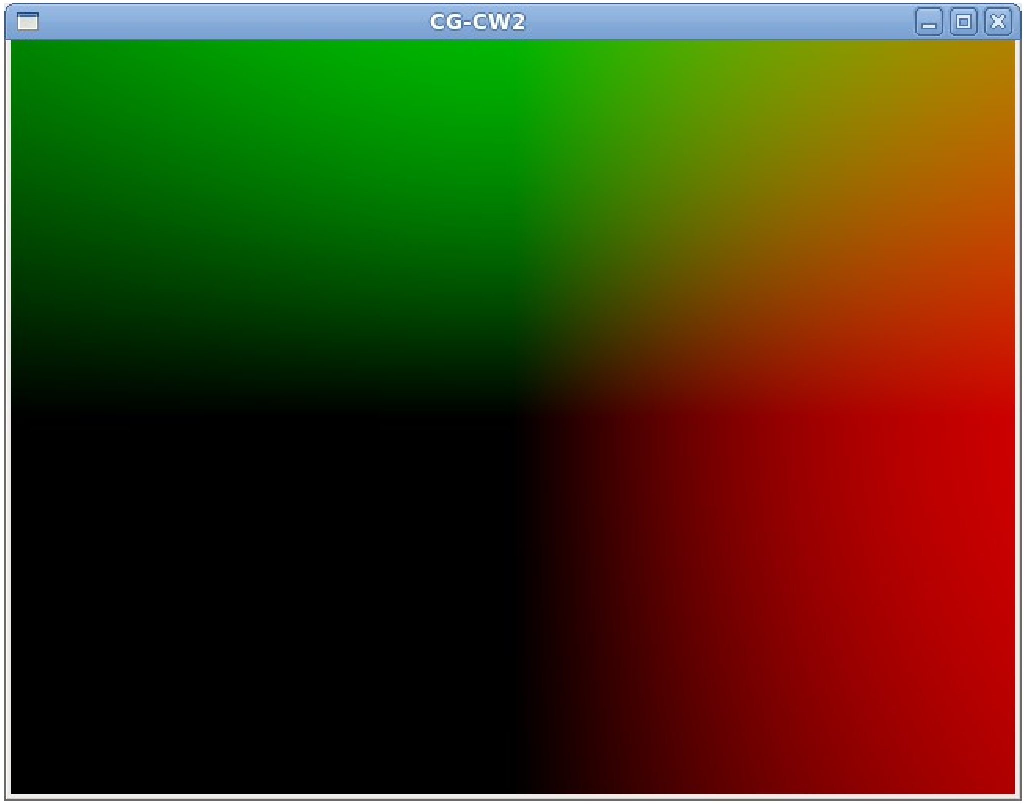
Computer Graphics

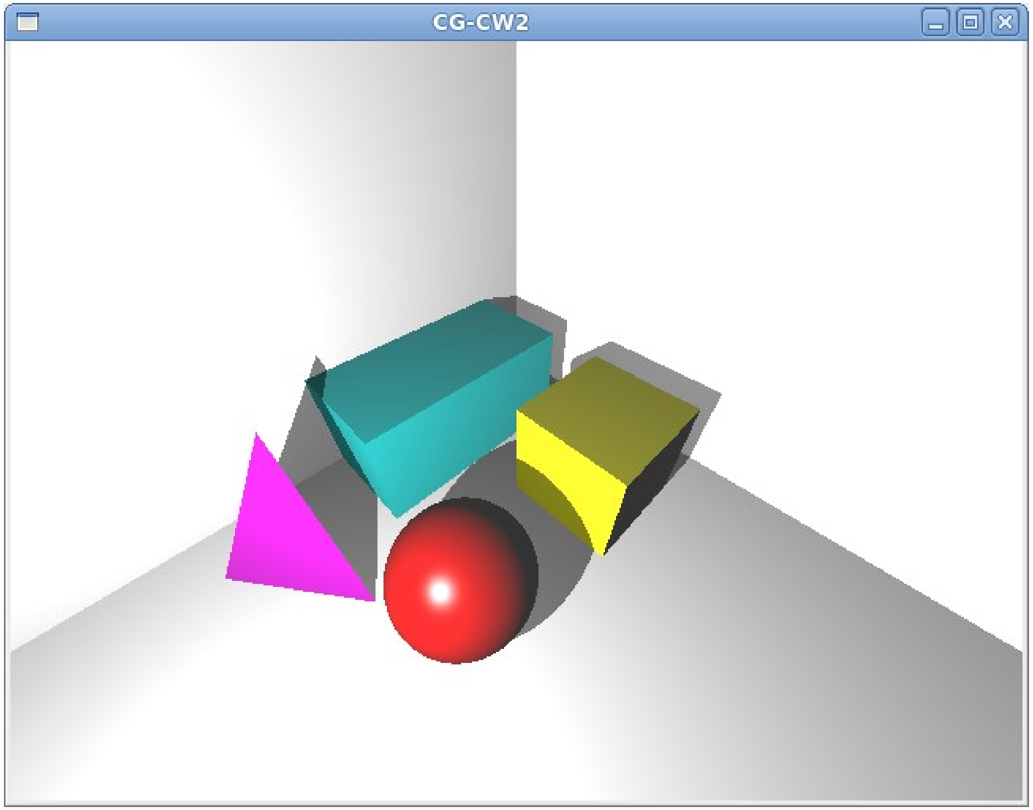
Assignment Two

Objective

- Input

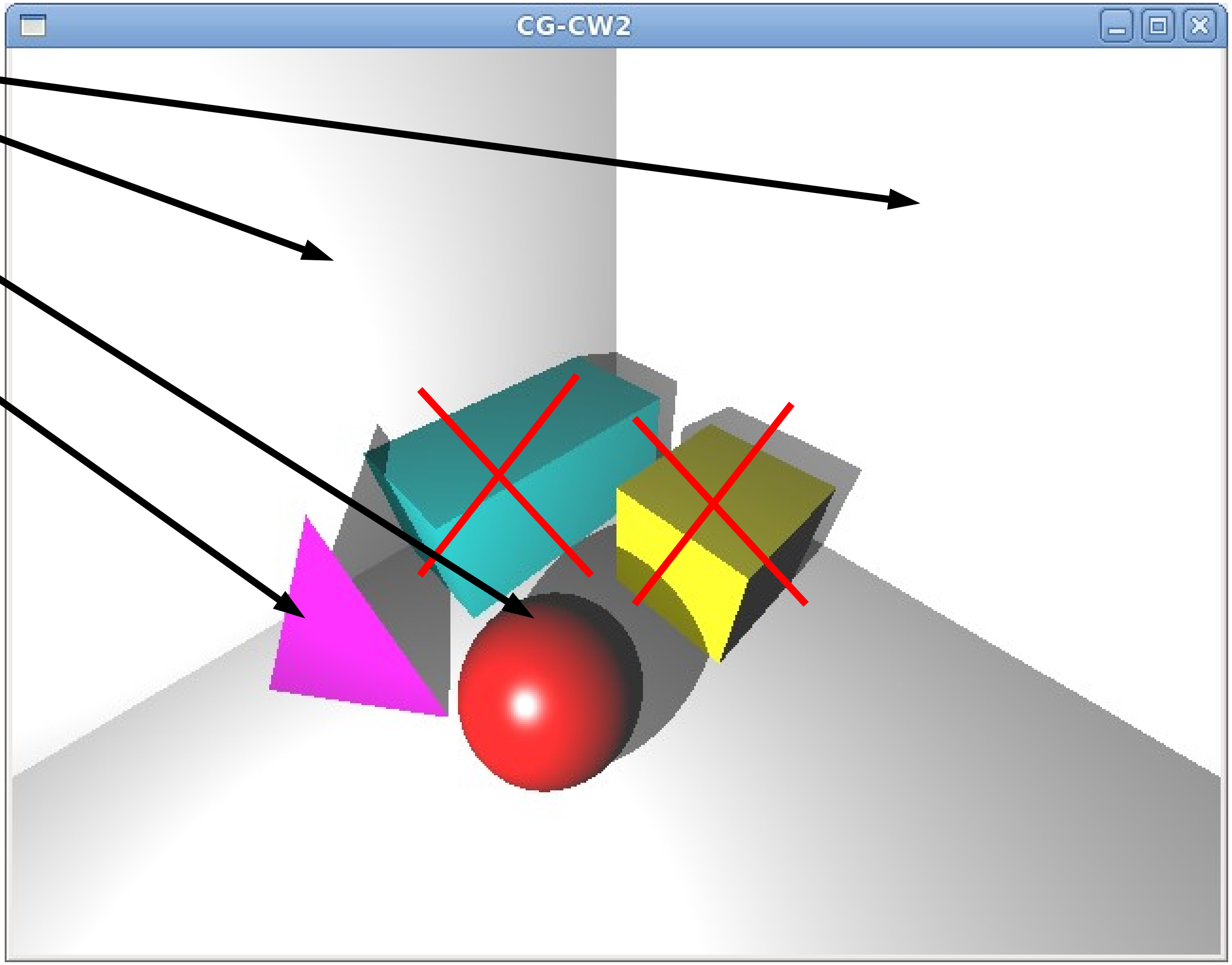


- Output



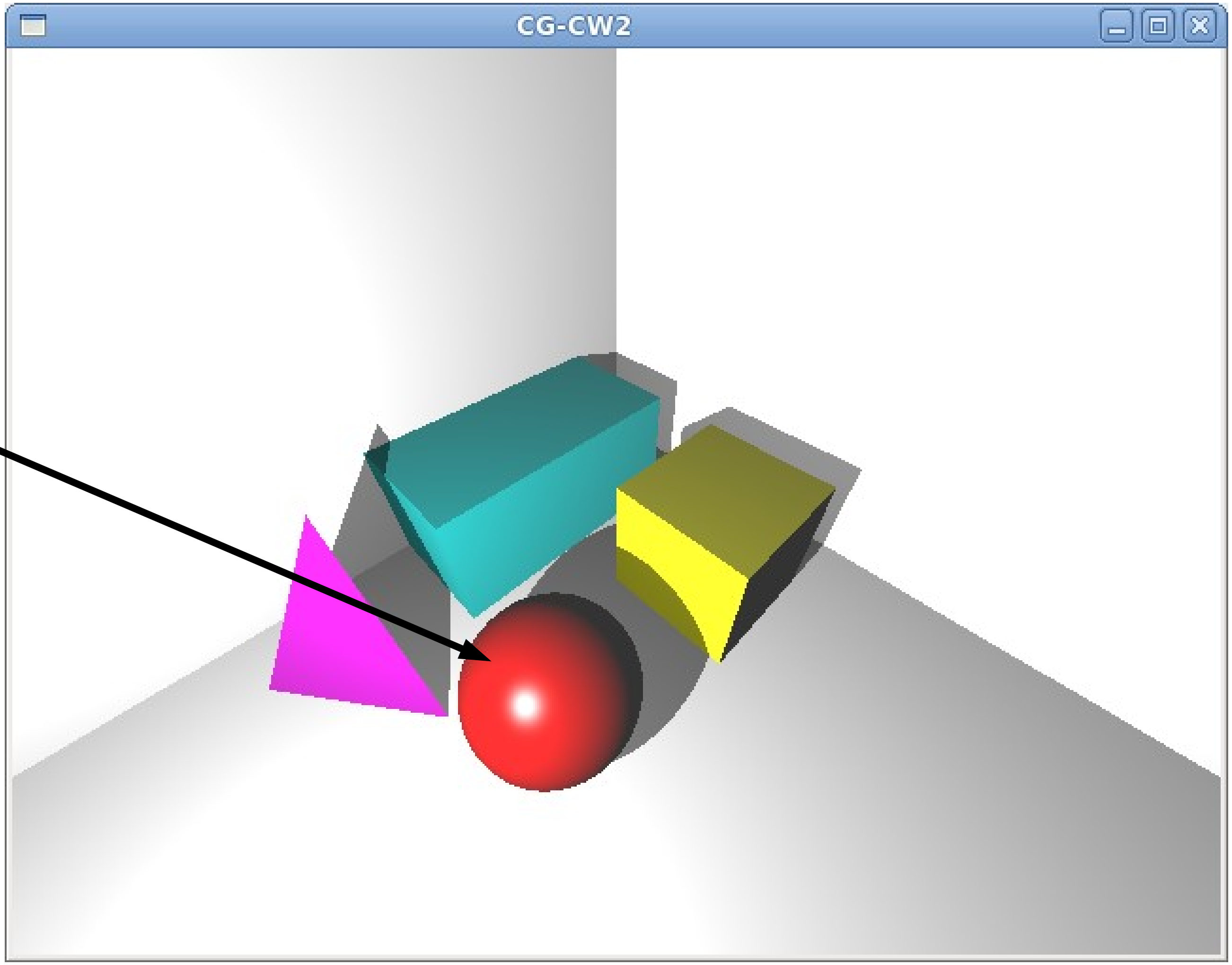
Objective

- 3 Primitive shapes
  - Plane
