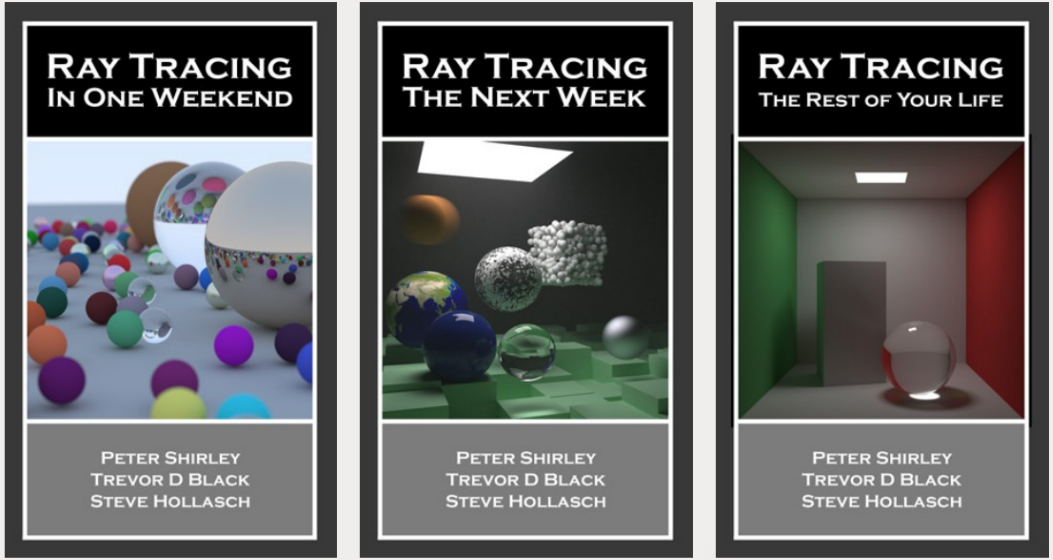




Ray Tracing in One Weekend

—

The Book Series



WOW

Roadmap

- Write all in C++
- Get Image
- Get Rays
- Get Colors
- Get Objects
- Profit?

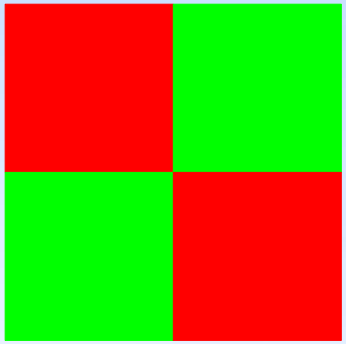
Write C++

```
// Vector Utility Functions

inline std::ostream& operator<<(std::ostream& out, const vec3& v) {
    return out << v.e[0] << ' ' << v.e[1] << ' ' << v.e[2];
}

inline vec3 operator+(const vec3& u, const vec3& v) {
    return vec3(u.e[0] + v.e[0], u.e[1] + v.e[1], u.e[2] + v.e[2]);
}

inline vec3 operator-(const vec3& u, const vec3& v) {
    return vec3(u.e[0] - v.e[0], u.e[1] - v.e[1], u.e[2] - v.e[2]);
}

inline vec3 operator*(const vec3& u, const vec3& v) {
    return vec3(u.e[0] * v.e[0], u.e[1] * v.e[1], u.e[2] * v.e[2]);
}

inline vec3 operator*(double t, const vec3& v) {
    return vec3(t*v.e[0], t*v.e[1], t*v.e[2]);
}
```

Get Image – Image.ppm

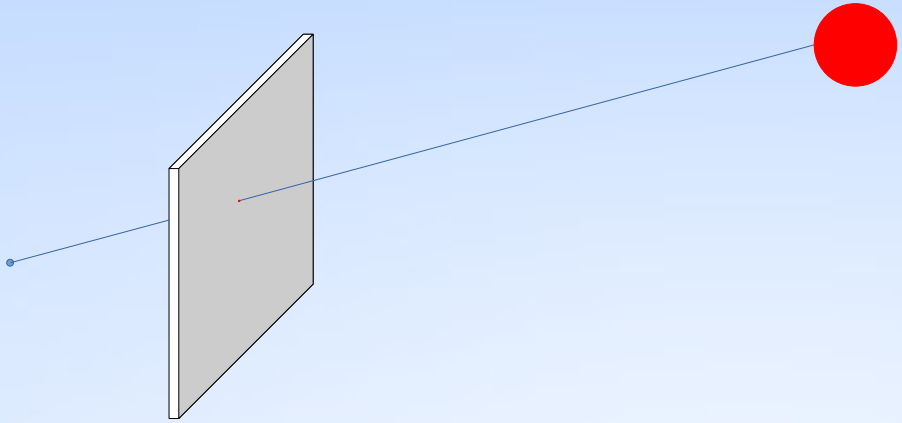
- Header
- RGB Values for each pixel

```
P3
2 2
255
255 0 0    0 255 0
0 255 0    255 0 0
```



Get Rays – Vectors with Distance

- Shoot Rays Through Viewport
- If Distance Infinity: Return Color of Background
- If Hit: Return Color of Object



