Weeks 8-9 Report

Jakub Dutkowski, s121937
Alexander Birke, s124044

December 16, 2012

1 INTRODUCTION

The purpose of this set of exercises is to add photon mapping to the previously implemented ray tracing solution. As a result, caustics should be included in the renderings.

2 EMITTING PHOTONS FROM LIGHT SOURCE

In order to emit photons we first create a random direction vector that is inside the unit sphere. This is done through rejection sampling. Then we cast a ray in that direction with origin at the light's position. If it hits something we calculate the irradiance the photon. Since we have the intensity of the light which unit is W/sr we just have to multiply the intensity with 4 pi in order to get the radiant flux:

Listing 1: emit function from PointLight.cpp file

```
bool PointLight::emit(Ray& r, HitInfo& hit, float3& Phi) const
{
    // Sample ray direction with rejection sampling
    // and create ray
    float x,y,z;
    do
    {
        x = randomizer.mt_random() * 2 - 1;
        y = randomizer.mt_random() * 2 - 1;
        z = randomizer.mt_random() * 2 - 1;
    }while(x*x + y*y + z*z > 1);
```

```
    float3 dir = normalize(make_float3(x, y, z));

    r.direction = dir;
    r.origin = light_pos;
    r.tmin = 1e-5;
    r.tmax = 9999;

    // Trace created ray
    // If a surface was hit, compute Phi and return true
    if (tracer -> trace_to_closest(r, hit))
    {
        Phi = intensity * 4 * M_PIf;
        return true;
    }

    return false;
}
```

3 TRACING AND STORING PHOTONS

The next step is to emit photons and storing them. First we check if we have enough photons, if not we continue with the function. The first step is to declare the necessary variables to shooting a ray and storing the radiant flux of the photon it casts. We then emit it and bounce the photon around until it reaches a difuse surface. If it hits a trnasparent surface we first trace the refracted ray. From that we get the reflectance of the surface and use it to calculate with russian roulette if we should trace the reflected ray instead. After the while loop we check if the ray has been traced more than once which means the created photon has been produced by a caustic. If that is the case it is stored in the caustics photon map.

Listing 2: trace_particle function from ParticleTracer.cpp file

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

  Ray r;
  HitInfo hit;
  float3 phi;

  // Shoot a particle from the sampled source
  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 3:  // mirror materials
      {
        // Forward from mirror surfaces here
        return;
      }
      break;
    case 11: // absorbing volume
    case 12: // absorbing glossy volume
      {
        // Handle absorption here (Week 11)
      }
    case 2:  // glossy materials
    case 4:  // transparent materials
      {
        // Forward from transparent surfaces here
          Ray r_out;
          HitInfo hit_out;
          float reflectance;
          trace_refracted(r, hit, r_out, hit_out, reflectance);
```

```
            if (mt_random() < reflectance && !trace_reflected(r, hit, r_out, hit_o
                return;
            else if(!hit_out.has_hit)
                return;

            r = r_out;
            hit = hit_out;
        }
        break;
      default:
        return;
      }
    }
    // Store in caustics map at first diffuse surface
    // The depth check is added to make sure that
    // no photons coming directly from a light source
    // are stored.
    if (hit.trace_depth > 0)
       caustics.store(phi, hit.position, -r.direction);
}
```

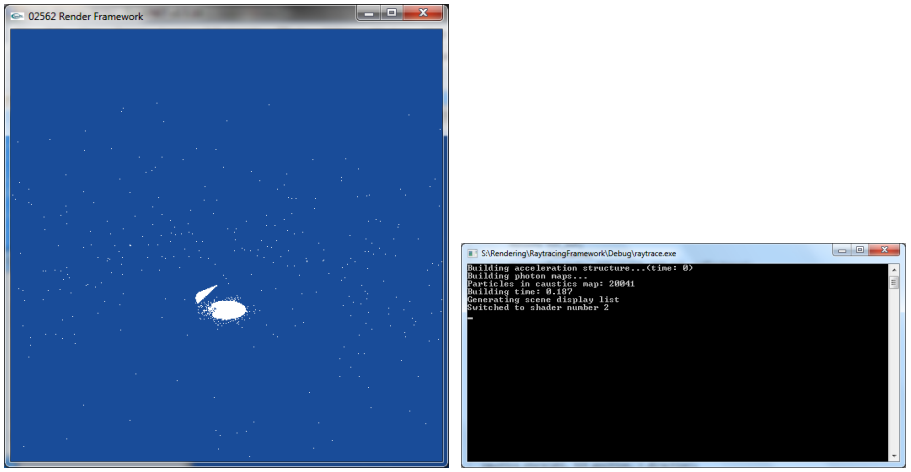

Figure 3.1: photon map preview

4 WRITING A SHADER

When we render caustics we need to convert from irradiance to radiance since the photon map gives us radiant flux over the surrounding area. The formula for calculating radiance from the photon map is given as:

$$L_r(x,\vec{w}) = \sum_{p=1}^{n} f_r(x,\overrightarrow{w_p},\vec{w})\frac{\Delta\Phi_p(x,\overrightarrow{w_p})}{\Delta A}$$

Since we are only storing the caustics photon map on difuse surfaces we can simplify the expression. Since a diffuse surface scatters light in all direction and radiance is equal to the irradiance that is reflected from the surface the solution is just to calculate the amount of photons that is inside the sphere and multiply it with the reflectivity of the surface and the BRDF of diffuse surfaces which is one over pi. In order to render the surface with normal diffuse shading we also add the result of a lambertian shading.

Listing 3: shade function from PhotonCaustics.cpp file