  - Sphere
  - Triangle



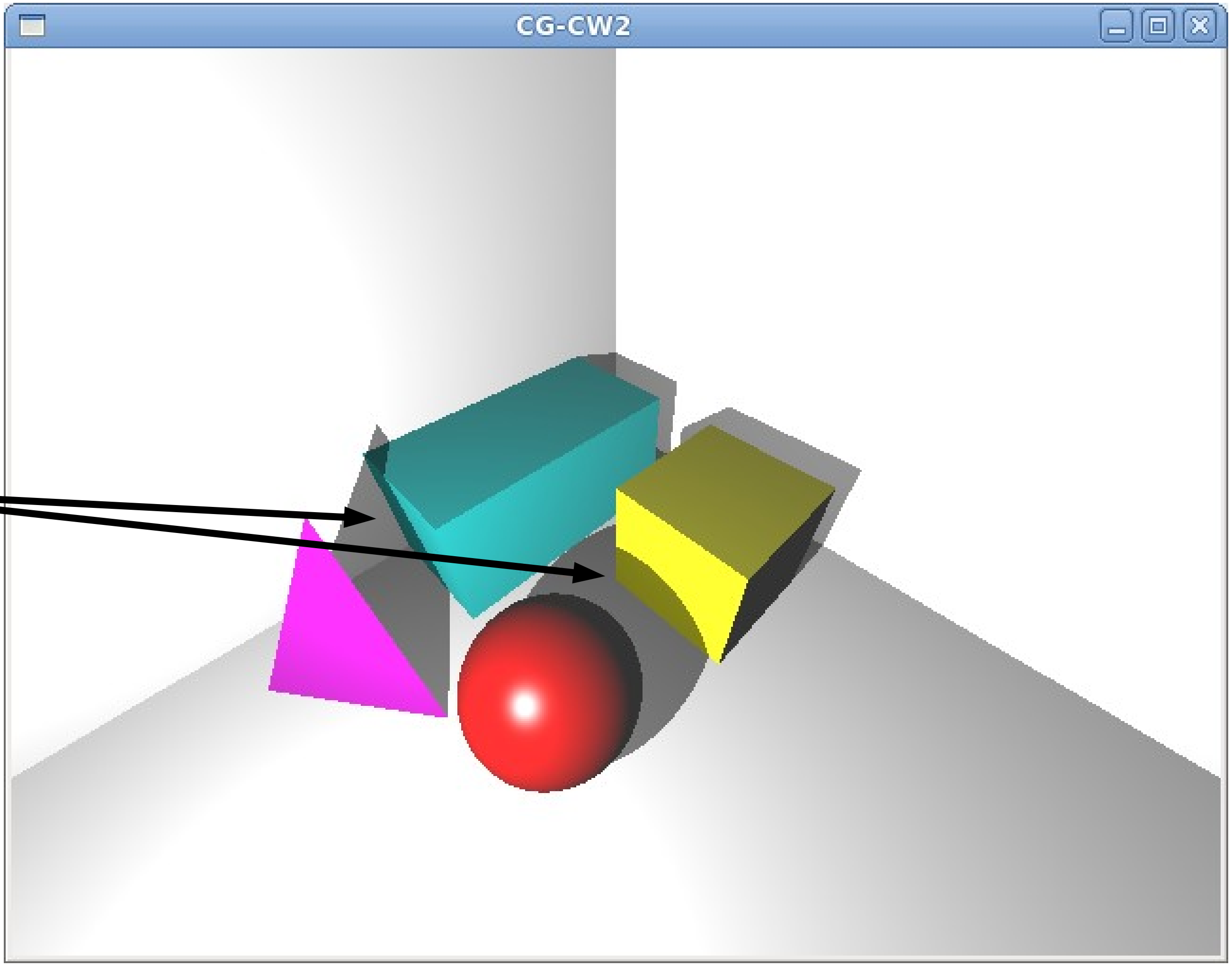
Objective

- 3 Primitive shapes
  - Sphere
  - Plane
  - Triangle
- Lighting



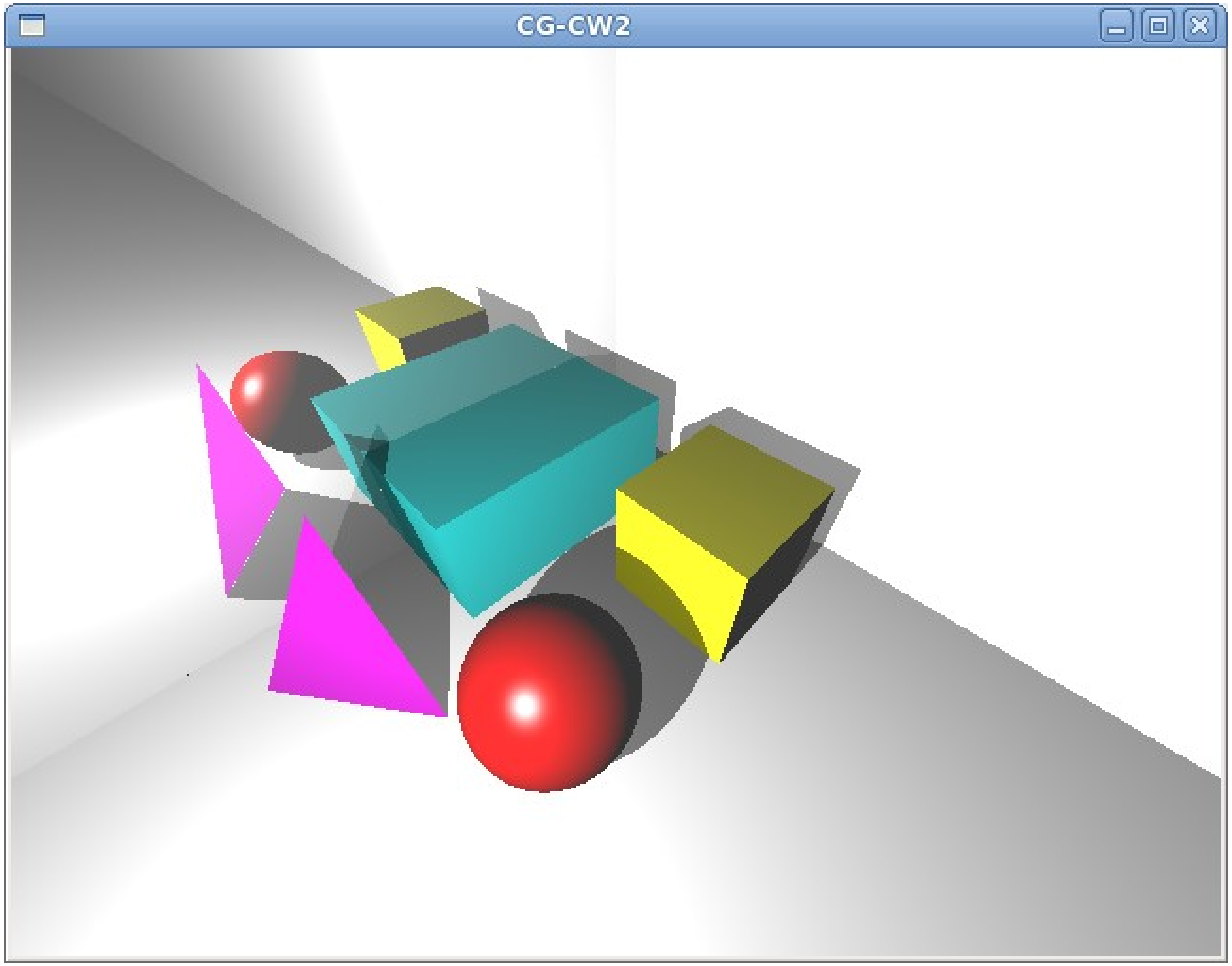
Objective

- 3 Primitive shapes
  - Sphere
  - Plane
  - Triangle
- Lighting
