Depth of Field & Arbitrarily shaped lights

CS500 – Project 5

Rahil Momin

Contents:

Arbitrary Shaped Lights:

A new class ‘Mesh’ was introduced for arbitrary shaped lights. This class contains a vector of all pointers to triangles related to that mesh.

```
class Mesh : public Shape
{
public:
  std::vector<Triangle*> triangles;
  float area;
};
```

The `Triangle` class now has a `Mesh*` mesh that points to the corelating mesh the triangle belongs to.
Finally, the `triangleMesh(MeshData* meshdata)` function was changed to accommodate these changes. If the mesh is a light, then the `Mesh*` is added to the vector of lights while all the triangles are added to the vector of shapes as always.

The snippet only includes the relevant code changes

```
  Mesh* mesh = new Mesh(meshdata->mat);

  for (size_t i = 0; i < numTriangles; i++)
  {
    Triangle* obj = new Triangle(v0.pnt, v1.pnt, v2.pnt, v0.nrm, v1.nrm, v2.nrm);
    shapes.push_back(obj);
  }

  if (meshdata->mat->isLight())
  {
    lights.push_back(mesh);
  }
```

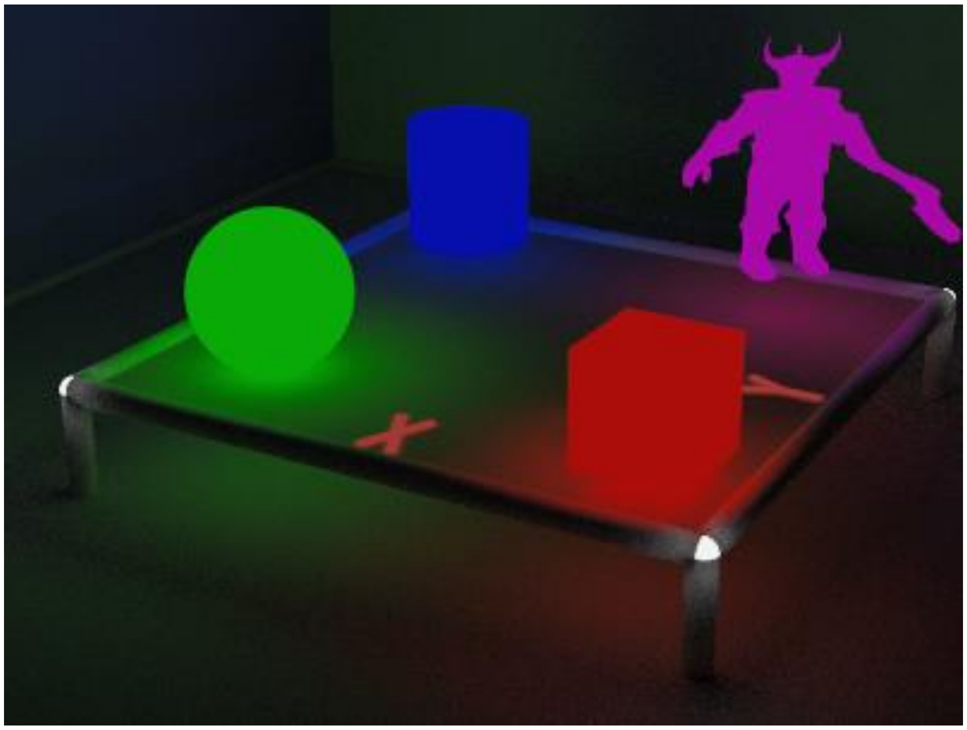
The following functions were modified to handle the changes for arbitrary shapes:

```
  Intersection RayTrace::SampleLight();

  float RayTrace::PdfLight(Intersection& Q);
```

Both functions use Dynamic Cast on the ‘Shape’ to determine the shape of the object, ‘Mesh’ being the special case.

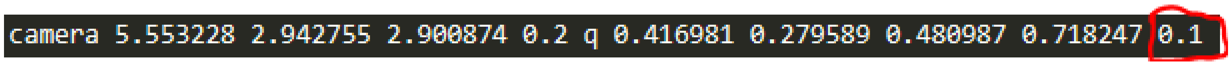
Lights in the scene:
1. ‘Green’ Sphere
2. ‘Blue’ Cylinder
3. ‘Red Box’
4. ‘Purple’ Dwarf
5. ‘Red’ Letter X
6. ‘Red’ Letter Y
7. ‘White’ 4 corners of the table

The Image was produced at **1024 passes**.



Depth of Field:

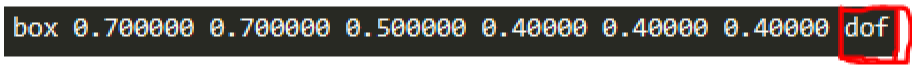
Two new variables were added to the `RayTrace` class, `float D, W;` where ‘D’ is the distance to the plane and ‘W’ is the size of circle of confusion. The scene parser was changed to accommodate these changes. The value of ‘W’ is added to the scene file on the camera line as the last element.



```
camera 5.553228 2.942755 2.900874 0.2 q 0.416981 0.279589 0.480987 0.718247 0.1
```

Which in this case is 0.1.