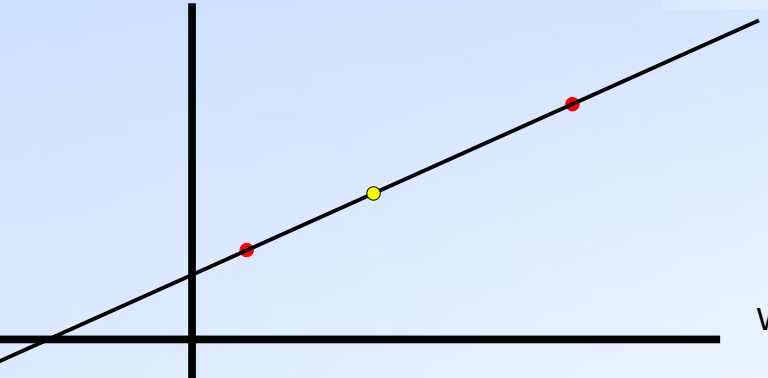
Write C++

```
// Vector Utility Functions

inline std::ostream& operator<<(std::ostream& out, const vec3& v) {
    return out << v.e[0] << ' ' << v.e[1] << ' ' << v.e[2];
}

inline vec3 operator+(const vec3& u, const vec3& v) {
    return vec3(u.e[0] + v.e[0], u.e[1] + v.e[1], u.e[2] + v.e[2]);
}

inline vec3 operator-(const vec3& u, const vec3& v) {
    return vec3(u.e[0] - v.e[0], u.e[1] - v.e[1], u.e[2] - v.e[2]);
}

inline vec3 operator*(const vec3& u, const vec3& v) {
    return vec3(u.e[0] * v.e[0], u.e[1] * v.e[1], u.e[2] * v.e[2]);
}

inline vec3 operator*(double t, const vec3& v) {
    return vec3(t*v.e[0], t*v.e[1], t*v.e[2]);
}
```

First Gradien Image

Lerp

Linear Int**erp**olation



Get Object – Spheres

- Create Spheres as Objects
- “Simple” Formula: $x^2 + y^2 + z^2 = r^2$
- On Hit Return Color

Write C++

```
// Vector Utility Functions

inline std::ostream& operator<<(std::ostream& out, const vec3& v) {
    return out << v.e[0] << ' ' << v.e[1] << ' ' << v.e[2];
}

inline vec3 operator+(const vec3& u, const vec3& v) {
    return vec3(u.e[0] + v.e[0], u.e[1] + v.e[1], u.e[2] + v.e[2]);
}

inline vec3 operator-(const vec3& u, const vec3& v) {
    return vec3(u.e[0] - v.e[0], u.e[1] - v.e[1], u.e[2] - v.e[2]);
}

inline vec3 operator*(const vec3& u, const vec3& v) {
    return vec3(u.e[0] * v.e[0], u.e[1] * v.e[1], u.e[2] * v.e[2]);
}

inline vec3 operator*(double t, const vec3& v) {
    return vec3(t*v.e[0], t*v.e[1], t*v.e[2]);
}
```

First Sphere(?)

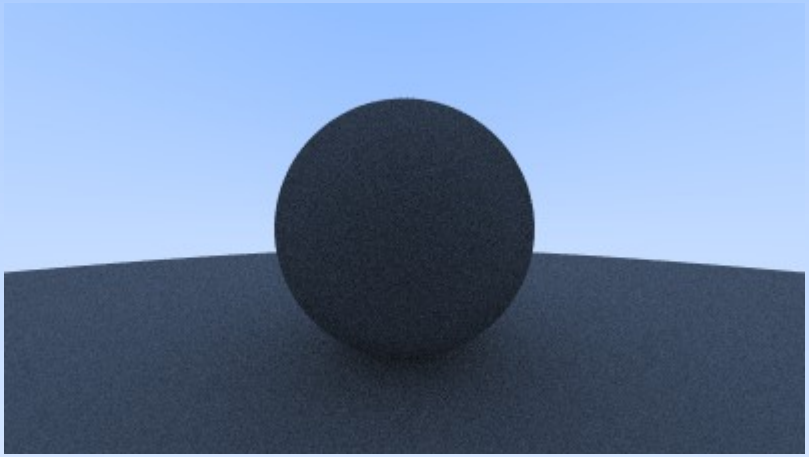
Write C++

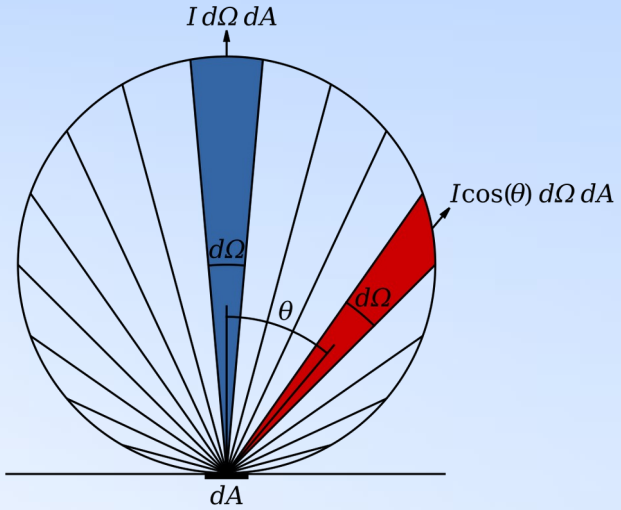
```
// Vector Utility Functions

inline std::ostream& operator<<(std::ostream& out, const vec3& v) {
    return out << v.e[0] << ' ' << v.e[1] << ' ' << v.e[2];
}

inline vec3 operator+(const vec3& u, const vec3& v) {
    return vec3(u.e[0] + v.e[0], u.e[1] + v.e[1], u.e[2] + v.e[2]);
}

inline vec3 operator-(const vec3& u, const vec3& v) {
    return vec3(u.e[0] - v.e[0], u.e[1] - v.e[1], u.e[2] - v.e[2]);
}

inline vec3 operator*(const vec3& u, const vec3& v) {
    return vec3(u.e[0] * v.e[0], u.e[1] * v.e[1], u.e[2] * v.e[2]);
}

inline vec3 operator*(double t, const vec3& v) {
    return vec3(t*v.e[0], t*v.e[1], t*v.e[2]);
}
```