- Shadows

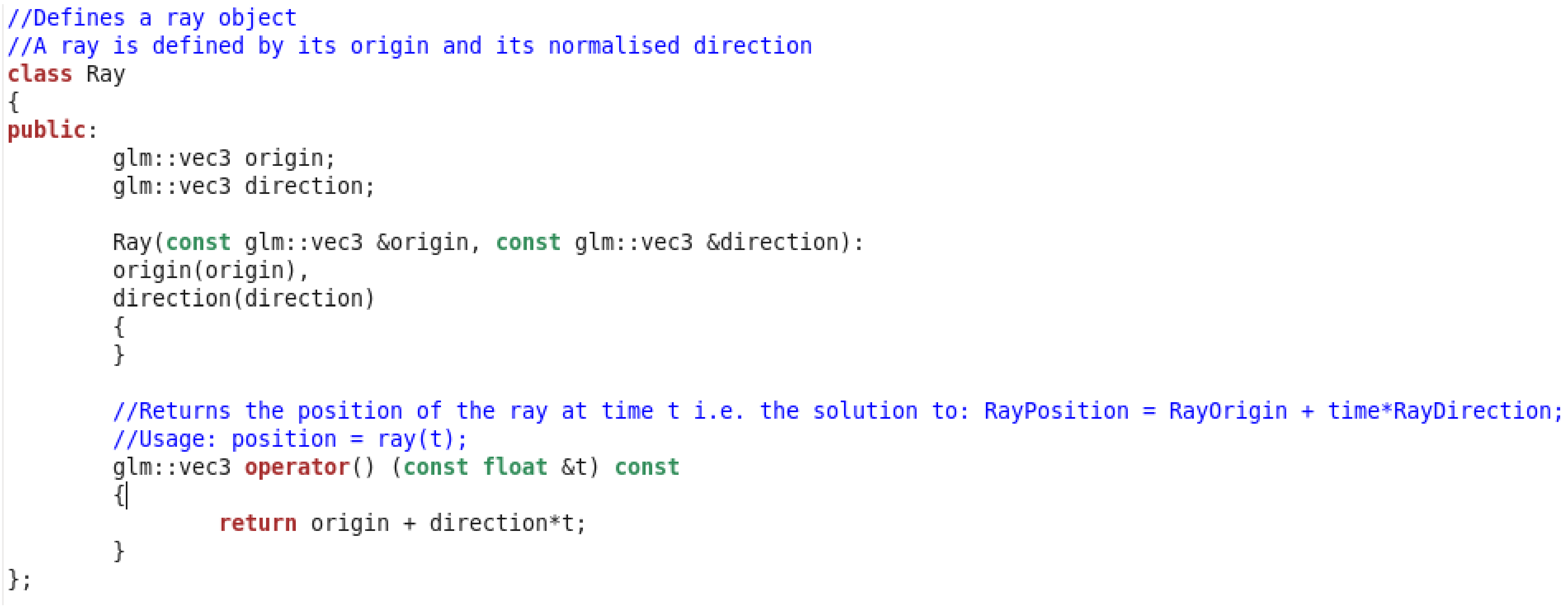


Objective

- 3 Primitive shapes
  - Sphere
  - Plane
  - Triangle
- Lighting
- Shadows
- Reflections



Source Code – The Ray class



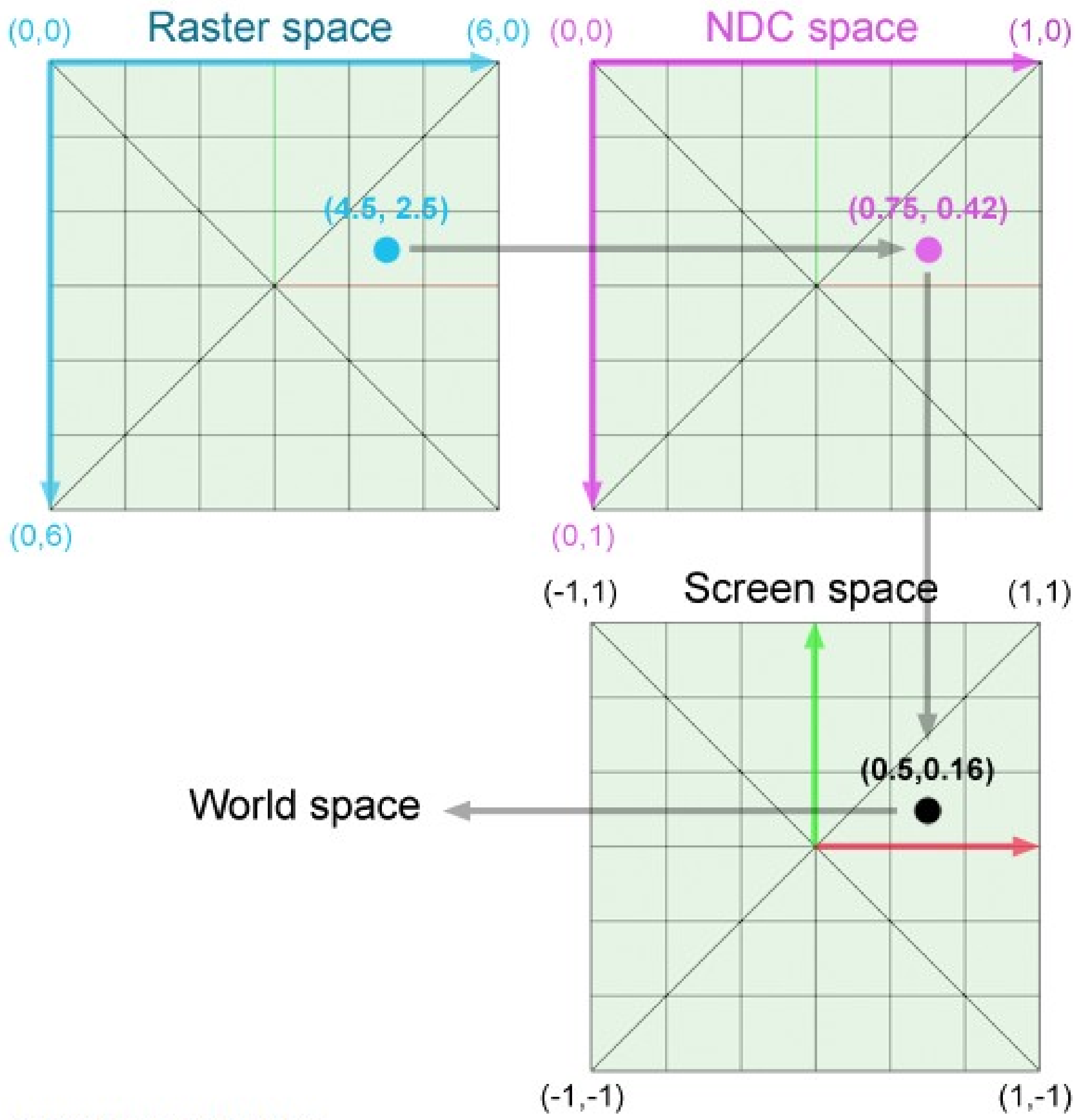
```
//Defines a ray object
//A ray is defined by its origin and its normalised direction
class Ray
{
public:
        glm::vec3 origin;
        glm::vec3 direction;

        Ray(const glm::vec3 &origin, const glm::vec3 &direction):
        origin(origin),
        direction(direction)
        {
        }

        //Returns the position of the ray at time t i.e. the solution to: RayPosition = RayOrigin + time*RayDirection;
        //Usage: position = ray(t);
        glm::vec3 operator() (const float &t) const
        {
                return origin + direction*t;
        }
};
```