Since ‘D’ is calculated as distance to the object of focus the scene parser was changed to accept the string “dof” at the end of the object which will be the subject of focus. At which point the position of said object becomes to point of focus.



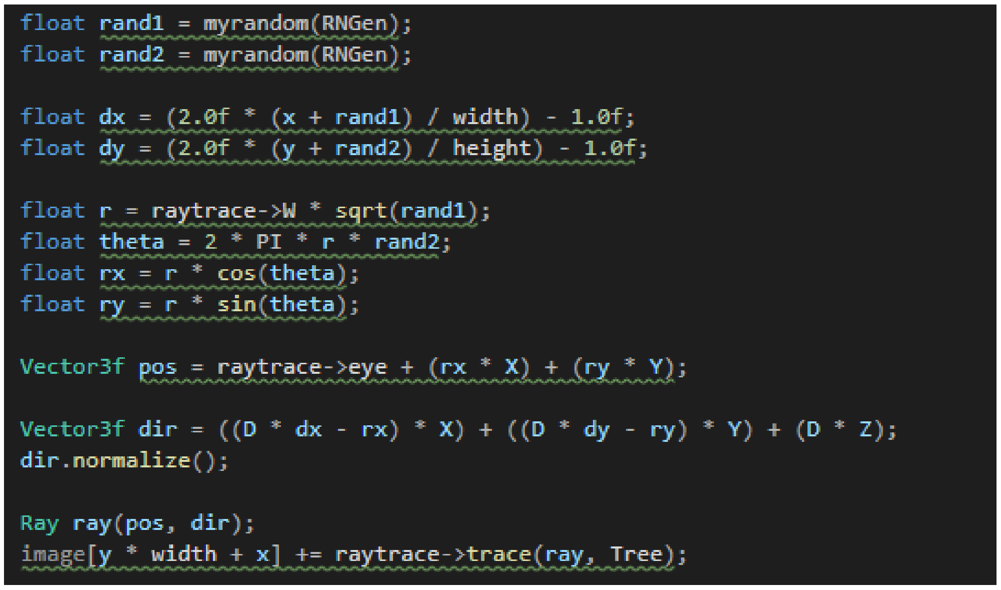
```
box 0.700000 0.700000 0.500000 0.40000 0.40000 0.40000 dof
```

'D' is calculated as

```
 void setDOF(const Vector3f& pos)
{
  D = (pos - eye).norm();
}
```

The Ray-Tracer loop was modified to include depth of field. The ray used to be

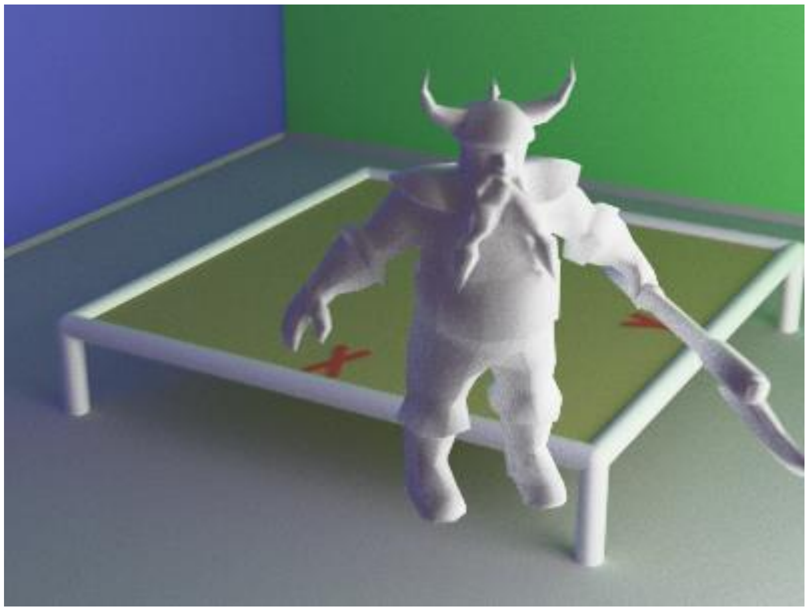
$Ray(Eye, d_x X + d_y Y + Z)$, changed to $Ray\ (Eye + r_x X + r_y Y\ ,\ (D\ d_x - r_x)\ X + (D\ d_y - r_y)\ Y + D\ Z)$.



```
float rand1 = myrandom(RNGen);
float rand2 = myrandom(RNGen);

float dx = (2.0f * (x + rand1) / width) - 1.0f;
float dy = (2.0f * (y + rand2) / height) - 1.0f;

float r = raytrace->W * sqrt(rand1);
float theta = 2 * PI * r * rand2;
float rx = r * cos(theta);
float ry = r * sin(theta);

Vector3f pos = raytrace->eye + (rx * X) + (ry * Y);

Vector3f dir = ((D * dx - rx) * X) + ((D * dy - ry) * Y) + (D * Z);
dir.normalize();

Ray ray(pos, dir);
image[y * width + x] += raytrace->trace(ray, Tree);
```

The resulting effect is as shown on the next page. The Images was produced at 1024 passes.

The Object of focus is the **White Dwarf.**



The point of focus is the **Corner of the room**. (placed a tiny object at the corner, not visible)