Week 11 Report

Jakub Dutkowski, s121937
Alexander Birke, s124044

December 16, 2012

1 INTRODUCTION

The point of this week's set of exercises is to introduce a physically based reflectance and absorbtion models to the raytracing framework. This is done using Fresnel equations and Bouguer's law.

2 FRESNEL REFLECTANCE

First part of the assignment focuses on calculating reflectance depending on angle.

Listing 1: the fresnel equations implemented in `fresnel.h` file

```
inline double fresnel_r_s(double cos_theta1, double cos_theta2,
    double ior1, double ior2)
{
  // Compute the perpendicularly polarized component of the
      Fresnel reflectance
  return (ior1*cos_theta1 - ior2*cos_theta2) / (ior1*cos_theta1 +
       ior2*cos_theta2);
}

inline double fresnel_r_p(double cos_theta1, double cos_theta2,
    double ior1, double ior2)
{
  // Compute the parallelly polarized component of the Fresnel
      reflectance
```

```
    return (ior2*cos_theta1 - ior1*cos_theta2) / (ior2*cos_theta1 +
        ior1*cos_theta2);
}

inline double fresnel_R(double cos_theta1, double cos_theta2,
    double ior1, double ior2)
{
    // Compute the Fresnel reflectance using fresnel_r_s(...) and
        fresnel_r_p(...)
    return 0.5 * ( pow( fresnel_r_s(cos_theta1, cos_theta2, ior1,
        ior2), 2) + pow( fresnel_r_p(cos_theta1, cos_theta2, ior1,
        ior2), 2) );
}
```

Then the `Fresnel_R` function is used to compute reflectance as shown in listing below. In case of a total internal reflection (when `optix::refract` returns false), reflectance is set to 1.

Listing 2: `trace_refracted` function from `RayTracer.cpp` file

```
bool RayTracer::trace_refracted(const Ray& in, const HitInfo&
    in_hit, Ray& out, HitInfo& out_hit, float& R) const
{
    float3 outDirection;
    float3 outNormal;
    float inTheta;

    out_hit.ray_ior = get_ior_out(in, in_hit, outDirection,
        outNormal, inTheta);

    float3 refractedDirection;

    if(optix::refract(refractedDirection, -outDirection,
        outNormal, out_hit.ray_ior / in_hit.ray_ior))
    {
        out.ray_type = 0;
        out.direction = refractedDirection;
        out.origin = in_hit.position;
        out.tmin = 0.001f;
        out.tmax = 99999;

        double cos_theta1 = inTheta;
        double cos_theta2 = optix::dot( refractedDirection, -
            outNormal);
        R = fresnel_R(cos_theta1, cos_theta2, in_hit.ray_ior,
            out_hit.ray_ior);
        if(this->trace_to_closest(out, out_hit))
        {
            out_hit.trace_depth = in_hit.trace_depth + 1;
```

```
                return true;
            }
        }
        else
            R = 1.0f;

        return false;
}
```

3 Absorption

The second part of the assignment focuses on handling absorbtion of light in a medium. This is done by using Bouguer's law.

The function listed below calculates transmittance from diffuse reflectance using Bouguer's law. The conditions comparing against small values are introduced to handle division by zero.

Listing 3: `get_transmittance` function from `Volume.cpp`

```
float3 Volume::get_transmittance(const HitInfo& hit) const
{
  if(hit.material)
  {
    // Compute and return the transmittance using the diffuse
       reflectance of the material.
    // Diffuse reflectance rho_d does not make sense for a
       specular material, so we can use
    // this material property as an absorption coefficient. Since
        absorption has an effect
    // opposite that of reflection, using 1/rho_d-1 makes it more
        intuitive for the user.
    float3 rho_d = make_float3(hit.material->diffuse[0], hit.
       material->diffuse[1], hit.material->diffuse[2]);

     float3 at;

     if(rho_d.x < 0.000001f)
          at.x = 99999999.0f;
     else
          at.x = (1.0f / rho_d.x) - 1;

     if(rho_d.y < 0.000001f)
          at.y = 99999999.0f;
     else
          at.y = (1.0f / rho_d.y) - 1;

     if(rho_d.z < 0.000001f)
```

```
            at.z = 99999999.0f;
        else
            at.z = (1.0f / rho_d.z) - 1;

        return optix::expf(-at*hit.dist);
    }

    return make_float3(1.0f);
}
```

Listing 4: `shade` function from`Volume.cpp`

```
float3 Volume::shade(const Ray& r, HitInfo& hit, bool emit) const
{
    // If inside the volume, Find the direct transmission through
        the volume by using
    // the transmittance to modify the result from the Transparent
        shader.

        if(dot(r.direction, hit.shading_normal) > 0)
          return Transparent::shade(r, hit, emit) *
              get_transmittance(hit);
        else
            return Transparent::shade(r, hit, emit);
}
```

After changing the parameters of the glass ball in `default_scene.mtl` file (`illum` set to 12 and `Kd` to `(0.8, 0.1, 0.3)`) the function listed below is implemented.