Source Code – Setting up the Ray

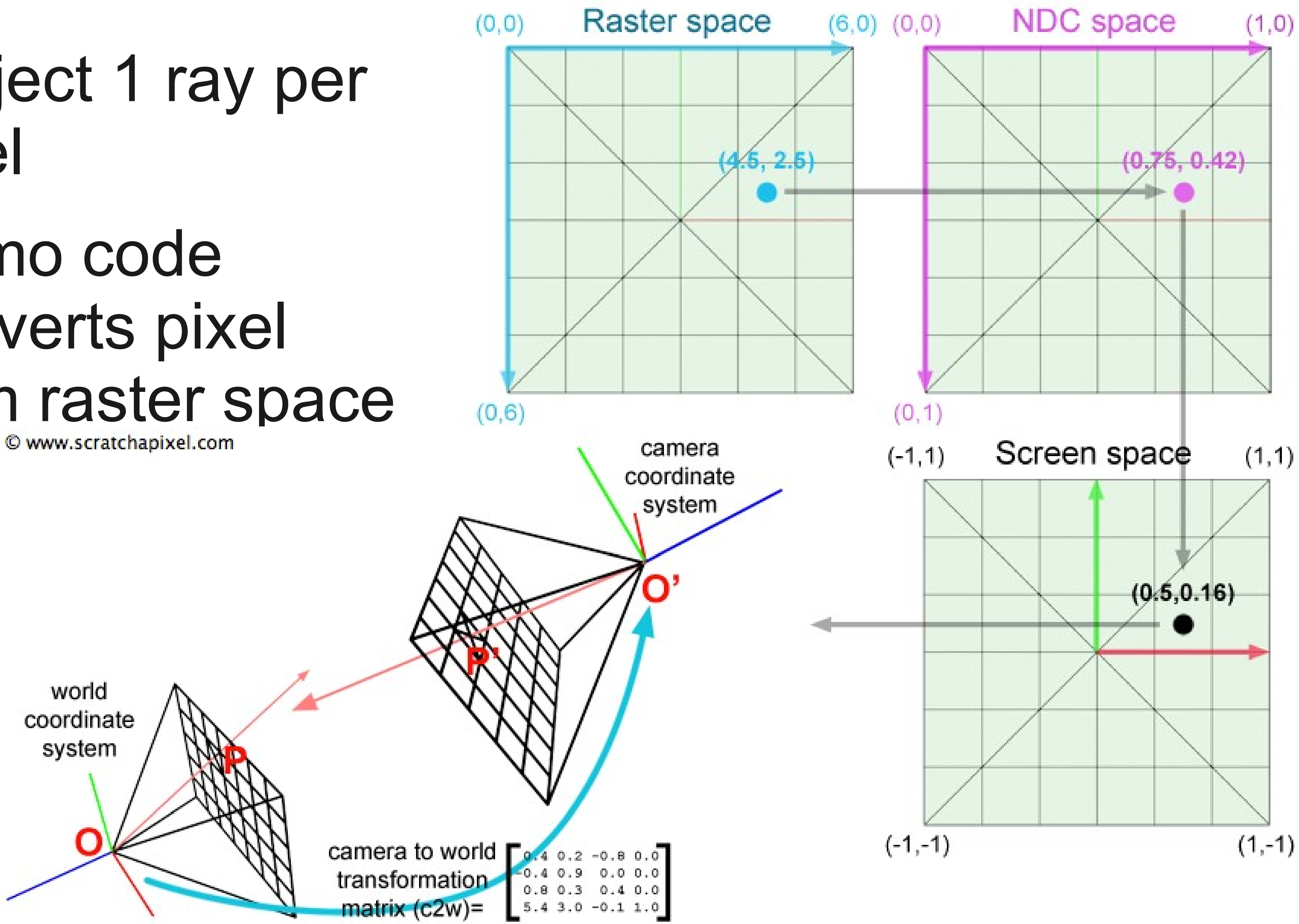
- Project 1 ray per pixel
- Demo code converts pixel from raster space to world space
- Demo code accounts for aspect ratio and field of view



