Physically Based Rendering, Reports

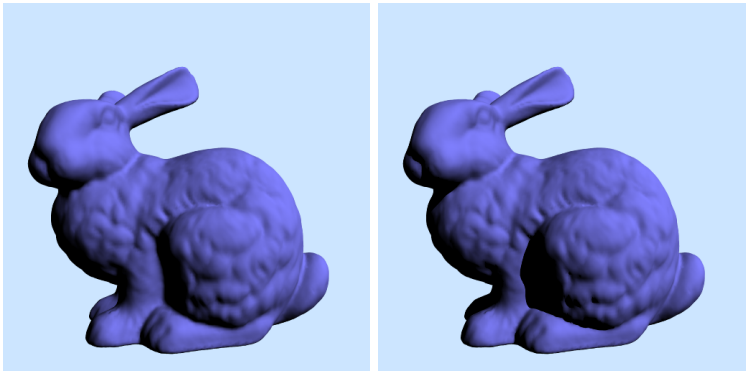
Student:
Andrea Distler, 130269

Teacher:
Jeppe Frisvald

May 9, 2013

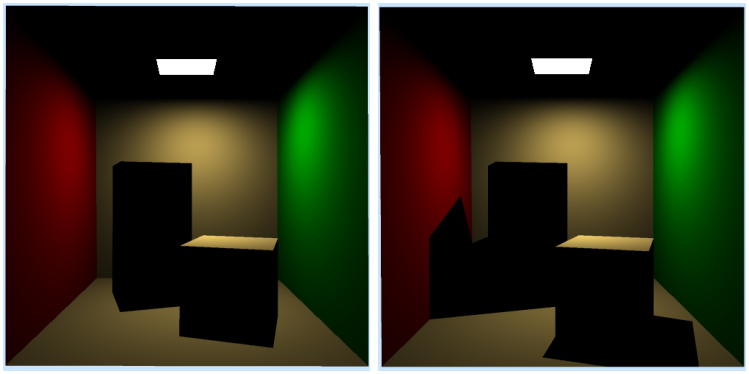
Contents

The code can be found on github:
`https://github.com/JungleJinn/Physically-Based-Rendering`



(a) Without shadows: 5s (b) With shadows: 8s

Figure 1: Bunny.obj, Tris: 69451, 36 samples, 1 directional light, Lambertian shader



(a) Without shadows: 0.3s (b) With shadows: 0.5s

Figure 2: Cornellbox.obj and CornellBlocks.obj, Tris: 36, 4 samples, 1 area light, Lambertian shaders

Report Exercise 1

- Implemented a directional light with shadows
- Implemented an area light with shadows

Relevant pictures: figures 1, and 2. Relevant listings: 1, 2, and 3.

Listing 1 : Directional.cpp

```
bool Directional::sample(const Vec3f& pos, Vec3f& dir, Vec3f& L)
    const
{
  dir = -light_dir;
  L = emission;

  // test for shadow
  Ray shadowRay(pos, -light_dir);
  bool inShadow = false;

  if (shadows)
    inShadow = tracer->trace(shadowRay);

  return !inShadow;
}
```

Listing 2 : Lambertian.cpp

```
Vec3f Lambertian::shade(Ray& r, bool emit) const
{
  Vec3f rho_d = get_diffuse(r);
  Vec3f result(0.0f);

  // temp light direction and radiance
  Vec3f lightDirection, radiance;
  for (std::vector<Light*>::const_iterator it = lights.begin(); it
      != lights.end(); it++)
  {
    if ((*it)->sample(r.hit_pos, lightDirection, radiance))
    {
      // output of Lambertian BRDF
      Vec3f f = rho_d * M_1_PIf;

      // directional light radiance
      // f - scattered light radiance, radiance - current light
          radiance, last term: cosine cut off at 0
      result += f * radiance * std::max(dot(r.hit_normal,
          lightDirection), 0.0f);
    }
  }

  return result + Emission::shade(r, emit);
}
```

Listing 3 : AreaLight.cpp

```
bool AreaLight::sample(const Vec3f& pos, Vec3f& dir, Vec3f& L)
    const
{
  // Get geometry info
  const IndexedFaceSet& geometry = mesh->geometry;
  const IndexedFaceSet& normals = mesh->normals;

  // averaged light position
  Vec3f lightPosition = Vec3f(0.0f);
  // averaged normals
  Vec3f lightNormal = Vec3f(0.0f);
  // emission summed up from all faces
  Vec3f emission = Vec3f(0.0f);

