KTH Stockholm :: EECS :: CST
Introduction to Visualization and Computer Graphics
DH2320/DD2258, Spring 2020
Tino Weinkauf, Hazal Koptagel, Wiebke Köpp
Tom Axblad, Zhenghong Li, Shuyuan Zhang

Homework assignment No. 06

Due February 28, 2020

Task 6.1: Raytracer: Intersections of a Ray with Geometric primitives **16 P**

In this task you will have to complete a Raytracer by computing the intersections between a ray and some geometric primitives. Use the workspace `intersections.inv` for this task. Once you press the *Render* button of the Raytracer, you should see a blue screen together with a setup of the scene. Implement the following functions:

- For the sphere-ray intersection test (8 P)

  ```
  bool Sphere::closestIntersectionModel(const Ray &ray, double maxLambda,
  RayIntersection& intersection) const
  ```

- For the triangle-ray intersection test (8 P)

  ```
  bool Triangle::closestIntersectionModel(const Ray &ray, double maxLambda,
  RayIntersection& intersection) const
  ```

You will find these functions in the files `sphere.cpp` and `triangle.cpp`. If an intersection is found, the function shall return `true`, else it shall return `false`. In case your function returns `true`, both functions should construct objects of type `RayIntersection` and assign them to the parameter `&intersection`. You can refer to the implementation of the Plane-Ray intersection in the file `plane.cpp`.

The constructor of the class `RayIntersection` takes the following parameters :

- `const Ray &ray`: constant reference of the Ray with which an intersection is found.
- `std::shared ptr<const Renderable> renderable`: Reference in the form of a smart pointer to the Object with which the intersection occurred. Use the term `shared_from_this()` here.
- `const real lambda`: Ray parameter where $\lambda \in [0, maxLambda]$.
- `const Vec3d &normal`: constant reference to the normal direction at the point of intersection on the Object's surface.
- `const Vec3d &uvw`: ignore this and assign the Null Vector (0, 0, 0).

After you implement the aforementioned functions, the result should look like the image in Figure 1.

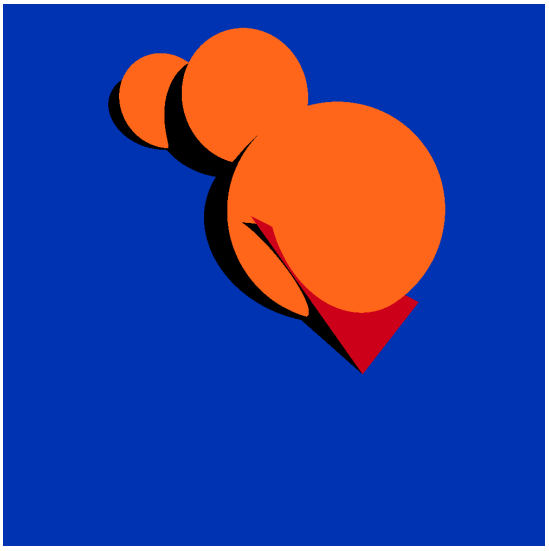

Figure 1: Result of Task 1

Task 6.2: Raytracer: Phong Shading **8 P**

Choose the workspace `illumination.inv` or Task 2 from the drop down. If you correctly implemented the method for the Ray-Sphere intersection you should see an output similar to Figure 2.

The scene for this task consists of 3 point light sources, 4 spheres and a plane. All of these objects are assigned a `PhongMaterial` material. Your task is to implement the Phong shading model.

In the file `phongMaterial.cpp` implement the function

```
Vec4d PhongMaterial::shade(const RayIntersection& intersection,
    const Light& light) const
```

for the reflectance at the point of intersection between a ray and an object. At the moment a simple Lambertian model is implemented with ideal diffuse reflection. You are supposed to implement the Phong model. The return value should be color (red, blue, green, 1). If all goes well, you should see a result similar to Figure 3. Notice the nice specular highlights of the point light sources on the surfaces of the spheres.