Include Surface Normals

Spheres: Diffuse Materials

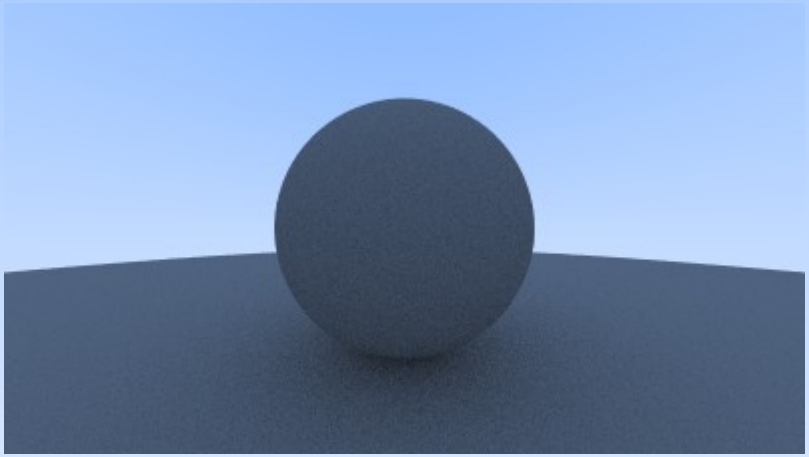
- Lambertian Reflection





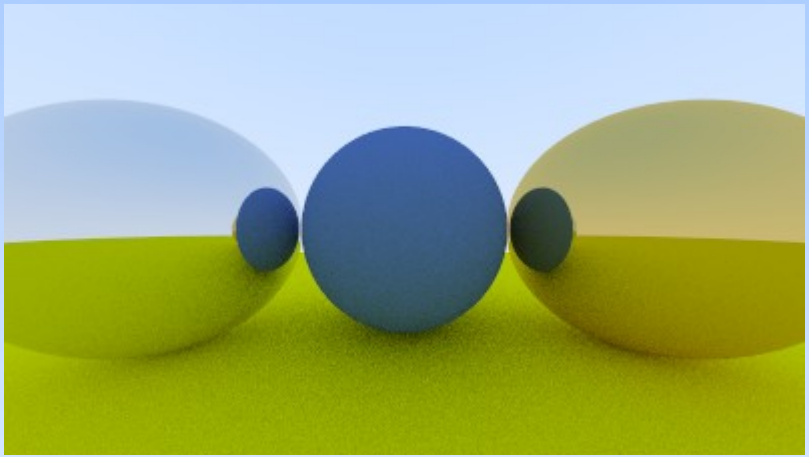
Sphere: Shadow Acne

- Floating Point Errors
- Ray Intersects Inside Sphere?
- Insert Padding
- 0.000001