```

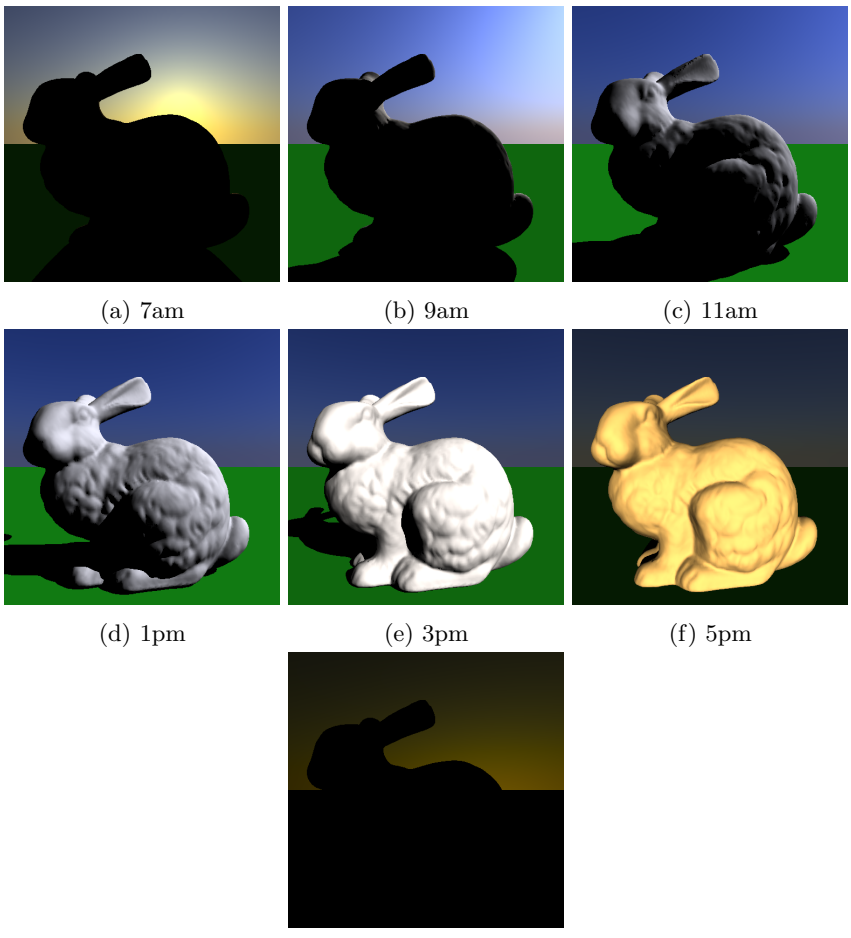
```
    // iterate over all faces
32  for (int i = 0; i < geometry.no_faces(); i++)
    {
34    // get the center of the face
      Vec3i face = geometry.face(i);
36    Vec3f v0 = geometry.vertex(face[0]);
      Vec3f v1 = geometry.vertex(face[1]);
38    Vec3f v2 = geometry.vertex(face[2]);
      Vec3f faceCenter = v0 + (v1 - v0 + v2 - v0) * 0.5f;
40
      // combine light position
42    lightPosition += faceCenter;

44    // average normals
      lightNormal += (normals.vertex(face[0]) + normals.vertex(face
          [1]) + normals.vertex(face[2])) / 3;
46
      // add emission
48    emission += mesh->face_areas[i] * get_emission(i);
    }
50
    // average light position
52  lightPosition /= geometry.no_faces();

54  lightNormal.normalize();

56  // get light direction and distance to light
    Vec3f lightDirection = lightPosition - pos;
58  float lightDistance = length(lightDirection);

60  // set area light direction, normalize
    dir = lightDirection / lightDistance;
62
    // set radiance
64  L = emission * std::max(dot(-dir, lightNormal), 0.0f) / (
        lightDistance * lightDistance);

66  // trace for shadows
    bool inShadow = false;
68  if (shadows)
    {
70    Ray shadowRay(pos, dir);
      shadowRay.tmax = lightDistance - 0.1111f;
72    inShadow = tracer->trace(shadowRay);
    }
74
    return !inShadow;
76}
```

Report Exercise 2

The input parameters for the sun sky light are theta, and phi, together creating the solar position, and the turbidity (how much light is scattered due to dirt in the atmosphere, using empirical values - Preetham). Using Preetham's paper, theta and phi are calculated from the latitude, declination, julian day of the year and the time of the day, as well as some constants from Preetham's paper. The code for this can be found in listing 4.

The model used for calculating the sun's intensity is calculated as shown in listing 5. The sun covers a solid angle of 2π degrees^2 ($\Rightarrow$ the whole hemisphere).

Relevant figure: 3. Relevant listings: 4, and 5.



(a) 7am (b) 9am (c) 11am

(d) 1pm (e) 3pm (f) 5pm

(g) 9pm

Figure 3: Bunny.obj and plane, Tris: 70.000, 1 sample, 1 skylight, Lambertian shaders $\Rightarrow$ approx. 3s per picture

Listing 4 : RenderEngine::init_tracer()

```
240 if(use_sun_and_sky)
    {
242   // Use the Julian date (day_of_year), the solar time (
          time_of_day), the latitude (latitude),
      // and the angle with South (angle_with_south) to find the
          direction toward the sun (sun_dir).
244
      // hard coded numbers are from Preetham et al.'s A Practical
          Analytical Model for Daylight, SIGGRAPH 1999
246   float declination = 0.4093 * sin(2 * M_PIf * (day_of_year - 81)
          / 368);
      float theta = M_PIf * 0.5f - asin(sin(latitude) * sin(
          declination) -
248     cos(latitude) * cos(declination) * cos(M_PIf * time_of_day /
            12));
      float phi = atan(-(cos(declination) * sin(M_PIf * time_of_day /
          12)) /
250     (cos(latitude) * sin(declination) - sin(latitude) * cos(
          declination) * cos(M_PIf * time_of_day / 12)));

252   sun_sky.setSunTheta(theta);
      sun_sky.setSunPhi(phi);
254   sun_sky.setTurbidity(turbidity);
      sun_sky.init();
256   tracer.set_background(&sun_sky);
    }
```

Listing 5 : PreethamSunSky::sample(..)