```
float3 PhotonCaustics::shade(const Ray& r, HitInfo& hit, bool emit) const
{
  float3 rho_d = get_diffuse(hit);
  float3 irradiance = tracer -> caustics_irradiance(hit, max_dist, photons);

  return irradiance * rho_d * M_1_PIf + Lambertian :: shade(r, hit, emit);
}
```

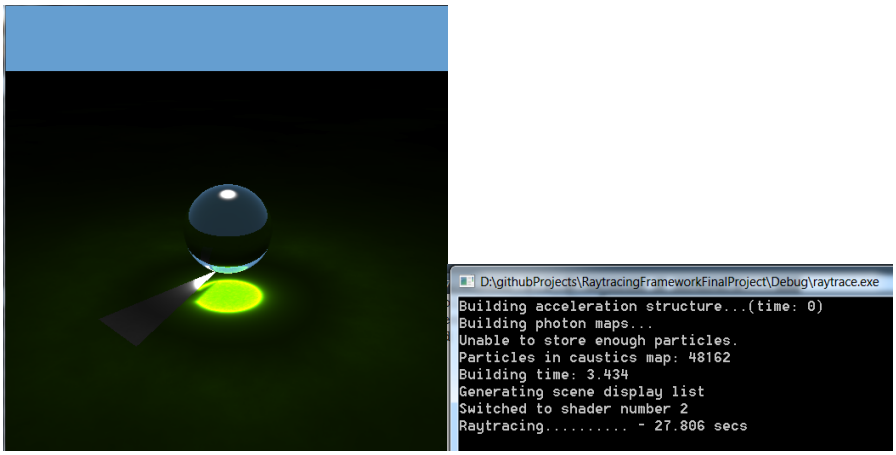

Figure 4.1: a render using caustics illumination only

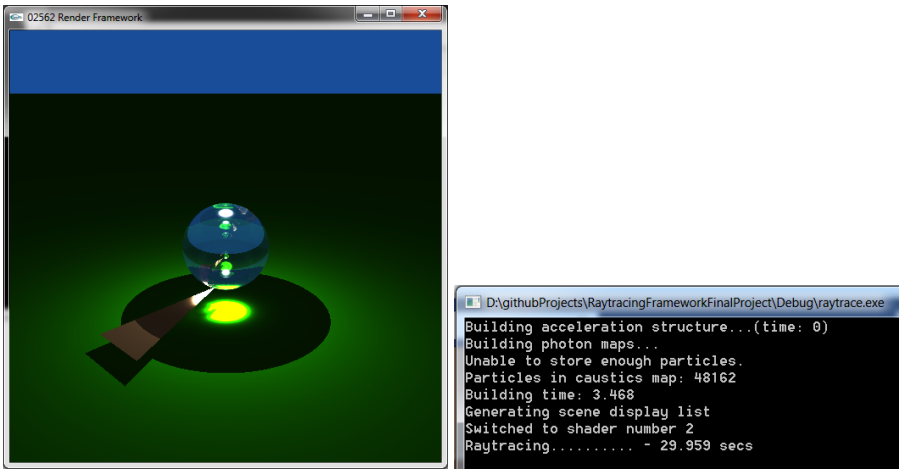

Figure 4.2: a complete render of the scene