UNIVERSITÄT DES SAARLANDES
PROF. DR.-ING. PHILIPP SLUSALLEK
COMPUTER GRAPHICS GROUP
STEFAN LEMME (LEMME@CG.UNI-SAARLAND.DE)

26. NOVEMBER 2018

INTRODUCTION TO COMPUTER GRAPHICS
ASSIGNMENT 5

Submission deadline for the exercises: 3. December 2018

The paper copies for the theoretical parts of the assignments will be collected at the beginning of the lecture on the due date. The programming parts must instead be marked as release before the beginning of the lecture on the due date.

The code submitted for the programming part of the assignments is required to reproduce the provided reference images. The submission ought to tag the respective commit in your group's git repository as well as attach the mandatory generated images to the release. The submission should also contain a creative image show-casing all extra-credit features that have been implemented.

The projects are expected to compile and work out of the box. A successful build by Drone CI is a good indicator. If it fails to do so on our Windows or Linux computers, we will try to do so on the CIP-pool students' lab as a "fallback".

To pass the course you need for every assignment at least 50% of the points.

5.1 Rendering Equation (10 Points)

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega^+} L_i(x, \omega_i)\, f(\omega_i, x, \omega_o)\, cos\,\theta_i \, \mathrm{d}\omega_i$$

In the practical assignment, we ask you to implement an algorithm which computes an approximation of this equation. Assuming only point lights are used, describe the approximation that is performed. Explain in detail what is being approximated and give a formula for $L_o(x, \omega_o)$ that describes what is really computed, given:

- N, the number of lights,

- f, the BRDF at point x,

- n, the normal at point x,

- $V(x, y) = \begin{cases} 1 & \textit{if x is visible from y} \\ 0 & \textit{otherwise} \end{cases}$, the visibility function,

- y_i, the position of the light source i,

- I_i, the intensity of the **point light** i.

Note: You can write the direction pointing from the point x to the point y as $x \to y$

5.2 Basic Materials (10 Points)

When a ray intersects a solid object, lighting at the hit point has to be computed. This depends on:

- The light sources around the hit point

- The material of which the solid is made

A `Material` is defined by:

- The reflectance (`getReflectance`) computes the amount of light that is reflected on a material surface. It depends on the following parameters:

  - `inDir` – the direction vector (in world coordinate space) from which the light is incoming
  - `outDir` – the direction to the viewer, where the light is reflected to
  - `normal` – the surface normal vector at the hit point
  - `texPoint` – the local coordinate of the hit point. Currently it should be simply `Intersection::local()`, in the future we will compute some texture coordinates to supply this function.

- The emission (`getEmission`) of the material itself towards given direction. Similarly to reflectance, it requires `outDir`, `normal` and `texPoint`.

Note that reflectance denotes only the percentage of that light that is being reflected. It does *not* compute the actual amount of light. For that reason you never need to trace any additional rays from within the material. Shooting rays is the job of the `Integrator`.

In some cases it is convenient for computational purposes to include the cosine term from the rendering equation into the material itself, and that is why the `getReflectance` function should account for the cosine term as well.

While these two functions can fully describe the material, they are sometimes insufficient for practical reasons. Only the material "knows" which combination of the above parameters yield significant results. For that reason a `Material` provides additional functionality - sampling - which we will discuss in the future. At the moment however the sampling-related functions may be left unimplemented.

Your task is to implement a very simple material (`DummyMaterial`) which should emit no light, and have reflectance $fr = 1$. (multiplied by the cosine term from the rendering equation)

You should also update all `Primitive` classes to now support `setMaterial`. Calling this function on a single `Solid` should set the material of that solid, while calling it on a `Group` should change the material of all the contained objects.

5.3 Lighting (5 + 30 Points)

In order to compute lighting we need light sources in the scene. A `Light` is an object that lights the scene and as such is being queried only in relation to some hit point between a ray and a solid. It is defined by two functions:

- `getLightHit` — provides a direction vector from the intersection point `p` towards the light source, and a distance along this vector.

- `getIntensity` — computes the amount of light reaching the intersection point (defined by the result of `getLightHit`) assuming no obstacles in between.

Note that these functions define only the light itself. We do not care about the scene, shadows, etc... that will be the task of the `Integrator`.

As a simple realization of a light source, implement a `PointLight` source. A point light is defined by a position in space and its intensity. The effective irradiance dissipates proportionally to the square distance from the source to the hit point.

Implement a new `RayTracingIntegrator`. When a hit point is found, it should compute the lighting:

- Iterate over all light sources in the `World`.

- For each light source compute the shadow ray.

- Confirm that the shadow ray and the primary ray leave the surface on the same side.

- Trace a shadow ray into the world to detect any obstacles between the light source and the hit point.

- If no obstacle is found, query the light for the intensity.

- Use the irradiance intensity and the material properties to compute the amount of reflected light.

- In addition add the emitted light of the material.

Figure 1: Final image using the colored point light sources. In the code we ask to create two images of the same scene with different scale, but the output should be identical.

5.4   Light Sources (8 + 12 Points)

Implement the following light sources:

- `DirectionalLight` — a light that illuminates the whole scene from a single direction. The distance to the light source is always infinity (use `FLT_MAX`) and the intensity is constant.

- `SpotLight` — clipped Warn spot light. It is similar to point light, except that it illuminates only in a cone. The irradiance is nonzero only along those directions that form an angle lower than `angle` with a specified `direction` vector, with intensity proportional to the cosine lobe raised to the exponent `exp`.

5.5   More Light Sources (15 Points)*

Implement a `ProjectiveLightSource` that behaves similarly to `PointLight` but emits a different light color depending on the direction vector. Use the Julia function as the projection pattern.
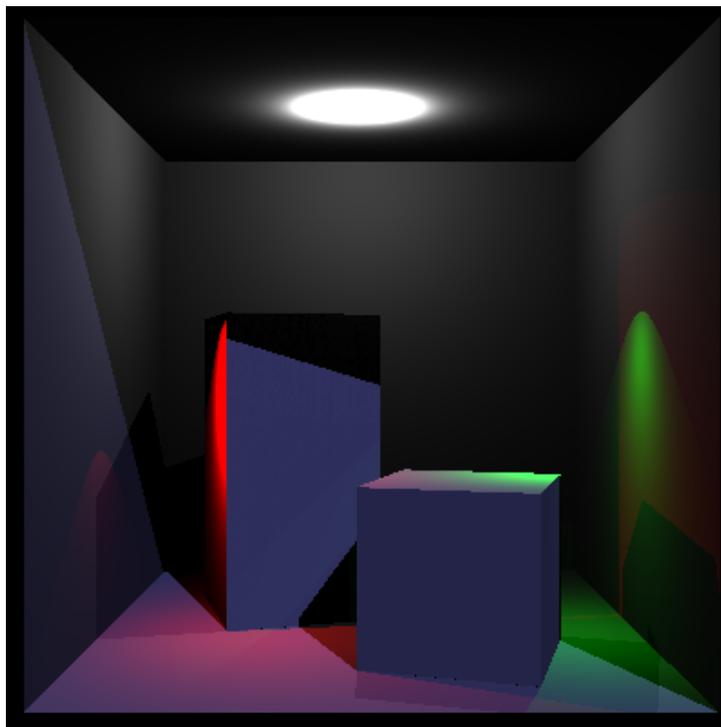
Figure 2: Final image using the colored spot lights (red and green) and a directional light (blue).