Computer Graphics SoSe16
Exercise 4

Tobias Knopp & Martin Hofmann
Institute for Biomedical Imaging

Next exercises:
06.06.2016, 14:15-15:45 & 16:30-18:00
13.06.2016, 14:45-15:45 & 16:30-18:00

In the last exercise we set up a basic ray tracer to render a very simple scene consisting of two cubes and two spheres. As a next step we will explore some advanced topics of ray tracing. More advanced lighting models, shadow casting, object transformations and reflections.

1. As a first task set up the latest renderings from the last exercise and make sure the following code is running.

```
# latest renderings as reference
# set up scene
sphere1 = Sphere(Float32[-0.5,0.5,0],0.25f0)
sphere2 = Sphere(Float32[-0.5,-0.5,0],0.5f0)
aabb1 = AABB(Float32[0.5,-0.5,0],0.25f0,0.25f0,0.25f0)
aabb2 = AABB(Float32[0.5,0.5,0],0.5f0,0.5f0,0.5f0)
scene = Scene(SceneObject[sphere1,sphere2,aabb1,aabb2])
# set up camera
camera = PinholeCamera(Float32[0,0,1],Float32[0,0,-1],Float32[0,1,0])
# set up lights
pointLight1 = PointLight(Vec4f(0.5,-0.5,0.3,1))
pointLight2 = PointLight(Vec4f(0,0,5,1))
sceneLights1 = SceneLights(Lights[pointLight1,pointLight2])

# trace rays using two hit and Lambert shader
tracerays(scene, camera, sceneLights1, hitShader)
tracerays(scene, camera, sceneLights1, lambertShader)
```

2. Before we start to integrate shadow into our shader write a shader `lightShader(ray::Ray,scene::Scene,sceneLights::SceneLights)` to shade each hitpoint with respect to the number of light sources which potentially influence this point. This can be done quite simple by checking if the surface normal and the vector pointing from hitpoint to the light sources have positive dot product. Note that we count ambient light as one light source. Check the result by rerendering the `scene` using the `lightShader`

```
tracerays(scene, camera, sceneLights1, lightShader)
```

This shader does not account for occlusion of light sources by other objects. So in order to implement a correct `shadowShader(ray::Ray,scene::Scene,sceneLights::SceneLights)` you will have to shoot a ray from each hitpoint to light source and check, if any objects occlude the path. Comparing the `lightShader` and the `shadowShader` you should be able to see additional shadows on the surface of each of the two spheres.

Note: Avoid collision of your primary object and the ray to the light source if checking for occlusion.

3. Up till now we only used point like light sources, which result in hard shadows. I.e. an infinitely sharp edge between shadow and light areas. For natural lights however we do not observe such sharp transitions due to the fact that natural lights are usually extended.

   In ray tracing we might model this by a new type

```
1  import Base:cross, norm
2  cross(v::Vec4f,w::Vec4f) = Vec4f(v.e2*w.e3-v.e3*w.e2,v.e3*w.e1-v.e1*w.e3,v.
       e1*w.e2-v.e2*w.e1,0f0)
3  norm(v::Vec4f) = sqrt(dot(v,v))
4
5  type FlatLight <: Lights
6      center::Vec4f
7      height::Vec4f
8      width::Vec4f
9      samplepoints::Int
10     intensity::Float32
11
12     FlatLight(center::Vec4f,height::Vec4f,width::Vec4f,samplepoints::Int) =
           new(center,height,width,samplepoints,Float32(norm(cross(height,width)
           )/samplepoints))
13 end
```

   which corresponds to a flat isotropic light source shaped like a parallelogram spanned by `height` and `width` with its center located at `center`.

   To calculate a realistic lighting of this flat light source at a certain point $\vec{p}$ we would have to evaluate which part of the light source is occluded by scene objects

$$\int_L o(\vec{r},\vec{p})d^3\vec{r}. \tag{1}$$

   Here the function $o(\vec{r},\vec{p})$ is zero, if the direct path from $\vec{p}$ to $\vec{r}$ is occluded and one otherwise. Integration has to be performed over the whole surface area of the light source L.

   Though we usually cannot evaluate (1), we can use Monte Carlo integration for evaluation. Drawing N uniform samples $r_i \in L$ and calculating the Area A_L of L, (1) can be approximated by

$$\frac{A_L}{N}\sum_{i_1}^{N} o(\vec{r}_i,\vec{p}). \tag{2}$$

   This corresponds to the approximation of our flat light source by N point like light sources which are uniformly distributed over the surface of the flat light source and shine with an intensity of A_L/N.

Write a function `positions(flatLight::FlatLight)`, which uniformly draws sample points from the surface area of the light source. Adapt the shadow shader to count each sample point as independent light source and render

```
flatLight1 = FlatLight(Vec4f(0,0,1,1),Vec4f(1,0,0,0),Vec4f(0,1,0,0),25)
flatLight2 = FlatLight(Vec4f(0,0,1,1),Vec4f(1,0,0,0),Vec4f(0,1,0,0),250)
sceneLights2 = SceneLights(Lights[flatLight1])
sceneLights3 = SceneLights(Lights[flatLight2])
tracerays(scene, camera, sceneLights2, shadowShader)
tracerays(scene, camera, sceneLights3, shadowShader)
```

What happens, if the number of sampling points is too low? What might influence the number of sampling points needed to create a noise free image?

4. With the basic shadow casting shader and a more realistic light source in place let us add some more realistic lighting. We start off with the lighting model used in the last session. This model contains ambient lighting. A constant term `a` added to any hit point regardless of its position with respect to all the light sources. Additionally, a Lambert light term was added, which was proportional to the dot product of normalized surface `normal` and the normalized direction from hit point to light source `l`.

   This lighting model very well reproduces the surface of rough surfaces (clay). Most often however a surface has some degree of shininess (polished marble, brushed metal), which leads to bright spots (highlights). Observe that in nature these highlights are centered around the point on the surface, which would reflect the light of the light source directly to the observer. Around this center the intensity usually falls off very quick (slower for rougher surfaces).

   This behavior can be captured by using the Pong lighting model. Starting off with the ambient and Lambert term we add another term. Namely the dot product of the backwards unit direction of the camera ray `e` and the unit direction of light reflected at the surface `r` to the power of `phongExponent`.

   All in all one ends up with the following combined shade for a single not occluded point like light source.

   ```
   shade = a + max(0,dot(normal,l)) + max(0,dot(r,e))^phongExponent
   ```

   Implement a `shadowBlinnPhongShader(ray::Ray,scene::Scene,sceneLights::SceneLights;phongExponent=64)` with realistic shadow casting, which works with any of our light types. Use this shader to render

   ```
   tracerays(scene, camera, sceneLights1, shadowBlinnPhongShader)
   tracerays(scene, camera, sceneLights3, shadowBlinnPhongShader)
   ```

5. As a last exercise you will add reflections to the surface of each object (`shadowReflectionBlinnPho`). The primary shade of a hit point is still given by the light model of the last exercise. To add reflections follow the ray reflected at the surface and add half of the shade of the reflected ray to the shade of the hit point. Follow this recursive strategy until you reach a depth of 3 and render the following scene.

   ```
   tracerays(scene, camera, sceneLights1, shadowReflectionBlinnPhongShader)
   ```