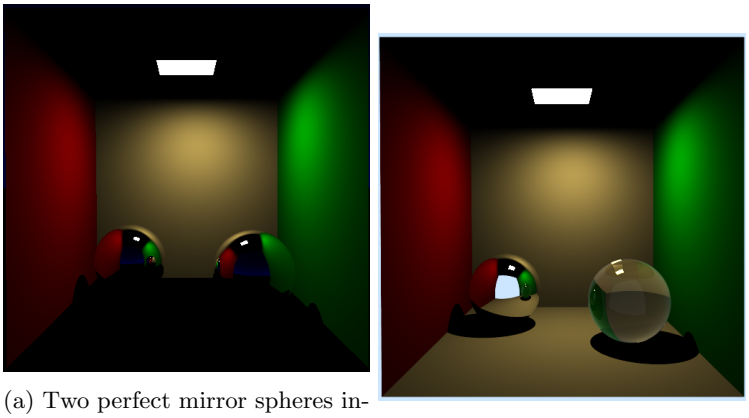
```
  bool PreethamSunSky::sample(const Vec3f& pos, Vec3f& dir, Vec3f& L)
      const
24 {
    dir = const_cast<PreethamSunSky*>(this)->getSunDir();
26
    float area = 1;
28  float solid_angle = 2 * M_PI;
    float cos_theta = dot(Vec3f(0, 1, 0), dir);
30
    // * 0.00001f to convert to the right unit (cd/m^2)
32  L = const_cast<PreethamSunSky*>(this)->sunColor() / (area *
        solid_angle * cos_theta) * 0.00001f;

34  // test for shadow
    Ray shadowRay(pos, dir);
36  bool inShadow = false;

38  if (shadows)
      inShadow = tracer->trace(shadowRay);
40
    return !inShadow;
42 }
```

Report Exercise 3

Implemented:

- Transparent shader
- Mirror shader
- Metal shader
- Russian Roulette
- fresnel equations for dielectric materials and conductors

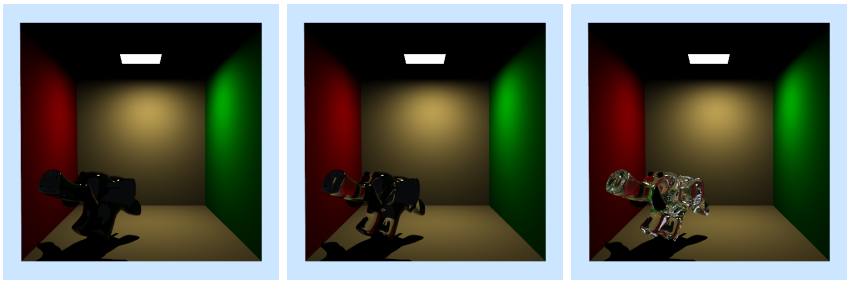
Relevant figures: 4, 5, 6, 7. Relevant listings: 7, and 6.



(a) Two perfect mirror spheres inside of the Cornell Box (using the sunsky lighting).

(b) A mirror and a glass sphere (using default lighting).

Figure 4: Two spheres with different shaders.

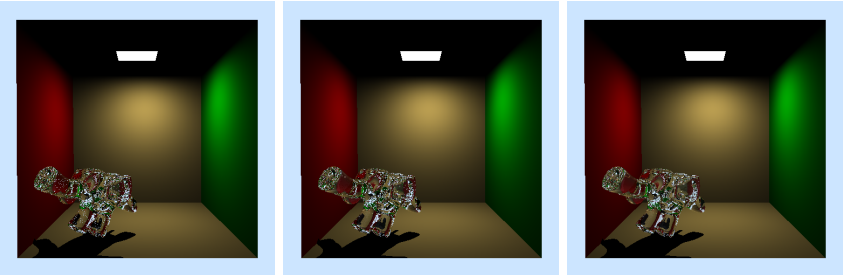


(a) Transparent elephant with cutoff of 1.

(b) Transparent elephant with cutoff of 2.

(c) Transparent elephant with cutoff of 10.

Figure 5: Transparent elephant with different cutoff depths. All pictures used 9 rays per pixel.



(a) Transparent elephant with cutoff of 1, russian roulette.

(b) Transparent elephant with cutoff of 2, russian roulette.

(c) Transparent elephant with cutoff of 10, russian roulette.

Figure 6: Transparent elephant with different cutoff dephts, using russian roulette. 9 rays per pixel

Listing 6 : Transparent::shade

```
12 Vec3f Transparent::shade(Ray& r, bool emit) const
13 {
14   Vec3f radiance = Vec3f(0.0f);
15
16   if (r.trace_depth < splits)
17   {
18     radiance = split_shade(r, emit);
19   }
20   else if (r.trace_depth < max_depth)
21   {
22     // refraction
23     Ray refracted;
24     double fresnelR;
25     tracer->trace_refracted(r, refracted, fresnelR); // fresnelR =>
           use as step probability
26
27     // russian roulette for reflections
28     float rand = randomizer.mt_random();
29
30     // 1st cond. -> russian roulette with fresnelR => pdf, 2nd cond
           . -> eliminating rays following surface
31     if (rand <= fresnelR && fresnelR > 0.001)
32     {
33       // reflect
34       Ray reflected;
35       tracer->trace_reflected(r, reflected);
36       radiance = shade_new_ray(reflected); // * fresnelR / fresnelR
             ; // divide by fresnelR, since fresnelR is used as the
             step probability
37     }
38     // if not reflecting, take refraction
39     else if (1 - fresnelR > 0.001)
40     {
41       radiance = shade_new_ray(refracted); // * (1 - fresnelR) / (1
             - fresnelR);
42     }
43   }
44
45   return radiance;
46 }
```

Listing 7 : Transparent::split_shade

```
Vec3f Transparent::split_shade(Ray& r, bool emit) const
{
  Vec3f radiance(0.0f);

  if (r.trace_depth < splits)
  {
    Ray refracted;
    double fresnelR;
    tracer->trace_refracted(r, refracted, fresnelR);

    if (1 - fresnelR > 0.001)
      radiance += shade_new_ray(refracted) * (1.0f - fresnelR);

    // eliminate rays following the surface
    if (fresnelR > 0.001)
    {
      Ray reflected;
      tracer->trace_reflected(r, reflected);
      radiance += shade_new_ray(reflected) * fresnelR;
    }
  }

  return radiance;
}
```

