Universität Tübingen
Prof. Dr.-Ing. Hendrik P.A. Lensch
Lehrstuhl Computergrafik
Dennis Bukenberger (dennis.bukenberger@uni-tuebingen.de)
Zabiha Kehribar (zabiha.kehribar@student.uni-tuebingen.de)

30. November 2020

Graphische Datenverarbeitung
Assignment 4

Submission deadline for the exercises: 7. December 2020 6.00 am

Source Code Solutions

- Upload **only** the source code files listed in the description in Ilias.
- Upload them one by one and **don't** zip them.

The source code must run on `cgpool120[0-7].informatik.uni-tuebingen.de` by extracting your submitted `.tar.gz` file to a certain folder and running `scons`. You can log onto this machine via ssh by using your WSI account. From outside the WSI network, you may have to use a ssh gateway, e.g. `cgcontact.informatik.uni-tuebingen.de`
Attention: Do not use `cgcontact` for working, but only for ssh-ing to the `cgpool` machines.

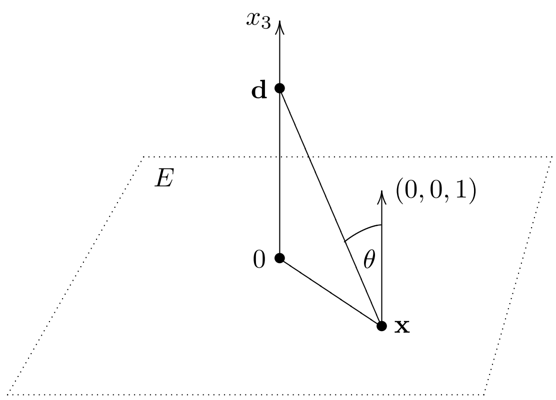
The framework you get for completion already compiles and runs as requested above and you only have to modify source and header files - no files have to be created.

Written Solutions

Written solutions have to be submitted digitally as one PDF file via Ilias.

4.1 Radiometry (written, 30 Points)

A point light source with isotropic radiance is placed in $\mathbf{d} = (0, 0, d) \in \mathbb{R}^3, d > 0$. Let its power be $\Phi_S = 100W$. Consider the infinite plane E spanned by the x_1- and x_2-axis, defined by the condition $x_3 = 0$.



Your tasks are the following (**Note: (a) and (b) can be solved independently of each other, i.e. you may use the result for $I(r)$ given in (a) in order to start with the second part**):

a) Let $\mathbf{x} \in E$ be an infinitesimal patch with distance $r \geq 0$ to the origin. Show that the irradiance $I(r)$ at this point is given by

$$I(r) \;=\; \frac{\Phi_S \cos(\theta)}{4\pi(r^2 + d^2)},$$

where θ is the angle between the plane normal $(0, 0, 1)$ and the direction from $\mathbf{x}$ to $\mathbf{d}$.

b) Compute the radiant power Φ_E received by E by integrating the irradiance over E. The surface integral in polar coordinates is

$$\Phi_E = \int_0^{2\pi} \int_0^{\infty} I(r) r \; dr \; d\phi.$$

Does the result surprise you?

4.2 Analytical solution of the rendering equation in 2D (written, 40 Points)

The Figure 1 shows a simple 2D scene with a linear light source L of uniform radiance 1 for each point and direction. Assume that the light source absorbs all light hitting it. Located at the $y = 0$ line is a Lambertian material with the following BRDF:

$$f_r(\omega, (p, 0), \omega_o) = \frac{1}{2}$$

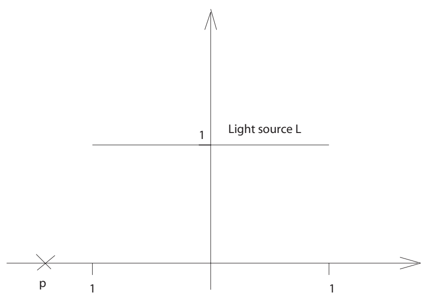


Figure 1: The linear light source reaches from -1 to 1 at y-position 1 with a uniform radiance of 1 for each point on the light source and each direction.