Task 6.3: Bonus: Sphere subtraction **8 EP**

Raytracing allows us to perform complex operations on the geometry. Implement a special type of sphere that gets substracted from other volumes in the same raytracing system as before. You are free to implement this however you want in the raytracer. An example image can be seen in Figure 4 - a simplified version without shadows and only rudimentary shading. The scene is called *Bonus Task*, the subtracting spheres are common spheres so far. A few pointers:

- You might want to add this as a property directly to the sphere class or as a new object type in the `sphere.h/.cpp` file. Either has its advantages.
- The actual raytracing is done in `Scene.render()`.
- Feel free to change all the code you need - just make sure the other task scenes still work.

Make sure that you can have several such spheres, and the example looks correct. Phong shading and shadows should work on it.

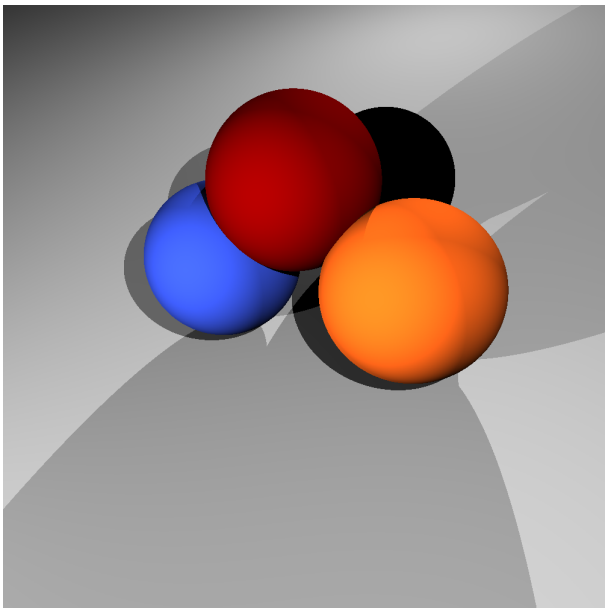

Figure 2: Initial setup for Task 2

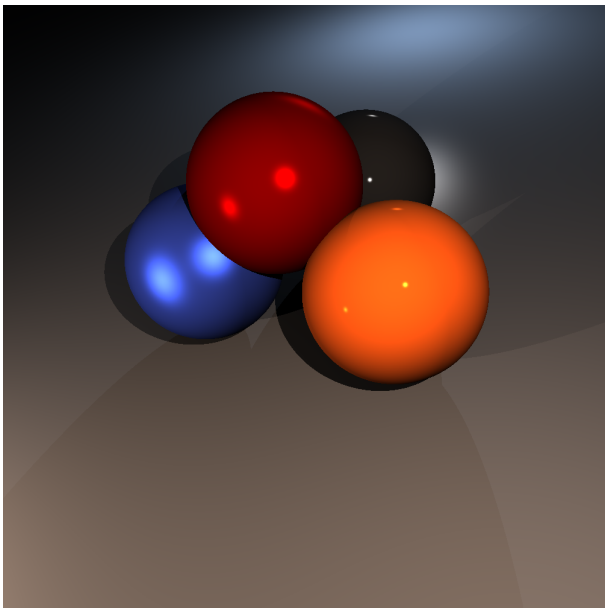

Figure 3: Task 2 result with Phong illumination model

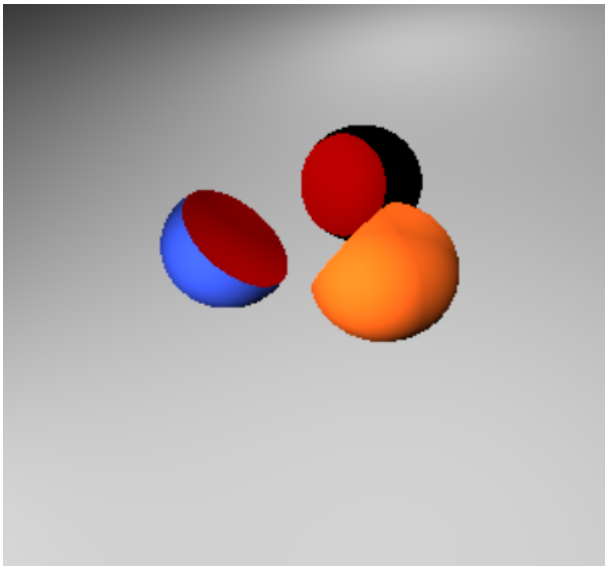

Figure 4: Example for sphere subtraction