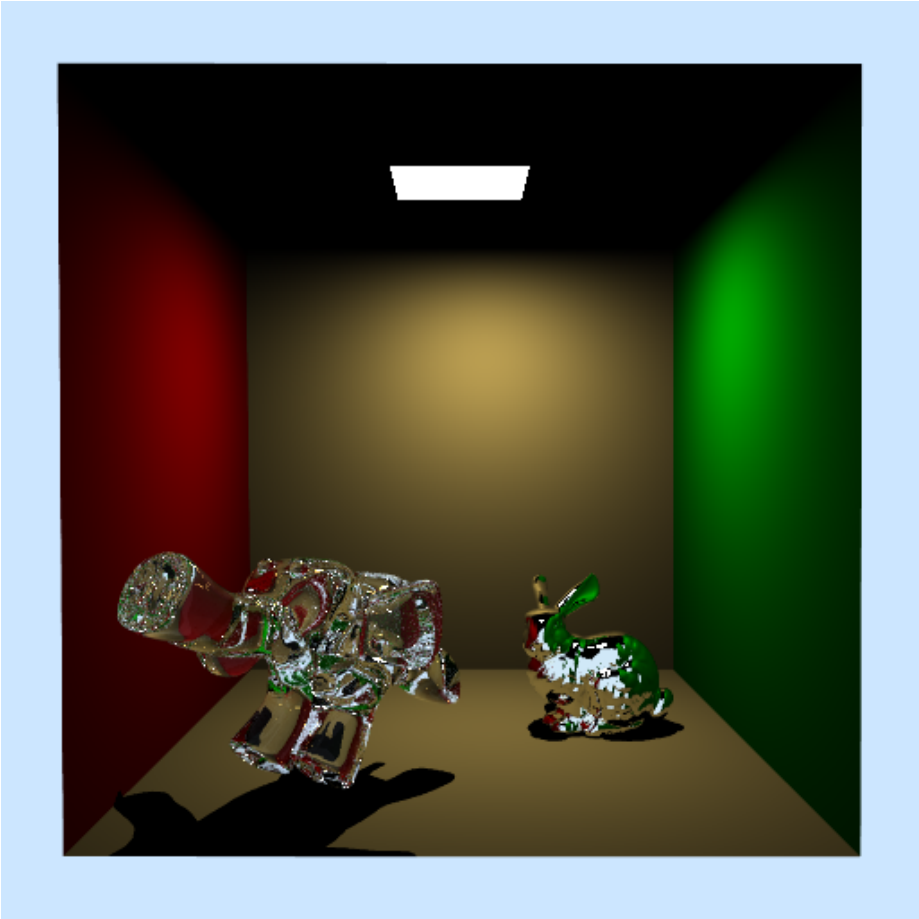


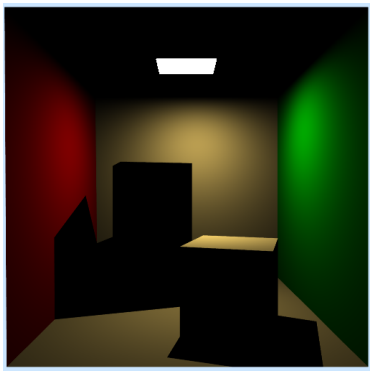
Figure 7: A silver bunny and a transparent elephant, using russian roulette, 9 rays per pixel.

Report Exercise 4

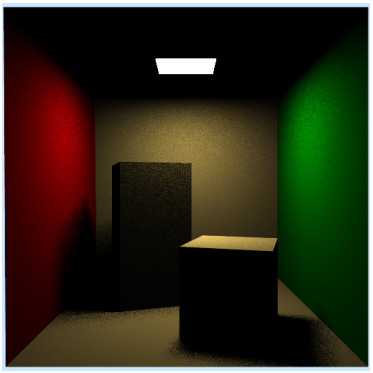
Implemented

- an area light using Monte Carlo integration
- ambient occlusion using cosine weighted sampling

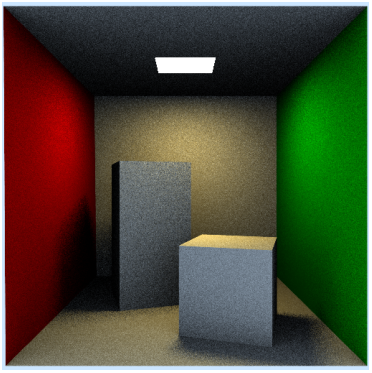
Relevant figures: 8, 9, 11, 10, and 12. Relevant listings: 8, 9, and 10.



(a) Hard shadows, 9 rays per pixel, 5 seconds



(b) Area light with Monte Carlo integration, 9rpp, 5s



(c) Area light with Monte Carlo integration and ambient light, 9rpp, 20s, 5 samples

Figure 8: The Cornellbox with its blocks, comparing hard shadows, soft shadows, and ambient light.