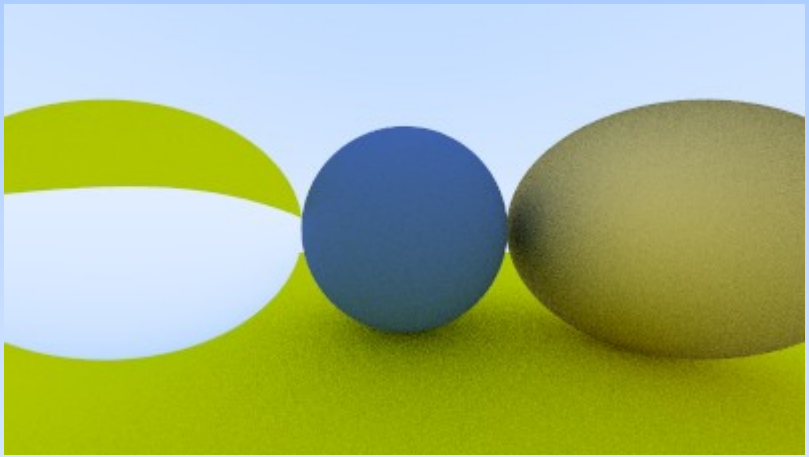
Spheres: Shiny Materials

- Incoming Angle =
  Outgoing Angle



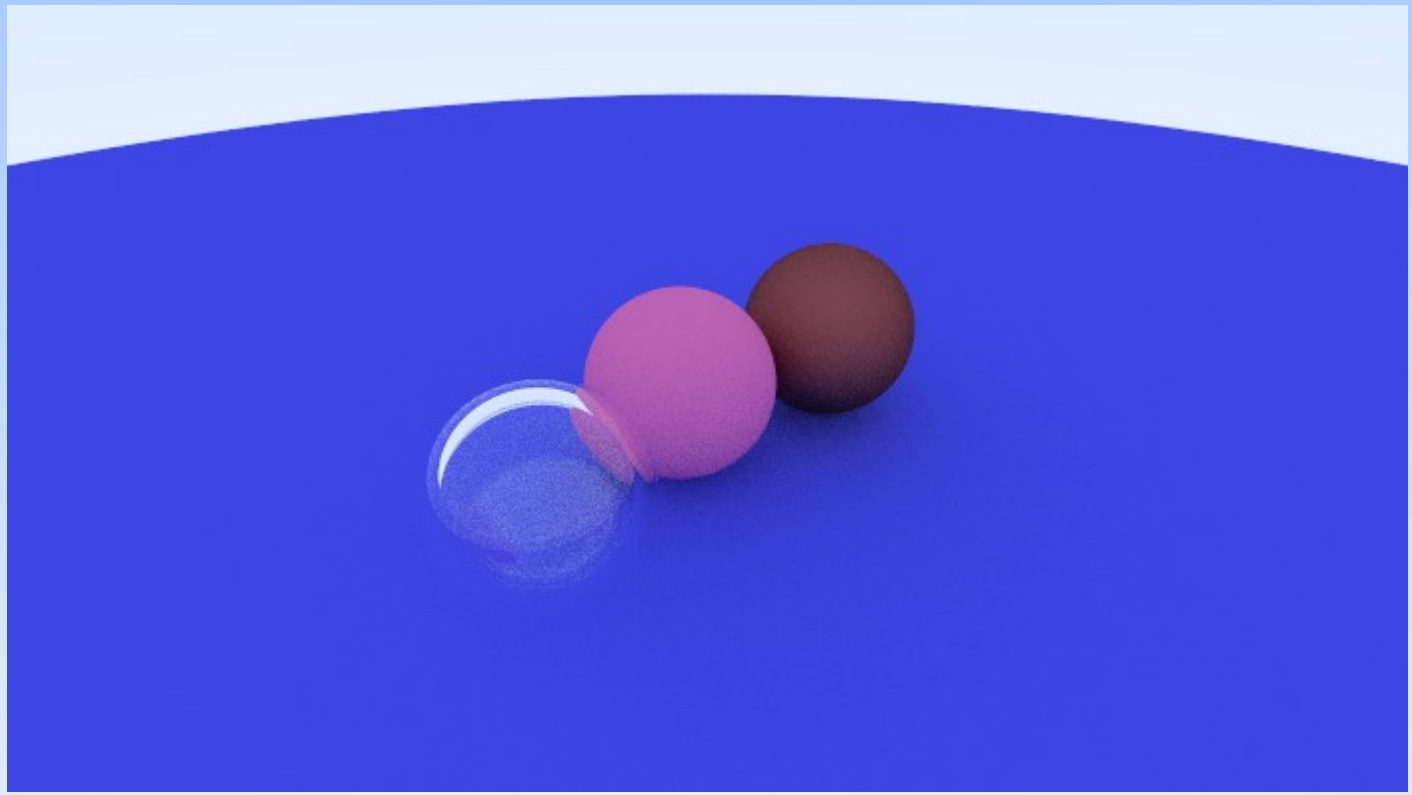
Spheres: Glass Materials

