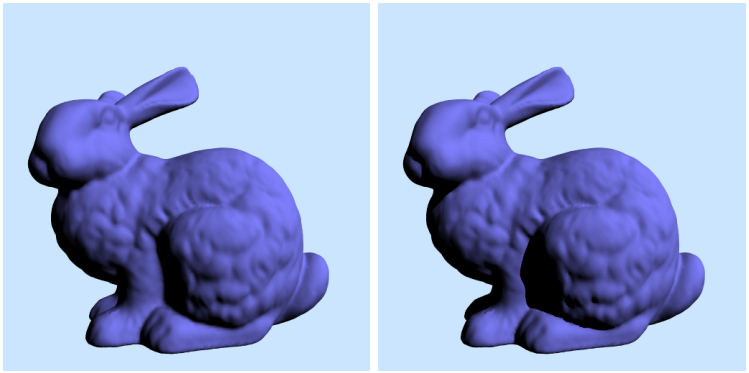
Physically Based Rendering

Student:
Andrea Distler, 130269

Teacher:
Jeppe Frisvald

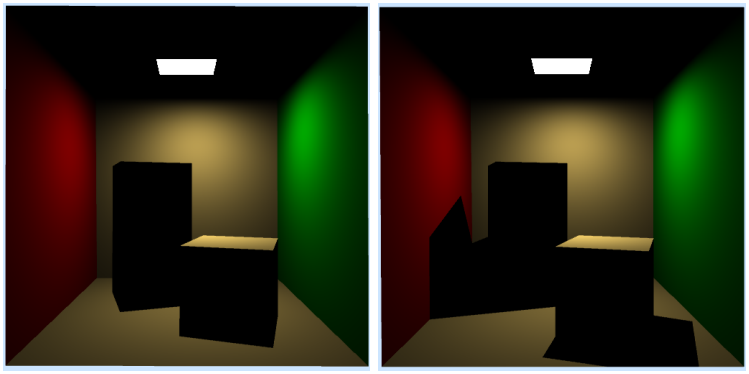
March 4, 2013

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

1 Report Exercise 1

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

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);
```

```
14    // iterate over all faces
      for (int i = 0; i < geometry.no_faces(); i++)
16    {
        // get the center of the face
18      Vec3i face = geometry.face(i);
        Vec3f v0 = geometry.vertex(face[0]);
20      Vec3f v1 = geometry.vertex(face[1]);
        Vec3f v2 = geometry.vertex(face[2]);
22      Vec3f faceCenter = v0 + (v1 - v0 + v2 - v0) * 0.5f;

24      // combine light position
        lightPosition += faceCenter;
26
        // average normals
28      lightNormal += (normals.vertex(face[0]) + normals.vertex(face
            [1]) + normals.vertex(face[2])) / 3;

30      // add emission
        emission += mesh->face_areas[i] * get_emission(i);
32    }

34    // average light position
      lightPosition /= geometry.no_faces();
36
      lightNormal.normalize();
38
      // get light direction and distance to light
40    Vec3f lightDirection = lightPosition - pos;
      float lightDistance = length(lightDirection);
42
      // set area light direction, normalize
44    dir = lightDirection / lightDistance;

46    // set radiance
      L = emission * std::max(dot(-dir, lightNormal), 0.0f) / (
          lightDistance * lightDistance);
48
      // trace for shadows
50    bool inShadow = false;
      if (shadows)
52    {
        Ray shadowRay(pos, dir);
54      shadowRay.tmax = lightDistance - 0.1111f;
        inShadow = tracer->trace(shadowRay);
56    }

58    return !inShadow;
  }
```