Listing 8 : AreaLight::sample

```
18  bool AreaLight::sample(const Vec3f& pos, Vec3f& dir, Vec3f& L)
        const
{
20    // this method uses monte carlo integration to create soft
          shadows

22    // Get geometry info
      const IndexedFaceSet& geometry = mesh->geometry;
24    const IndexedFaceSet& normals = mesh->normals;

26    // get random triangle
      int triangleIndex = randomizer.mt_random_int32() % geometry.
          no_faces();
28
      // get index for vertices of triangle
30    Vec3i vertexIndex = geometry.face(triangleIndex);
      // get index for normals of triangle
32    Vec3i normalIndex = normals.face(triangleIndex);
```

```
34    // get random position on triangle
      // ref: http://mathworld.wolfram.com/TrianglePointPicking.html
36    float sqrt_e1 = sqrt(randomizer.mt_random());
      float e2 = randomizer.mt_random();
38
      // sample barycentric coordinates
40    float u = 1 - sqrt_e1;
      float v = (1 - e2) * sqrt_e1;
42    float w = e2 * sqrt_e1;

44    // linear interpolation of vertices and normals, to get a point
          on the triangle and the according normal
      Vec3f lightPosition = Vec3f(0.0f);
46    Vec3f lightNormal = Vec3f(0.0f);

48    Vec3f uvw = Vec3f(u, v, w);
      for (int i=0; i<3; i++)
50    {
        lightPosition += geometry.vertex(vertexIndex[i]) * uvw[i];
52      lightNormal += normals.vertex(normalIndex[i]) * uvw[i];
      }
54    lightNormal.normalize();

56    // get light direction and distance to light
      Vec3f lightDirection = lightPosition - pos;
58    float lightDistance = length(lightDirection);

60    // set area light direction, normalize
      dir = lightDirection / lightDistance;
62
      // emission is scaled by geometry.no_faces() bec only 1/n are
          sampled
64    Vec3f emission = mesh->face_areas[triangleIndex] * get_emission(
          triangleIndex) * geometry.no_faces();
      // set radiance
66    L = emission * max(dot(lightNormal, -dir), 0.0f) / (lightDistance
           * lightDistance);

68    // trace for shadows
      bool inShadow = false;
70    if (shadows)
      {
72      Ray shadowRay(pos, dir);
        shadowRay.tmax = lightDistance - 0.1111f;
74      inShadow = tracer->trace(shadowRay);
      }
76
      return !inShadow;
78  }
```

Listing 9 : Ambient::shade

```
Vec3f Ambient::shade(Ray& r, bool emit) const
{
  Vec3f rho_d = get_diffuse(r);
  Vec3f radiance(0.0f);

  for (int sample = 0; sample < samples; sample++)
  {
    Ray ray;

    bool inShadow = tracer->trace_cosine_weighted(r, ray);

    if (!inShadow)
    {
      Vec3f sampleRadiance = tracer->get_background(ray.direction);
      radiance += sampleRadiance * dot(r.hit_normal, ray.direction)
          ;
    }
  }
  radiance *= rho_d / samples;

  radiance += Lambertian::shade(r, emit);

  return radiance;
}
```

Listing 10 : sampler.h sample_cosine_weighted

```
inline CGLA::Vec3f sample_cosine_weighted(const CGLA::Vec3f&
    normal)
{
  // ref: http://www.rorydriscoll.com/2009/01/07/better-sampling/
  // ref: http://pathtracing.wordpress.com/2011/03/03/cosine-
      weighted-hemisphere/
  // Get random numbers
  const float r1 = randomizer.mt_random();
  const float r2 = randomizer.mt_random();

  const float theta = acos(sqrt(1.0f - r1));
  const float phi = 2.0f * M_PI * r2;

  // Calculate new direction as if the z-axis were the normal
  CGLA::Vec3f _normal(sin(theta) * cos(phi), sin(theta) * sin(phi),
      cos(theta));

  // Rotate from z-axis to actual normal and return
  rotate_to_normal(normal, _normal);
  return _normal;
}
```

Report Exercise 5

Implemented global illumination with russian roulette and splitting at the first diffuse surface.

Relevant figures: 13, and 14. Relevant listings: 12, and 11.

In figure 14, the caustics are marked with letters. The caustics next to letter *a* are primary caustics, caused by light from the area light hitting the sphere on the top, and exiting it on the bottom (after refracting) and being reflected by the diffuse floor to the eye. The caustics next to letter *c* are hardly visible. They can be seen in the sphere's shadow best, being green caustics caused by reflected rays from the right wall. The caustics next to letter *b* are caused by the area light being reflected from the silver sphere and transmitted through the glass sphere, arriving at the right wall.

Listing 11 : MCLambertian::split_shade

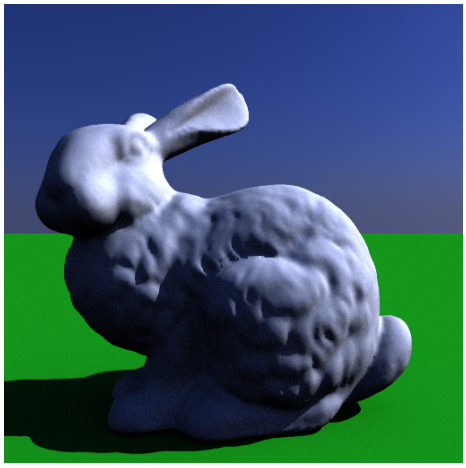
```
Vec3f MCLambertian::split_shade(Ray& r, bool emit) const
{
  Vec3f rho_d = get_diffuse(r);
  Vec3f result(0.0f);

  // indirect light
  for (unsigned int sample = 0; sample < samples; sample++)
  {
    Ray hR;
    tracer->trace_cosine_weighted(r, hR); // trace diffuse light in
        a hemisphere
    result += shade_new_ray(hR) * dot(r.hit_normal, hR.direction);
  }

  // f -> BRDF for Lambertian surfaces
  Vec3f f = rho_d / M_PI;
  result *= f / samples; // average sampled radiances

  result += Lambertian::shade(r, emit); // direct light

  return result;
}
```