- Snells Law
- Ideal Refraction

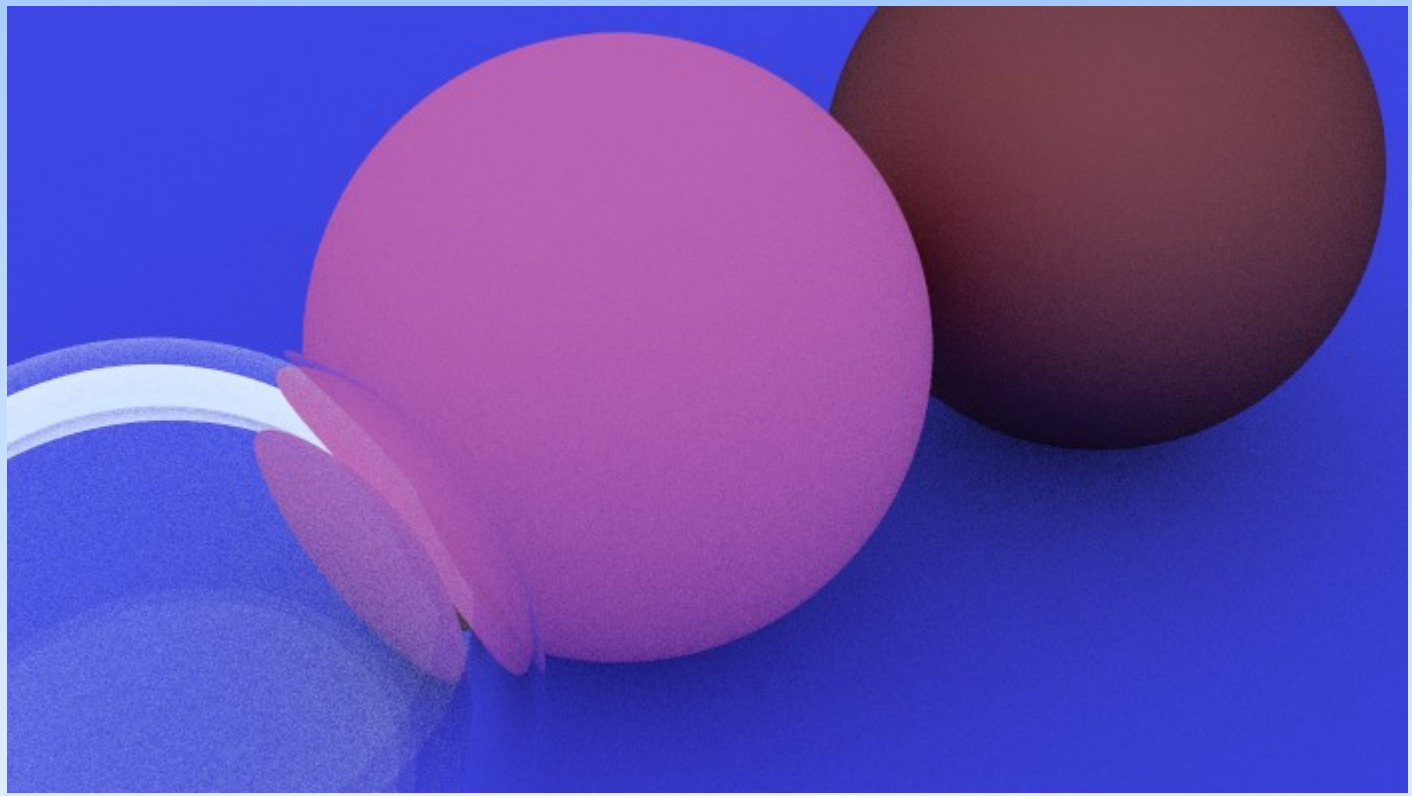


$$\frac{\sin\theta_1}{\sin\theta_2} = n_{2,1} = \frac{n_2}{n_1} = \frac{v_1}{v_2}$$