© www.scratchapixel.com

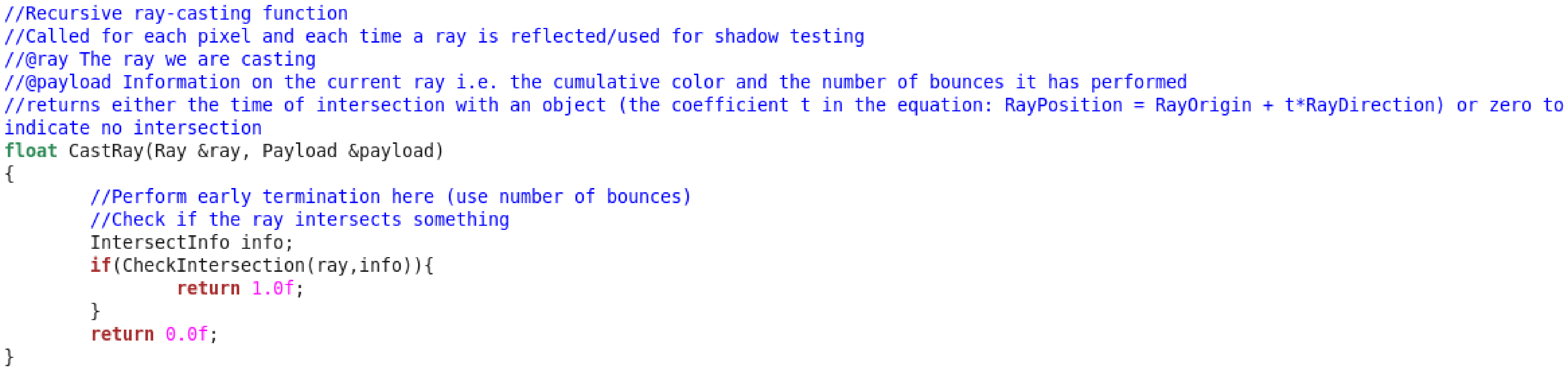
Source Code – Setting up the Ray

- Project 1 ray per pixel
- Demo code converts pixel from raster space to w



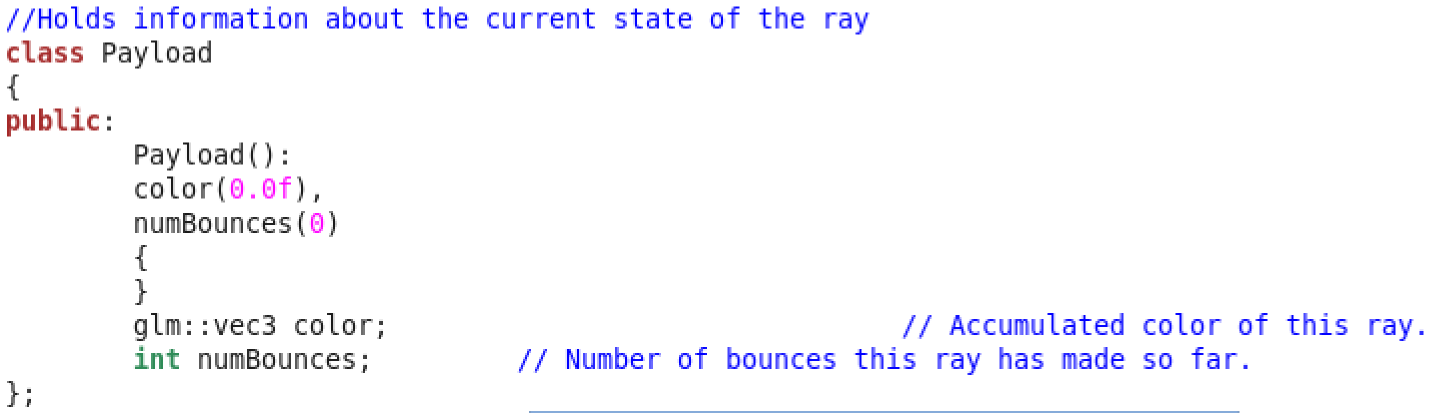
Source Code – Projecting the Ray

- Use “CastRay” as recursive function



```
//Recursive ray-casting function
//Called for each pixel and each time a ray is reflected/used for shadow testing
//@ray The ray we are casting
//@payload Information on the current ray i.e. the cumulative color and the number of bounces it has performed
//returns either the time of intersection with an object (the coefficient t in the equation: RayPosition = RayOrigin + t*RayDirection) or zero to
indicate no intersection
float CastRay(Ray &ray, Payload &payload)
{
        //Perform early termination here (use number of bounces)
        //Check if the ray intersects something
        IntersectInfo info;
        if(CheckIntersection(ray,info)){
                return 1.0f;
        }
        return 0.0f;
}
```