Listing 12 : MCLambertian::shade

```
Vec3f MCLambertian::shade(Ray& r, bool emit) const
{
  if(!r.did_hit_diffuse)
    return split_shade(r, emit);

  Vec3f rho_d = get_diffuse(r);
  double luminance = get_luminance(rho_d);
  Vec3f result(0.0f);

  float rand = mt_random();

  // the more light is being reflected by a surface, the higher the
      probability for reflection should be.
  // also: weight different diffuse colors depending on human eye
      sensitivity
```

```
26    float y = 0.2989 * rho_d[0] + 0.5866 * rho_d[1] + 0.1145 * rho_d
          [2];
      //float y = (rho_d[0] + rho_d[1] + rho_d[2]) / 3.0f;
28
      // reference: http://www.youtube.com/watch?v=xIPKmbuVHQI
30    if (rand < y)
      {
32      // direct light
        result += Lambertian::shade(r, emit) / y; // divide by
            probability to make sure that monte carlo gives valid
            results
34
        // indirect light
36      for (unsigned int sample = 0; sample < samples; sample++)
        {
38        Ray hR;
          tracer->trace_cosine_weighted(r, hR); // trace diffuse light
              in a hemisphere
40        result += shade_new_ray(hR) * dot(r.hit_normal, hR.direction)
               / y / samples;
        }
42
        // f -> BRDF for Lambertian surfaces
44      Vec3f f = rho_d / M_PI;
        result *= f; // average sampled radiances
46    }
      else // not tracing this ray any further
48    {
        result += Lambertian::shade(r, emit);
50    }

52    return result;
  }
```

(a) The bunny with sunsky lighting model, 4rpp, 15s



(b) The bunny with sunsky lighting model at 12am, with ambient light. 9rpp, 83s

Figure 9: The bunny in different light conditions. Ambient light adds much to the realism of the picture.

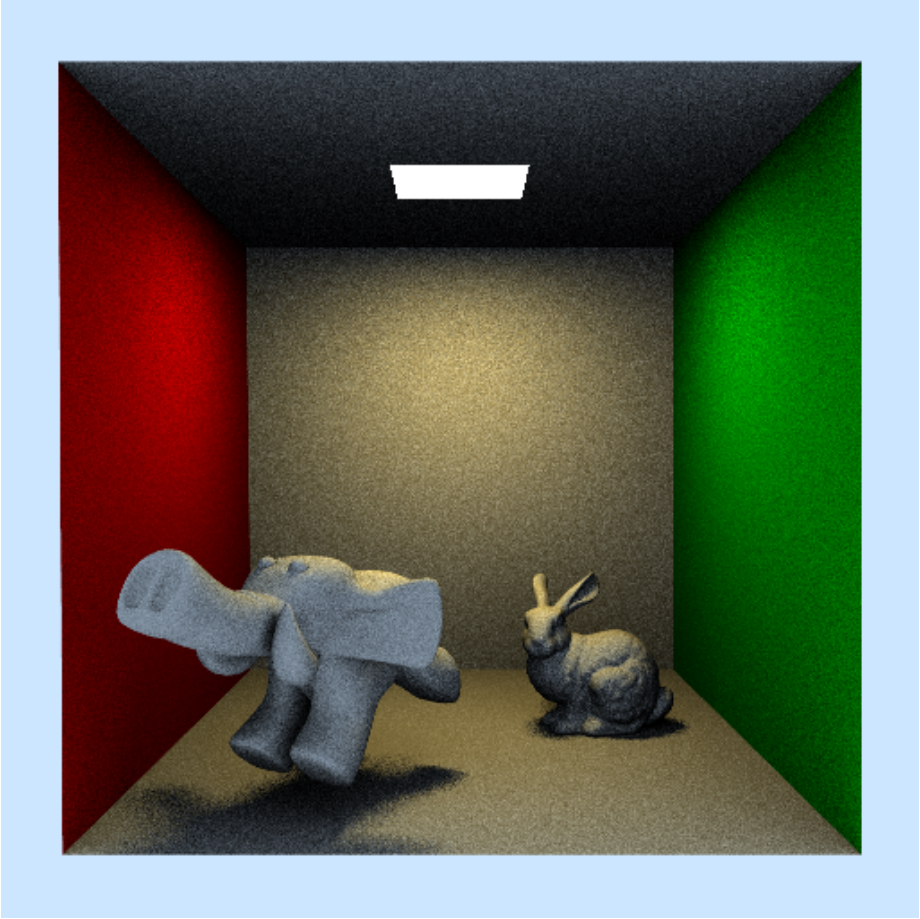
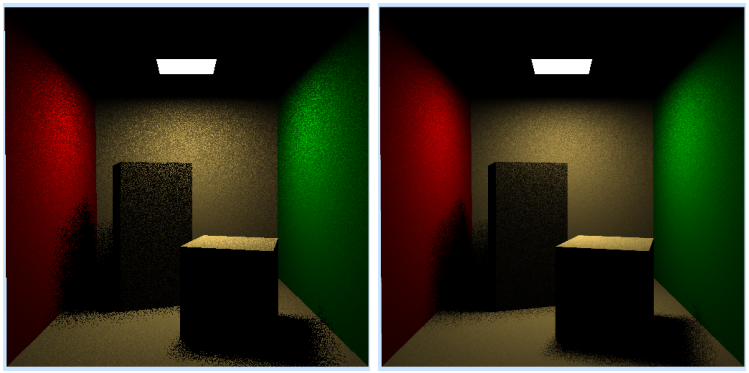
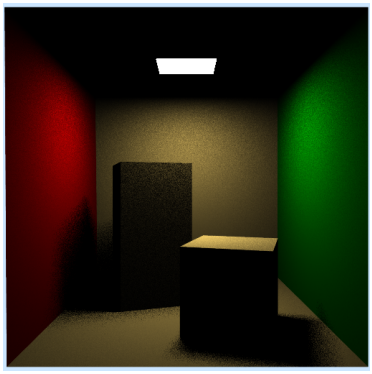