Listing 5: `shade` function from `GlossyVolume.cpp`

```
float3 GlossyVolume::shade(const Ray& r, HitInfo& hit, bool emit)
     const
{
    // Compute the specular part of the glossy shader and attenuate
        it
    // by the transmittance of the material if the ray is inside (
        as in
    // the volume shader).

        float3 transmittance = make_float3(1);

        if(dot(r.direction, hit.shading_normal) > 0)
        {
            transmittance = get_transmittance(hit);
        }

        float3 rho_s = get_specular(hit);
```

```
    float s = get_shininess(hit);

        float3 result = make_float3(0);

        for(int i = 0 ; i < lights.size() ; i++)
        {
            float3 L;
            float3 dir;
            if( lights[i]->sample(hit.position, dir, L) )
            {
                float3 R = optix::reflect(-dir,hit.
                   shading_normal);
                unsigned int sInt = (int)s;

                result += ( rho_s*L* int_pow(optix::dot(R, -r
                   .direction), sInt));
            }
        }

        return Volume::shade(r, hit, emit) + result *
           transmittance;
}
```

Next step is to copy `get_transmittance` function from Listing 3 to the function of the same name in `ParticleTracer.cpp`.

Listing 6: `trace_particle` function from `ParticleTracer.cpp`

```
void ParticleTracer::trace_particle(const Light* light, const
    unsigned int caustics_done)
{
  if(caustics_done)
    return;

  // Shoot a particle from the sampled source
  Ray r;
  HitInfo hit;
  float3 phi;

  light -> emit(r, hit, phi);

  // Forward from all specular surfaces
  while(scene -> is_specular(hit.material) && hit.trace_depth <
     500)
  {
    switch(hit.material->illum)
    {

     ...

    case 12: // absorbing glossy volume
      {
            if(dot(r.direction, hit.shading_normal) > 0)
            {
                  float3 transmittance = get_transmittance(hit);
                  float P = (transmittance.x + transmittance.y +
                     transmittance.z ) / 3;

                  if (mt_random() < P)
                  {
                     phi *= transmittance / P;
                  }
                  else
                      return;
            }
      }

     ...

  }
  if (hit.trace_depth > 0)
     caustics.store(phi, hit.position, -r.direction);
}
```

Implementing glossy absorption shown in listing 6 is the last part of this assignment. Here it is first checked if we are are inside the object before the absorption is calculated. This is done by taking the dot product between the ray's direction and the surface normal.

4 Results

The renderings below show the result of implementing absorption and Fresnel reflectance together with a render of non Fresnel reflectance.

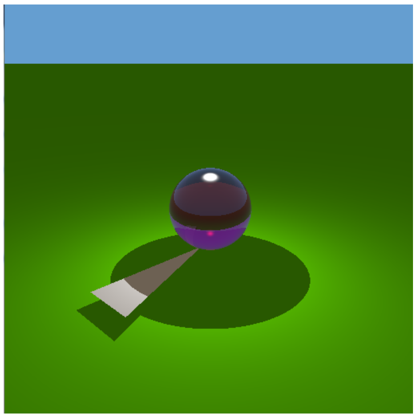
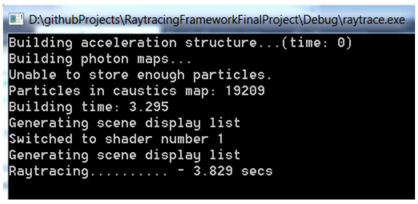


Figure 4.1: Fresnel reflectance and absorption without caustics



```
D:\githubProjects\RaytracingFrameworkFinalProject\Debug\raytrace.exe
Building acceleration structure...(time: 0)
Building photon maps...
Unable to store enough particles.
Particles in caustics map: 19209
Building time: 3.295
Generating scene display list
Switched to shader number 1
Generating scene display list
Raytracing.......... - 3.829 secs
```