Changing The View

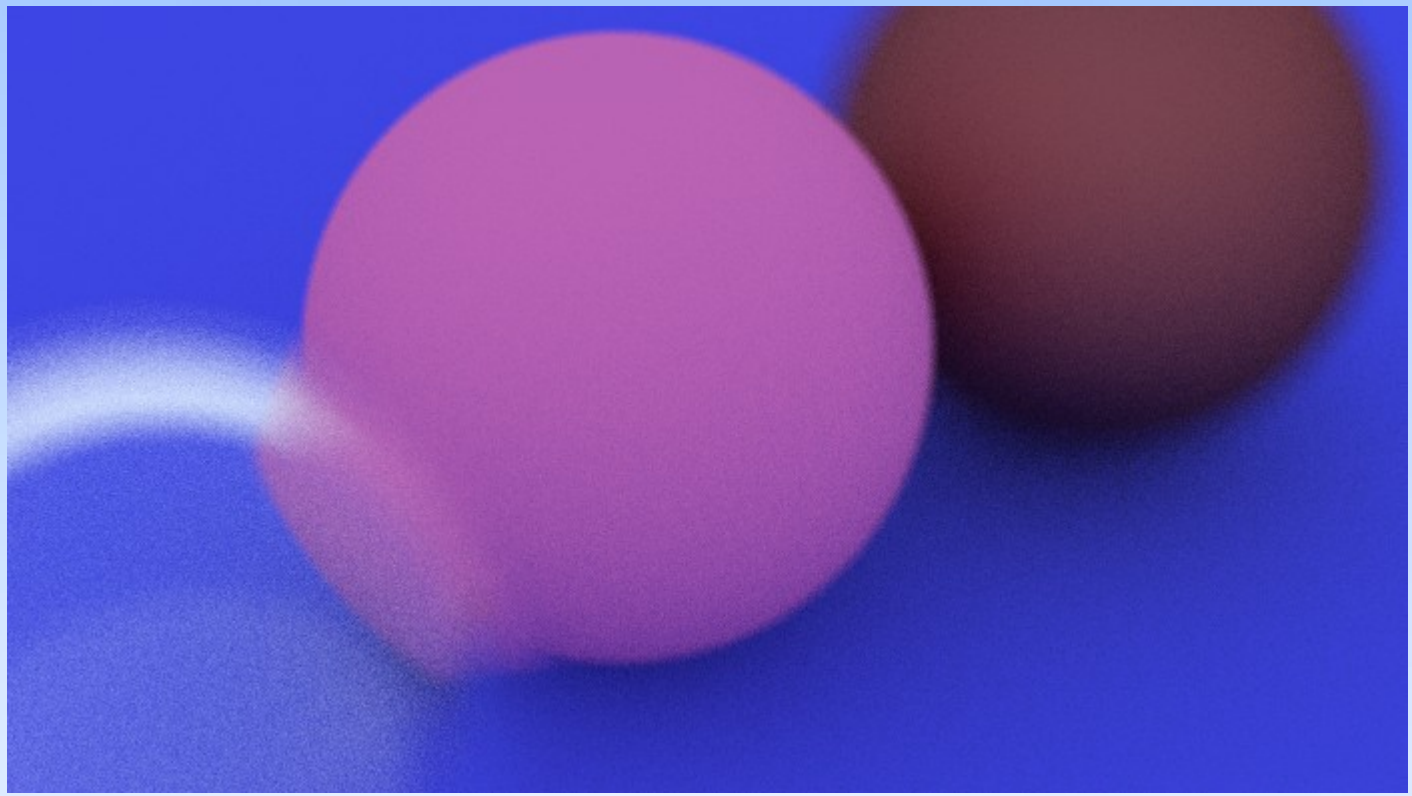


Zooming In



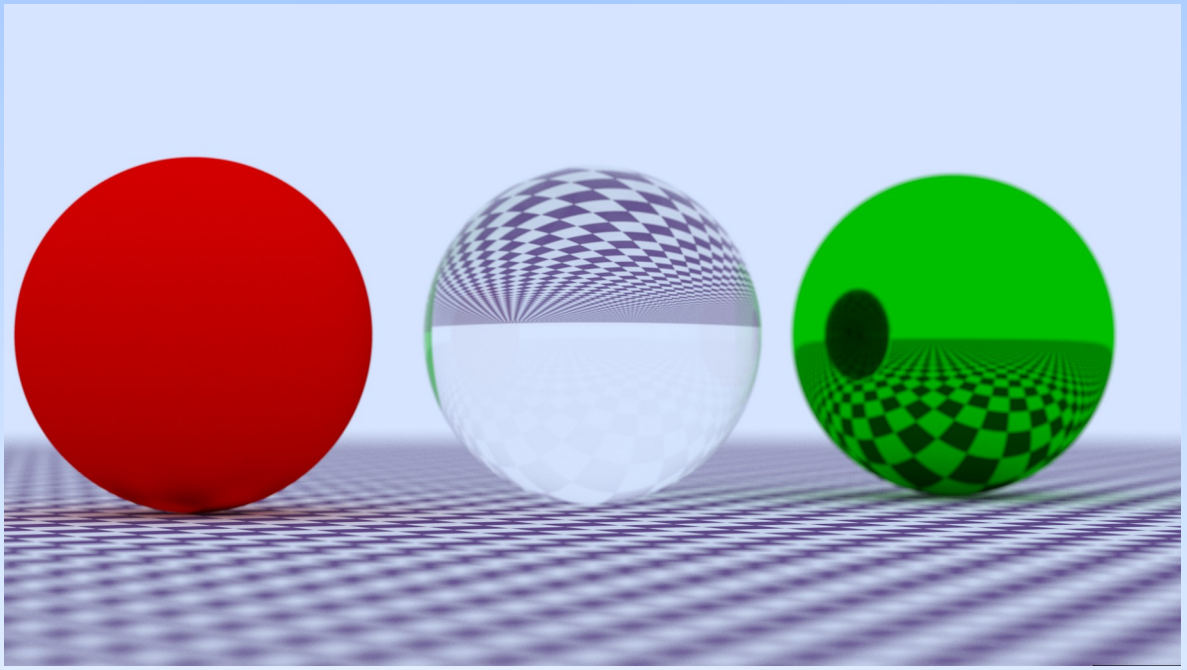
Defocus Blur

- Create Fuzz
- Fuzz by Creating Random Values On Unit Sphere