a) The standard rendering equation in 2D is given by:

$$L(x, \omega_o) = L_e(x, \omega_o) + \int_0^{\pi} f_r(\omega, x, \omega_o) \cdot L(x, \omega) \cdot cos\phi \cdot d\omega$$

Solve the rendering equation analytically for each point $x = (p, 0)$ and direction ω_o. As the rendering equation shows, you have to integrate over the hemicircle for each point x.

b) Let S be the set of all points on the light source (the $y = 0$ plane can be ignored here) then the point form of the rendering equation in 2D is given by:

$$L(x, \omega_o) = L_e(x, \omega_o) + \int_{y \in S} f_r(\omega_i, x, \omega_o) \cdot L(y, -\omega_i(x, y)) \cdot \frac{cos\phi_i cos\phi_y}{|x - y|} \cdot dy_x$$

Note that the denominator $|x - y|$ is really the distance not the squared distance as in the 3D version. Again solve the equation analytically, but now by integrating over the light source.

Hint: A computer algebra system such as Maple can be used to solve especially the second part of the exercise. Integration is performed by typing

```
int(sin(Pi*x),x=-2 ...  1);
```

for instance to get the integral of the function $sin(\pi x)$ with ranges -2 to 1.

Framework Update

Download the new framework from the course website. There are some improvements:
The 3D data you will use for rendering now contains also `.scn` files which contain window size and material parameters.
The rendering framework features a `Scene` and a `Render` struct now, which encapsulates everything that belongs to a scene respectively everything that is needed during rendering. It can load scenes consisting of `.ra2`, `.n` and `.scn` files into the `Scene` struct.
Additionally, the framework now supports High Dynamic Range (HDR) environment maps. One is supplied.
For materials, the `Scene` struct contains an array `material` which contains all the materials and an index array `mat_index` with one element per triangle. `material[mat_index[tri_id]].color_d`, for example, is the diffuse color of the triangle with id `tri_id`.
Note that you can now switch through different shaders by using the `1`(=debug), `2`(=simple) and `3`(=pathtracing) keys on the keyboard.
Last but not least, the rendering framework now supports multithreading. With the supported linux/scons setup, just call `scons openmp=1` for building. With other build tools, define the `OPENMP` macro in the code (can usually be done for the whole project with a compiler option) and enable OpenMP (usually by setting a compiler flag and linking against its library). You should use `openmp=0` for debugging and `openmp=1` to generate nice images.

Scenes

To load scenes, call `./coRT SceneName EnvmapFile`. The framework contains the following two scenes:

• **Bunny** is the already known Bunny scene which doesn't contain any light sources and should therefore be rendered with an environment map to be lightened. Load it with `./coRT Bunny pisa_oct.hdr`

• **CornellBox** is a box with some geometrical objects and a light emitting plane in it and can be rendered without environment map. Is is loaded per default or with `./coRT CornellBox`

4.3 Simple Path Tracer (30 Points)

Implement a simple `Render::shade_path(Ray &ray, HitRec &rec, int depth, int thread)` method. The method works as follows:

a) If you hit a triangle with `material[mat_index[tri_id]].color_e != Vec3(0.0f)` return this value.

b) Return `Vec3(0.0f)` if the recursion depth gets too large (e.g. 5).

c) Otherwise, compute the hit point and the (interpolated) normal.

d) Use the `Material::diffuse(...)` method to generate a new cosine distributed direction out of the normal and two random numbers (you may have to flip the normal direction to point towards the incoming ray).

e) Use the computed direction and the hit point to construct a new ray (in order to prevent self intersection start this ray from `ray.tmin = RAY_EPS`).

f) Shoot this ray.

g) If you don't hit a triangle, return `material[...].color_d * getEnvironment(...)` method and return this value.

h) Otherwise recurse: return `material[...].color_d * shade_path(ray, rec)`.

Hint I: The `thread` parameter of `Render::shade_path(...)` is there because every thread has its own random number generator (to avoid synchronization issues). You may obtain a random number in $(0,1)$ with `mtrand[thread]->rand()`.
Hint II: The multiplications in **g,h)** ought to be elementwise (`Vec3::product`) and not dot products.