Figure 10: Bunny and elephant inside the cornellbox, ambient and area light. 9rpp, 36s, 5 samples

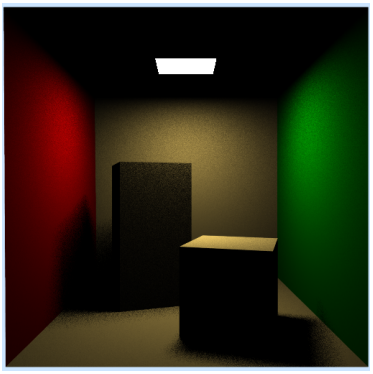


(a) 1rpp, 1s

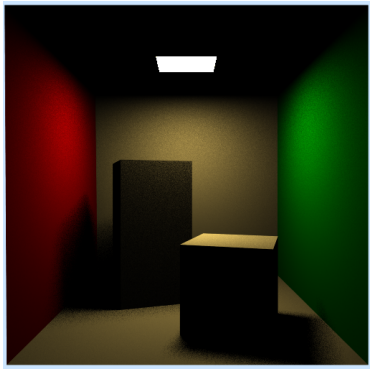
(b) 4rpp, 2s



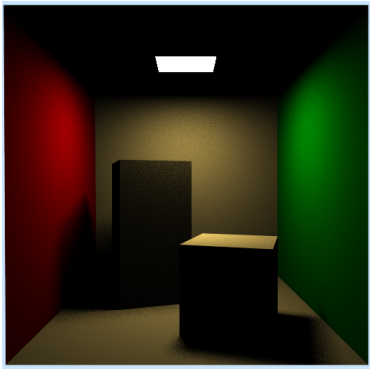
(c) 9rpp, 5s



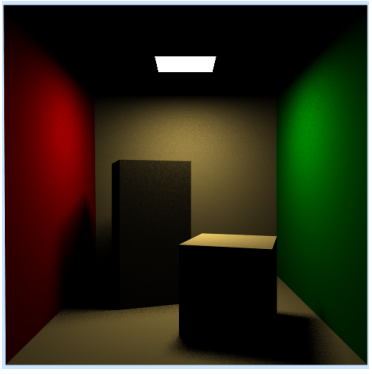
(d) 16rpp, 10s



(e) 25rpp, 17s

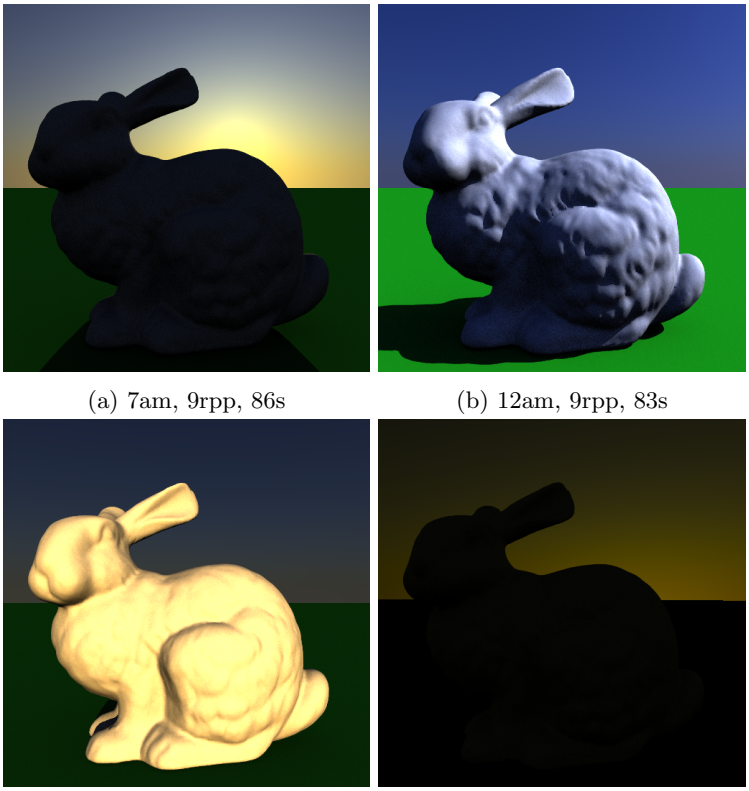


(f) 36rpp, 22s



(g) 49rpp, 34s

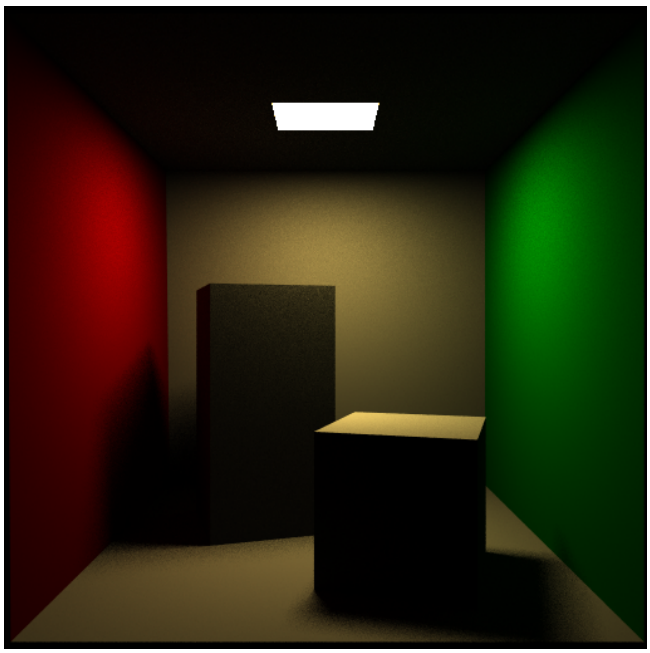
Figure 11: Different ray densities, between 1 to 49 rays per pixel.