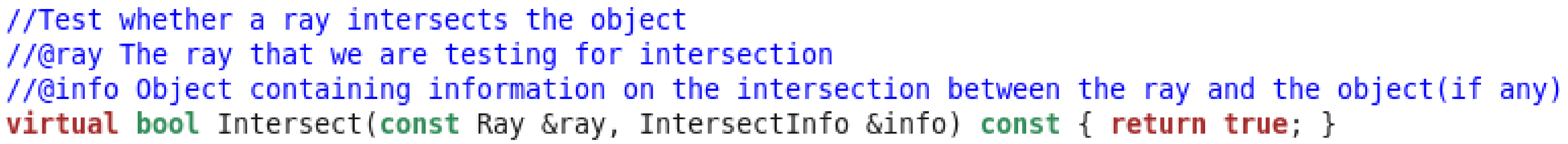
- Use “PayLoad” to record current state of the ray



```
//Holds information about the current state of the ray
class Payload
{
public:
        Payload():
        color(0.0f),
        numBounces(0)
        {
        }
        glm::vec3 color;                                        // Accumulated color of this ray.
        int numBounces;                 // Number of bounces this ray has made so far.
};
```

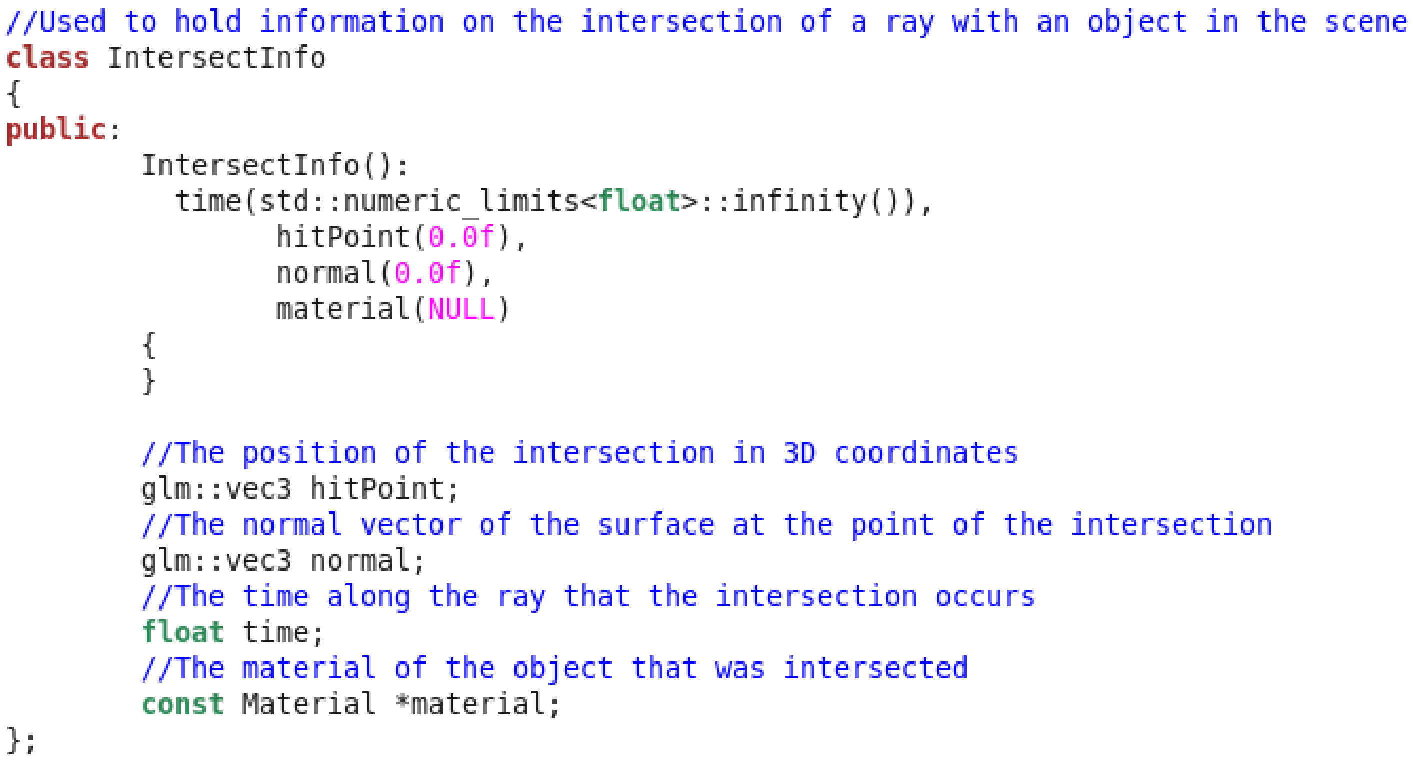
Source Code – Ray-Object intersections

- Extend “Object” class for primitive shapes
  - Override “Intersect” function:



```
//Test whether a ray intersects the object
//@ray The ray that we are testing for intersection
//@info Object containing information on the intersection between the ray and the object(if any)
virtual bool Intersect(const Ray &ray, IntersectInfo &info) const { return true; }
```

  - Use to fill in “IntersectInfo” class
  - Find material properties for the nearest object to the Ray’s origin



```
//Used to hold information on the intersection of a ray with an object in the scene
class IntersectInfo
{
public:
        IntersectInfo():
          time(std::numeric_limits<float>::infinity()),
                hitPoint(0.0f),
                normal(0.0f),
                material(NULL)
        {
        }

        //The position of the intersection in 3D coordinates
        glm::vec3 hitPoint;
        //The normal vector of the surface at the point of the intersection
        glm::vec3 normal;
        //The time along the ray that the intersection occurs
        float time;
        //The material of the object that was intersected
        const Material *material;
};
```

Additional Functions

- Refractions
- Intersection of other primitives
- Acceleration structures e.g. Grid, BVHs
- Soft shadows
- Soft reflections
- Depth of Field
- Subsurface scattering