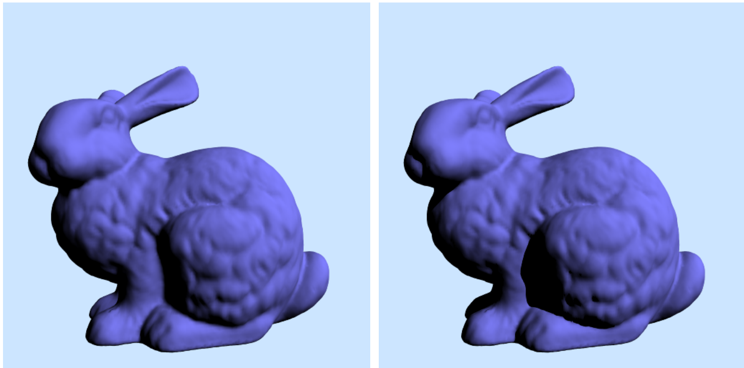
Physically Based Rendering, Reports

Student:
Andrea Distler, 130269

Teacher:
Jeppe Frisvald

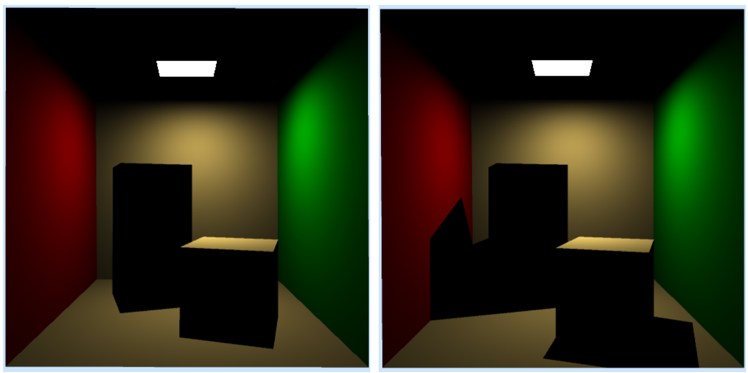
May 2, 2013

Contents



(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);
```

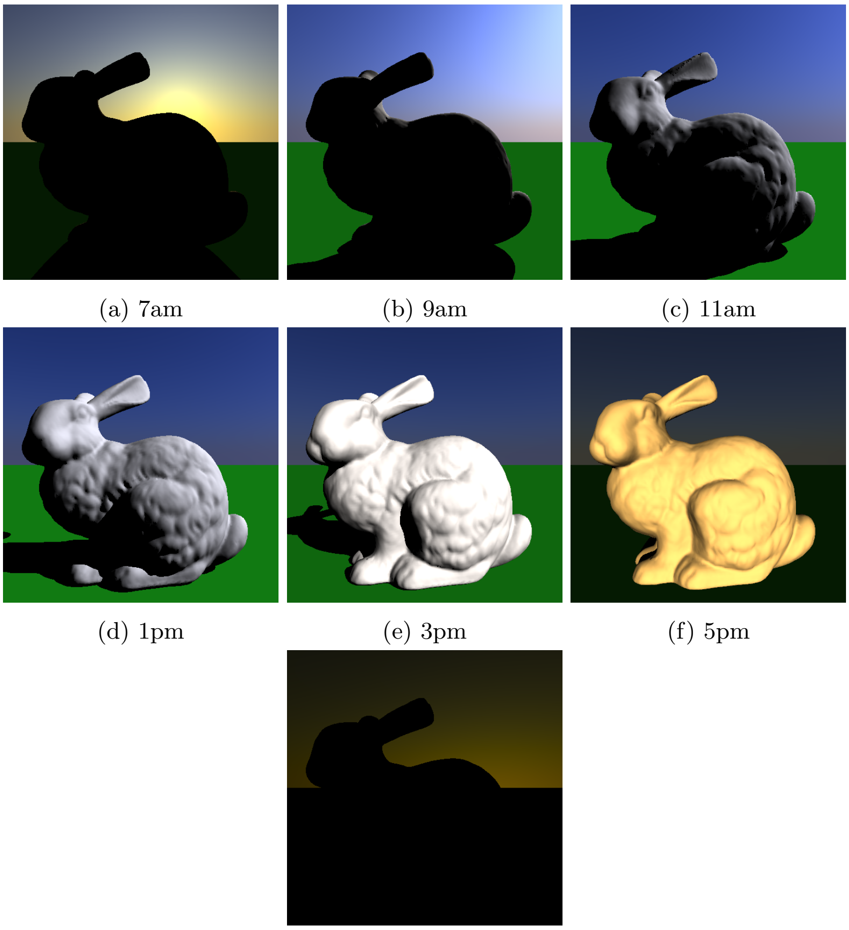
```
  // iterate over all faces
  for (int i = 0; i < geometry.no_faces(); i++)
  {
    // get the center of the face
    Vec3i face = geometry.face(i);
    Vec3f v0 = geometry.vertex(face[0]);
    Vec3f v1 = geometry.vertex(face[1]);
    Vec3f v2 = geometry.vertex(face[2]);
    Vec3f faceCenter = v0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

