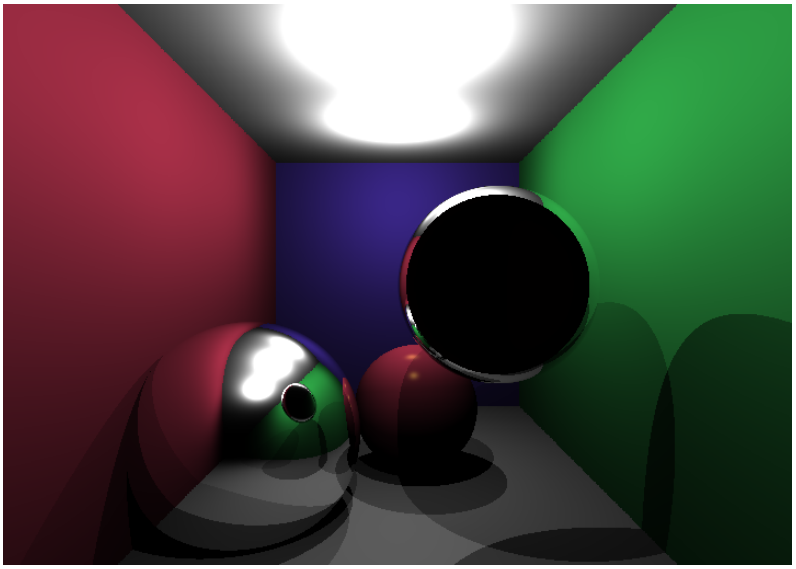
ACG LAB 2 Report
Javier Echavarri Trillo, 205574

1. Mirror and Refraction



2 Materials were created:

1. Mirror: which only has a function that returns the reflection direction and the ones inherited from material are set to false or default except for hasSpecular. In DirectShader, after the intersection loop, we check that the material is Specular and if it is, the direction of the Reflection is calculated and used following the code given in the handout with the recursive calls.

2. Transmissive: transmissive adds a 2 functions. getMu returns mu which is needed to calculated the refraction and to check if the material refracts or reflects. In DirectShader we calculate the determinant and if it is negative then we know that the material reflects at that point (it can be seen in the picture as a halo of the sphere) so it uses the getReflectionDirection and computes the color recursively. If the discriminant is positive then Refractions occurs and the same is done for computing the color but in this case getTransmissionDIrection is used.

```
Vector3D final_color = Vector3D(0.0);
// TODO: if to see if material is mirror
if (its.shape->getMaterial().hasSpecular())
{
    Vector3D wr = its.shape->getMaterial().getReflectionDirection(its.
        normal, -r.d);
    Ray reflectionRay = Ray(its.itsPoint, wr, int(r.depth) + 1);
    final_color = computeColor(reflectionRay, objList, lsList);
    // return final_color;
}

if (its.shape->getMaterial().hasTransmission())
{
    Vector3D w0 = -r.d;
    double cosThetaI = -dot(its.normal, w0);
    double discriminant = 1 - (its.shape->getMaterial().getMu() * its.
        shape->getMaterial().getMu()) * (1 - cosThetaI * cosThetaI);
    if (discriminant < 0)
    {
        Vector3D wr = its.shape->getMaterial().getReflectionDirection
            (its.normal, -r.d);
        // std::cout << "wr: " << wr << std::endl;
        Ray reflectionRay = Ray(its.itsPoint, wr, int(r.depth) + 1);
        final_color = computeColor(reflectionRay, objList, lsList);
    }
    else
    {
        Vector3D wt = its.shape->getMaterial().getTransmissionDirection
            (its.normal, -r.d);
        // std::cout << "wt: " << wt << std::endl;
        // std::cout << "mu: " << its.shape->getMaterial().getMu() <<
            std::endl;
        Ray transmissionRay = Ray(its.itsPoint, wt, int(r.depth) + 1);
        final_color = computeColor(transmissionRay, objList, lsList);
        // std::cout << "final_color_trans: " << final_color <<
            std::endl;
    }
    return final_color;
}
```

```
Vector3D Transmissive::getTransmissionDirection(const Vector3D &n, const
    Vector3D &wo) const
{
    // compute Refractrete
    double cosThetaI = -do
    double muT = mu;
    Vector3D v1 = n * sqrt
    Vector3D v2 = (wo - n * cosThetaI);
    Vector3D v3 = v2.operator*(muT);
    Vector3D wt = v3 - v1;

    return wt;
}

Vector3D Transmissive::getReflectionDirection(const Vector3D &n, const
    Vector3D &wo) const
{
    Vector3D wr = n * 2 * dot(n, wo) - wo;
    return wr;
}

float Transmissive::getMu() const
{
    return mu;
}
```

```
Vector3D Vector3D::operator*(double a) const

1 sobrecarga más

Multiply a vector by a scalar and return the result as a new object
```