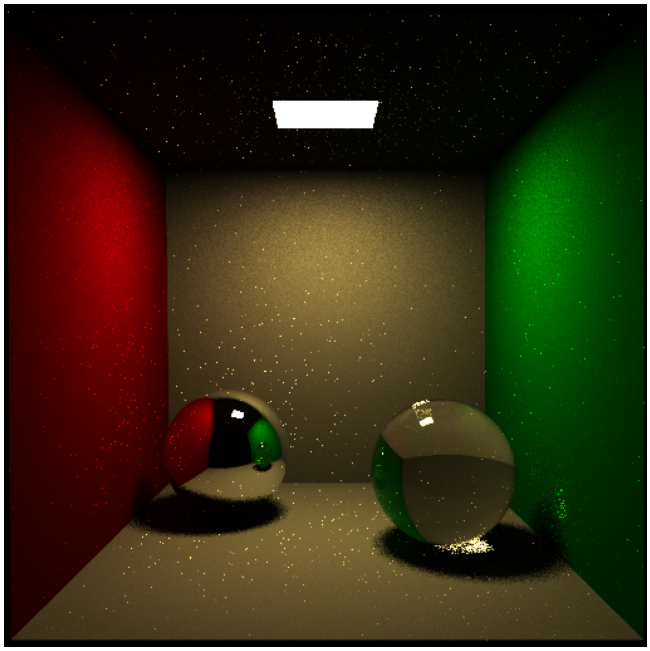
(a) 7am, 9rpp, 86s (b) 12am, 9rpp, 83s

(c) 7pm, 9rpp, 84s (d) 7pm, 9rpp, 245s

Figure 12: The bunny at different times of the day. Lightmodel: sunsky, using ambient lighting



(a) Cornell box with blocks. 75 iterations, 164 seconds, 5 samples, 9 rays per pixel.



(b) Cornell box with silver and glass sphere. 25 iterations, 95 seconds, 5 samples, 9rpp.

Figure 13: The Cornell box with diffuse, transparent, and metal materials. Both images use global illumination with splitting at the first diffuse surface and russian roulette for following diffuse surfaces.

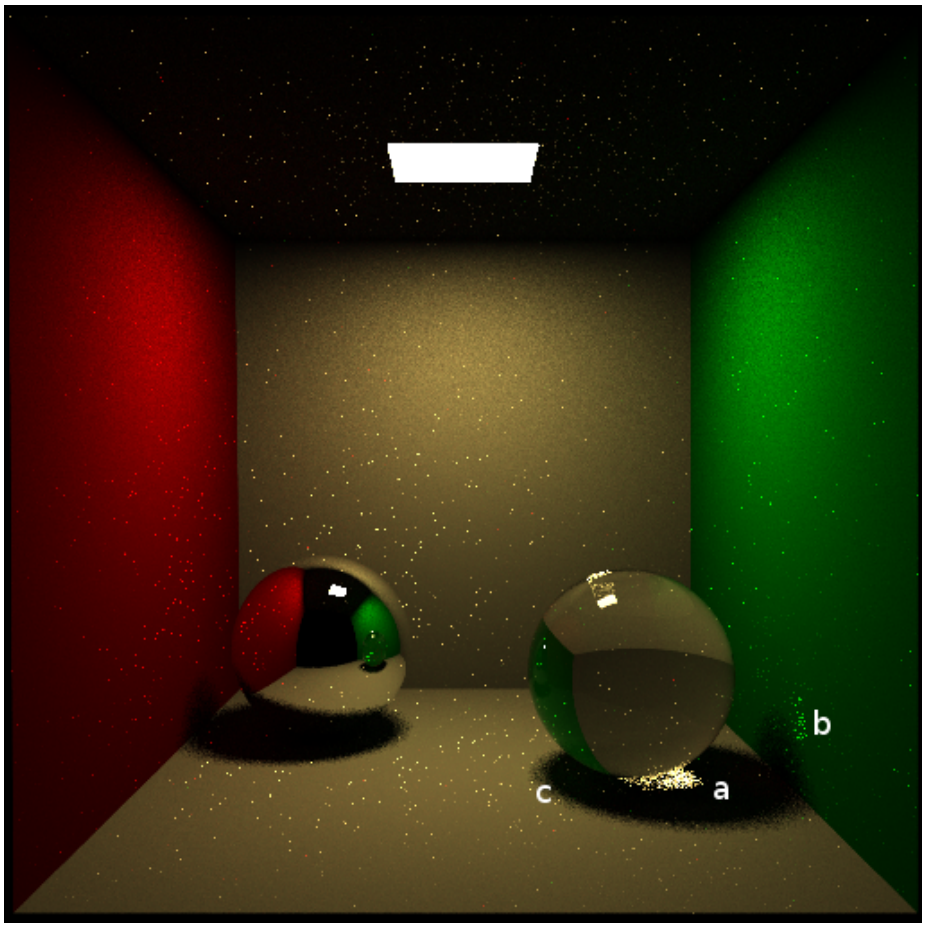


Figure 14: Figure 13b with indicators for caustics.