    // combine light position
    lightPosition += faceCenter;

    // average normals
    lightNormal += (normals.vertex(face[0]) + normals.vertex(face
        [1]) + normals.vertex(face[2])) / 3;

    // add emission
    emission += mesh->face_areas[i] * get_emission(i);
  }

  // average light position
  lightPosition /= geometry.no_faces();

  lightNormal.normalize();

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

  // set area light direction, normalize
  dir = lightDirection / lightDistance;

  // set radiance
  L = emission * std::max(dot(-dir, lightNormal), 0.0f) / (
      lightDistance * lightDistance);

  // trace for shadows
  bool inShadow = false;
  if (shadows)
  {
    Ray shadowRay(pos, dir);
    shadowRay.tmax = lightDistance - 0.1111f;
    inShadow = tracer->trace(shadowRay);
  }

  return !inShadow;
}
```

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()

```
if(use_sun_and_sky)
  {
    // 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).

    // hard coded numbers are from Preetham et al.'s A Practical Analytical Model for Daylight, SIGGRAPH 1999
    float declination = 0.4093 * sin(2 * M_PIf * (day_of_year - 81) / 368);
    float theta = M_PIf * 0.5f - asin(sin(latitude) * sin(declination) -
      cos(latitude) * cos(declination) * cos(M_PIf * time_of_day / 12));
    float phi = atan(-(cos(declination) * sin(M_PIf * time_of_day / 12)) /
      (cos(latitude) * sin(declination) - sin(latitude) * cos(declination) * cos(M_PIf * time_of_day / 12)));

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

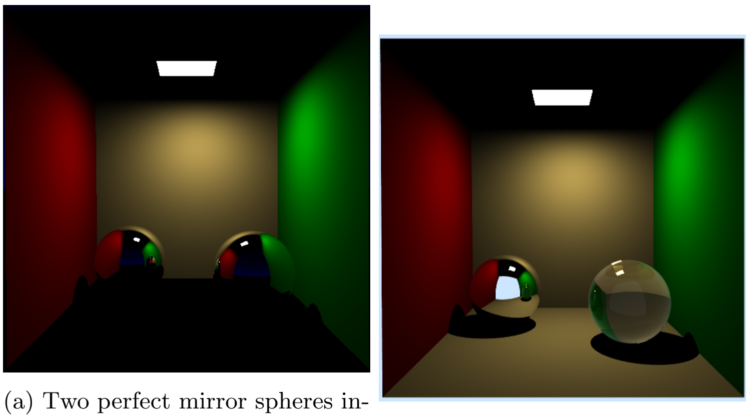
Listing 5 : PreethamSunSky::sample(..)

```
bool PreethamSunSky::sample(const Vec3f& pos, Vec3f& dir, Vec3f& L) const
{
  dir = const_cast<PreethamSunSky*>(this)->getSunDir();

  float area = 1;
  float solid_angle = 2 * M_PI;
  float cos_theta = dot(Vec3f(0, 1, 0), dir);

  // * 0.00001f to convert to the right unit (cd/m^2)
  L = const_cast<PreethamSunSky*>(this)->sunColor() / (area * solid_angle * cos_theta) * 0.00001f;

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

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

  return !inShadow;
}
```

Report Exercise 3

Implemented:

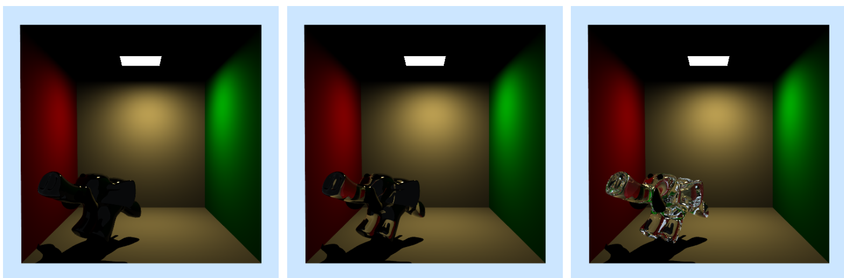
- Transparent shader
- Mirror shader
- Metal shader
- Russian Roulette
- fresnel equations for dielectric materials and conductors



(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.

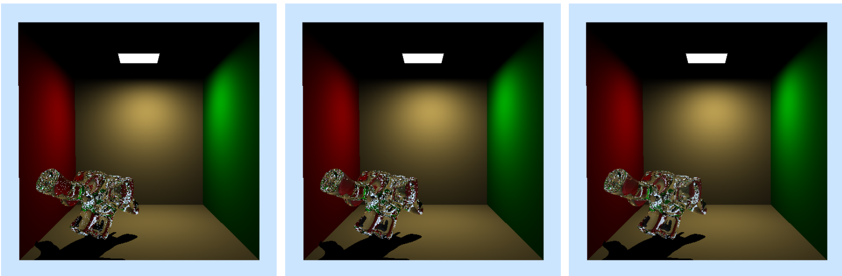


(a) Transparent elephant with cutoff of 1.

(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.

(b) Transparent elephant with cutoff of 2.

(c) Transparent elephant with cutoff of 10.

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