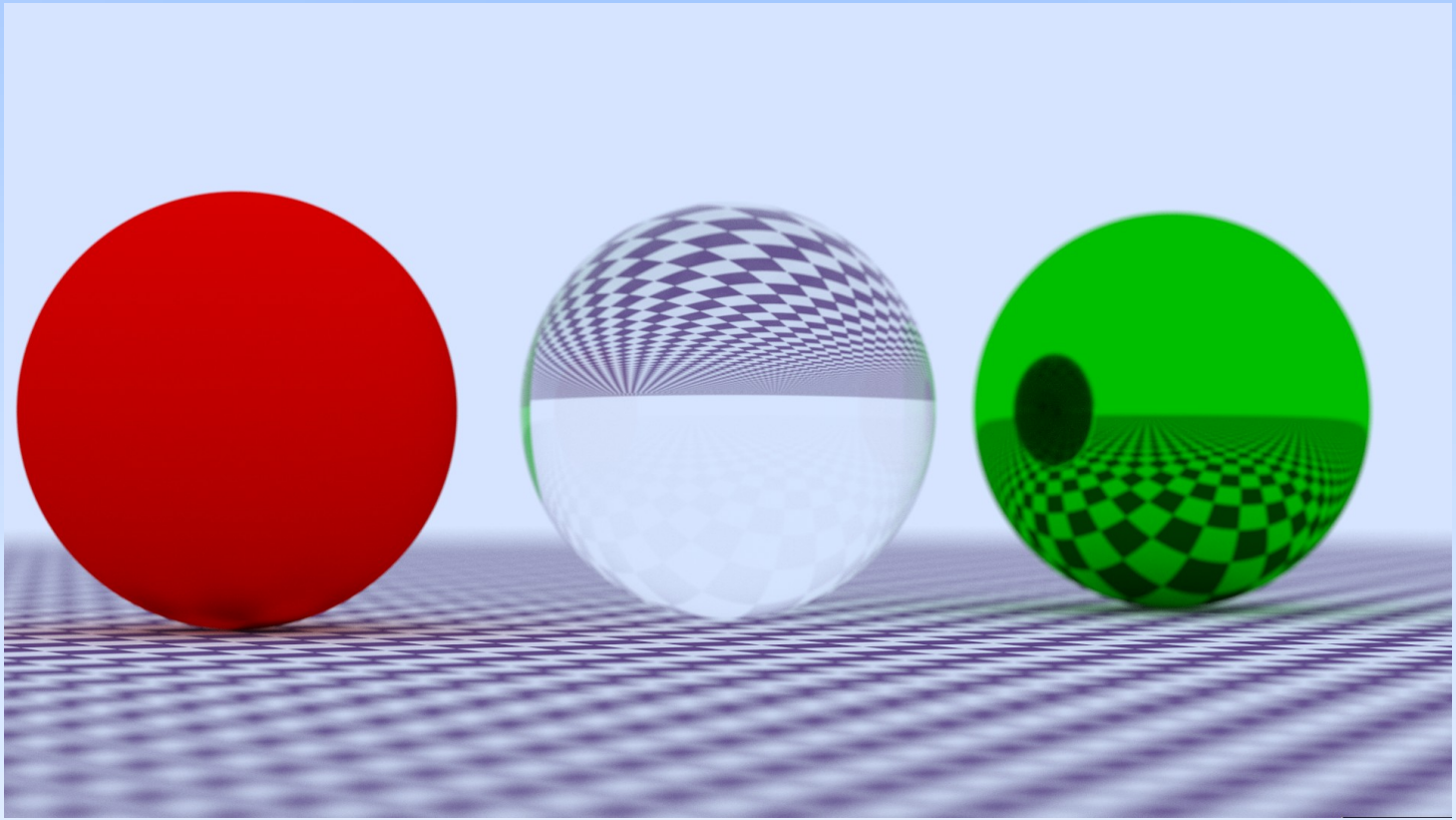


Defocus Blur - Issues

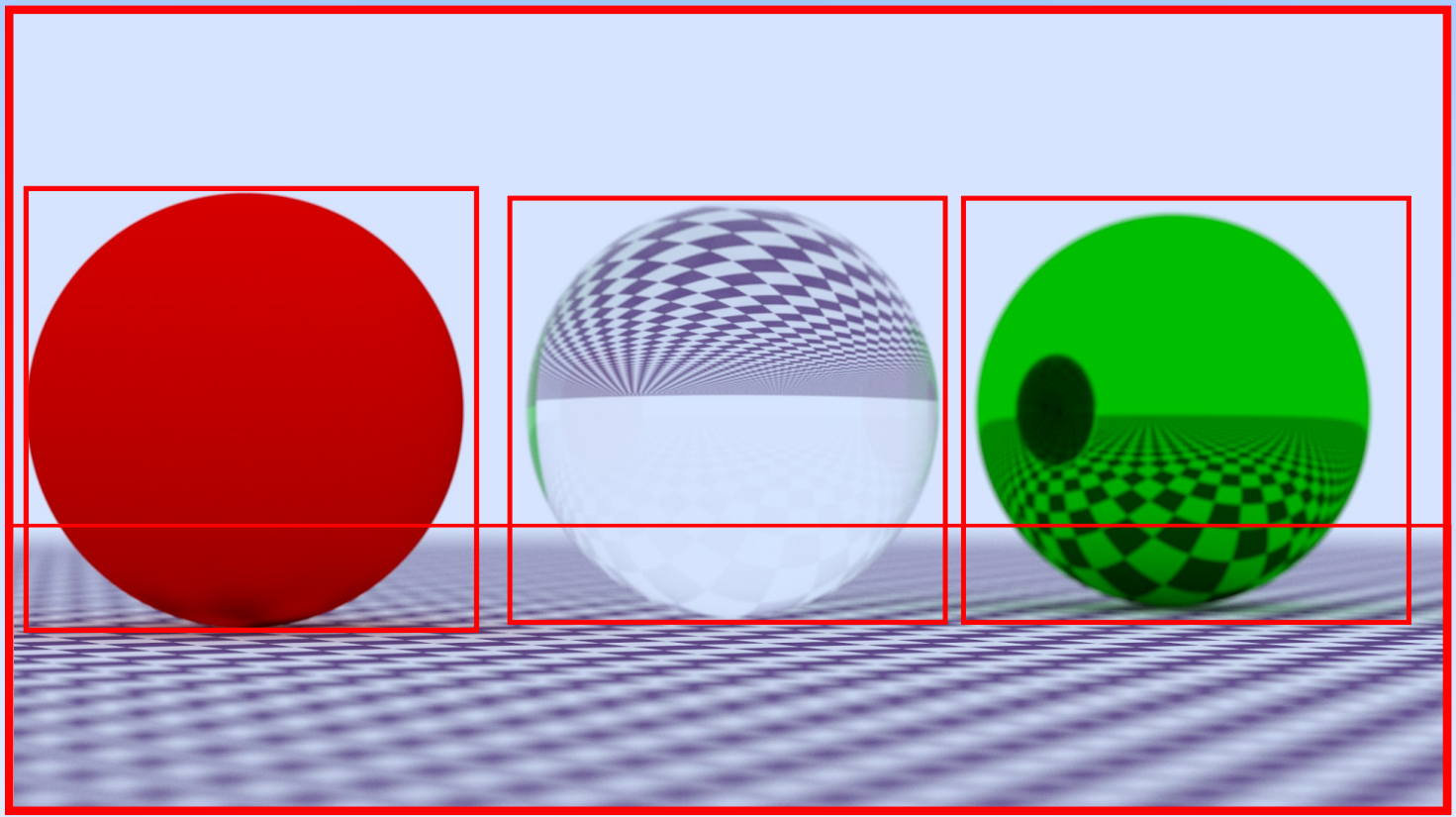
- Create Fuzz
at given Distance
- Glass Behind Sphere
Still Sharp