Figure 4.2: Log for fresnel reflectance without caustics

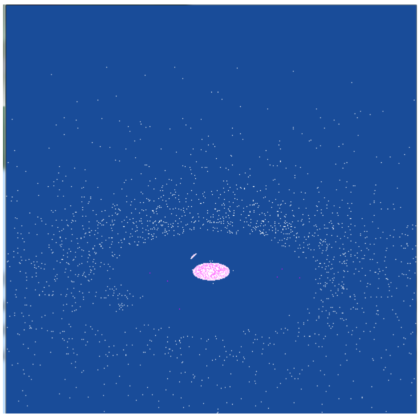


Figure 4.3: Photon map with fresnel reflectance

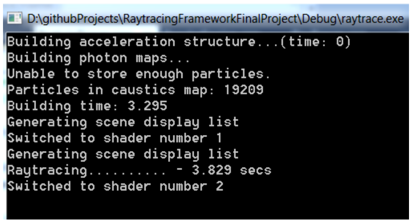


Figure 4.4: Log for fresnel reflectance and aborption without caustics

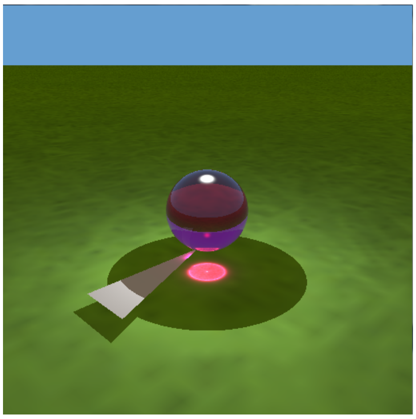


Figure 4.5: Final render result

5 Comparison to old glossy shader

If we compare the final rendering result to a render of the glossy shader without fresnel reflectance we can see that the reflection of the sky is shaped a bit differently.

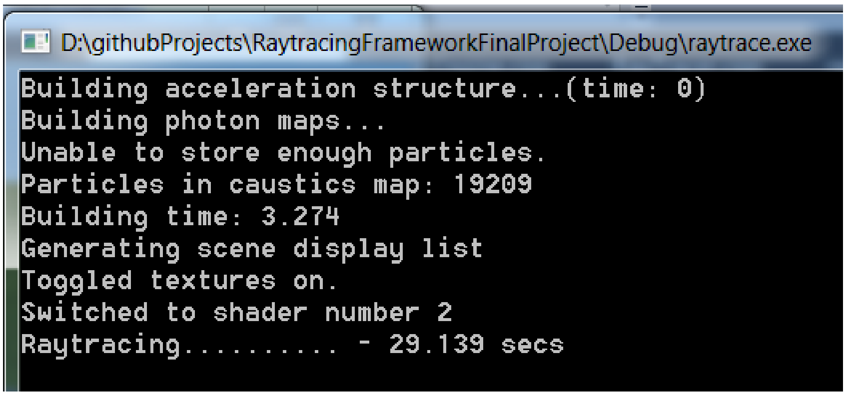
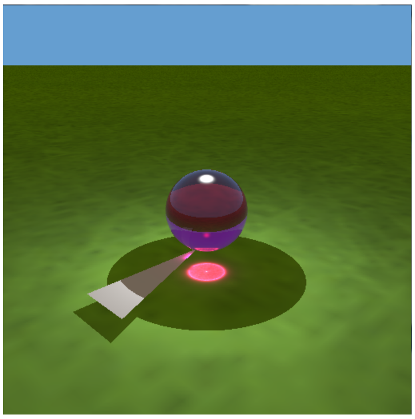
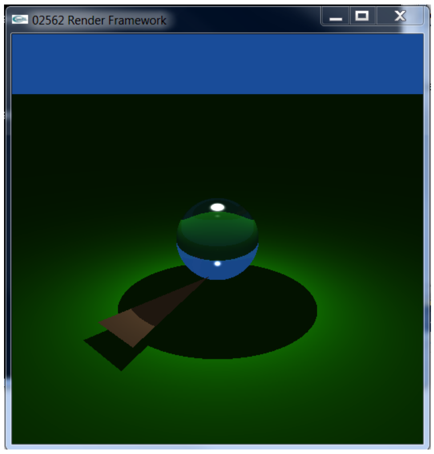


Figure 4.6: Log for final render result



(a) Absorption and Fresnel reflectance



(b) old glossy shader