2. Global Illumination

1. Introduction

- GlobalShader is created, copying DirectShader

- getDiffuseCoefficient is created in Phong material, the implementation is done in the .h file.

- computeColor is modified in globalShader so that after calculating the direct light in the illumination for loop the indirect light is added only if the material is Phong (indicated in the handout)

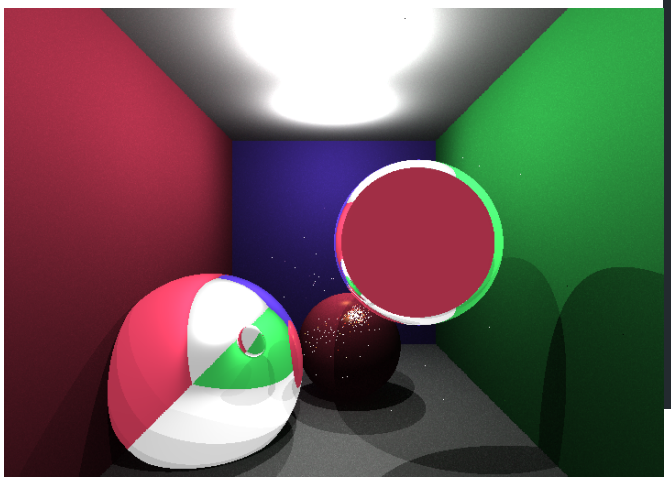
```
if (its.shape->getMaterial().hasDiffuseOrGlossy())
{
    Vector3D at = Vector3D(0.2, 0.5, 0.5);
    Vector3D kd = its.shape->getMaterial().getDiffuseCoefficient();
    final_color += at * kd;
}

return final_color;
```

2. Task 1

The Code is modified following the handout. When depth of the rays is 0 we send nSamples using the hemispherical samplers this will output n light directions. The a ray is cast for every direction and the color is computed. Then the reflectance is obtained and lastly the indirect light is calculated. If the depth is more than 0 the approach of the previous exercise is done.

```
if (its.shape->getMaterial().hasDiffuseOrGlossy())
{
    if (r.depth == 0)
    {
        Vector3D Lind = Vector3D(0.0);
        int nSamples = 16;
        HemisphericalSampler sampler;
        for (int j = 0; j < nSamples; j++)
        {
            Vector3D wi = sampler.getSample(its.normal);
            Ray ray = Ray(its.itsPoint, wi, r.depth + 1);
            Vector3D Li = computeColor(ray, objList, lsList);
            Vector3D reflectance = its.shape->getMaterial().
                getReflectance(its.normal, -r.d, wi);
            Lind += Li * reflectance;
        }
        Lind *= (1 / (2 * M_PI * nSamples));
        final_color += Lind;
    }
    else
    {
        Vector3D at = Vector3D(0.9, 0.9, 0.9);
        Vector3D kd = its.shape->getMaterial().getDiffuseCoefficient();
        final_color += at * kd;
    }
}
```



3. Task 2

I was not able to do the this task, I encountered a segmentation error and I could not fix it