Why Wasting Time



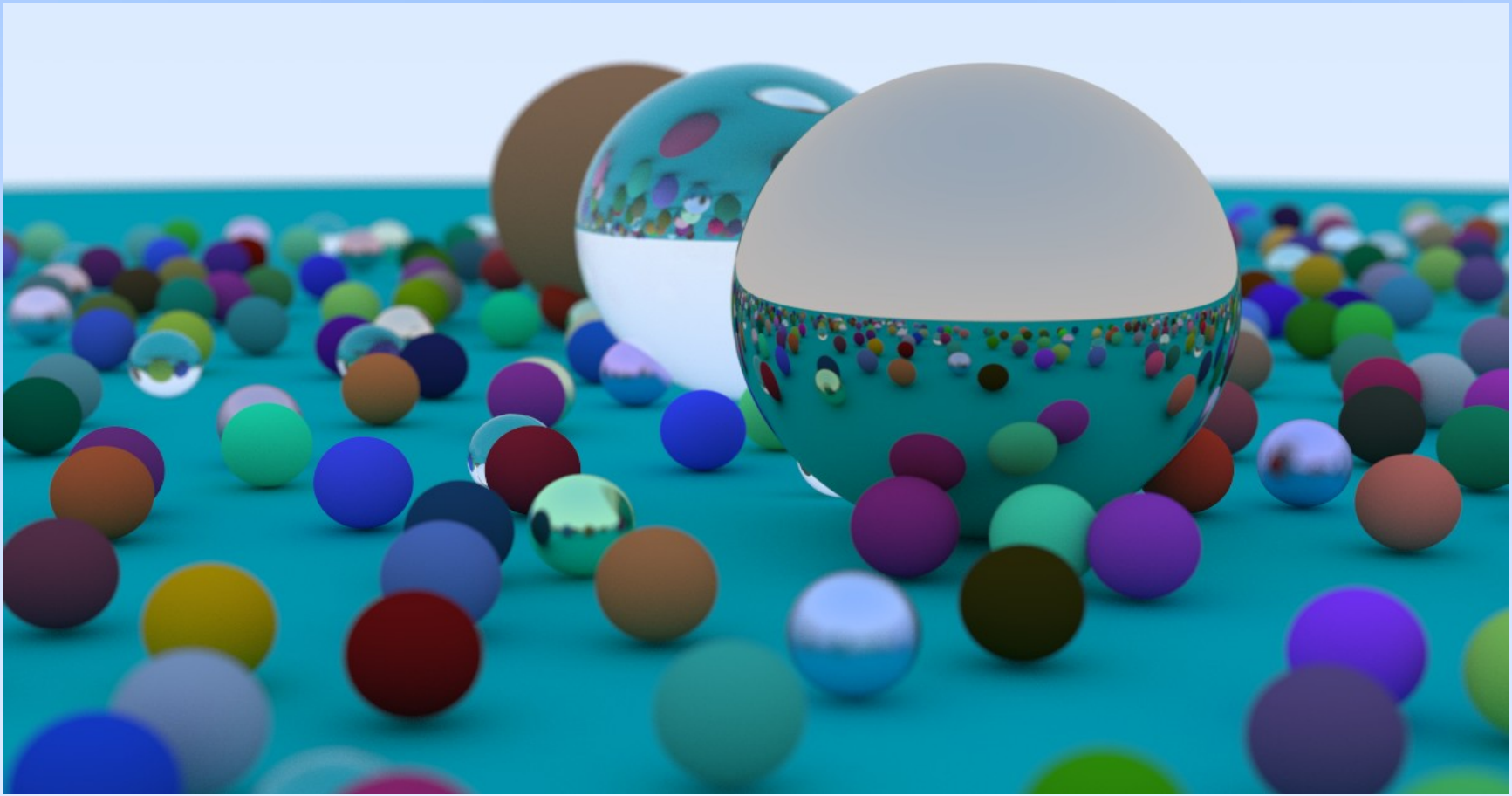
Bounding Volume Hierarchies



Wasting Time - CPU

- Only Using CPU
- Single Threaded
- Use GPU for Parallelization

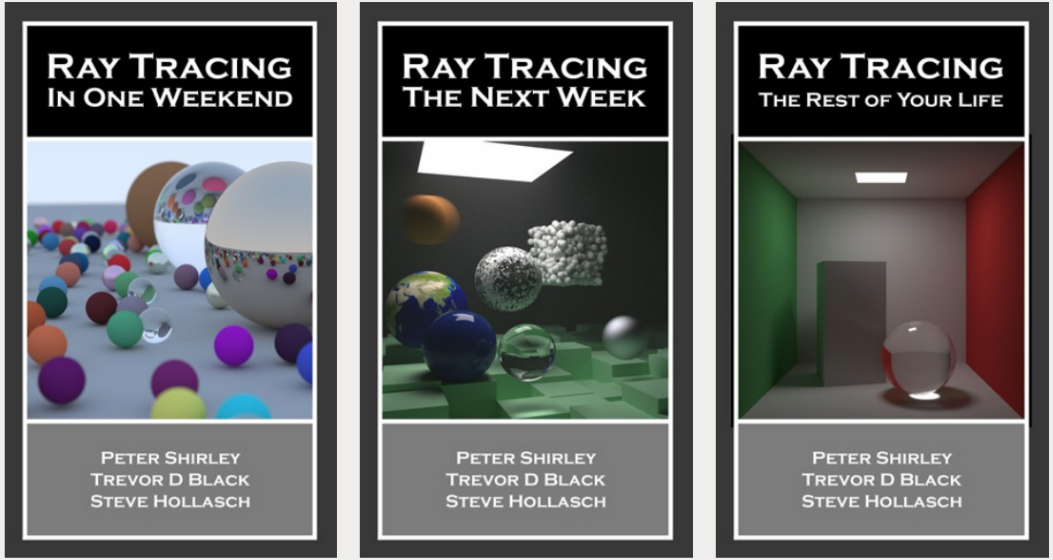
Final Image



Ray Tracing in One Weekend

—

The Book Series



Wasting Time – Stop While You Can

- No “Advanced Computer Graphics”
- Never Perfect
- What About Light Sources?
- Caustics?
- Never Ending
- Simply Use An Existing Ray Tracer

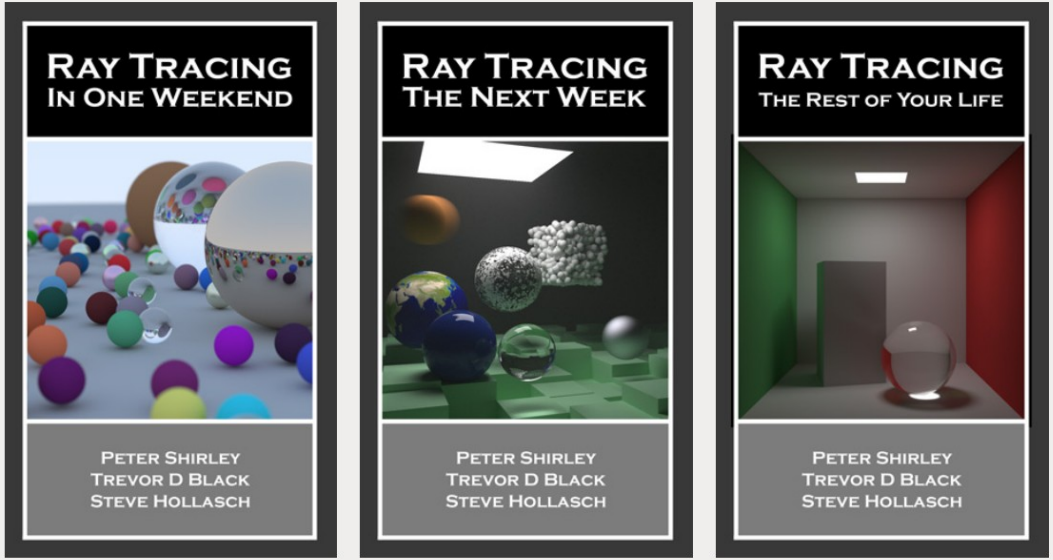
Benefits

- Deepen C++ Knowledge
- Own Makefile Creation
- Pretty Renders
- Playing Around With The Code
- Enhance Ray Tracer Further

Ray Tracing in One Weekend

—

The Book Series



1.5h