Computer Graphics II: TP1

Enseignants :

Nicolas Holzschuch

- Back to the lecture page

TP1: Material models

- Source code
- GLSL specification
- OpenGL 4.4 API Quick Reference Card
- If you didn't follow last year's "Introduction to Computer Graphics" course, a short reminder on GLSL programing.

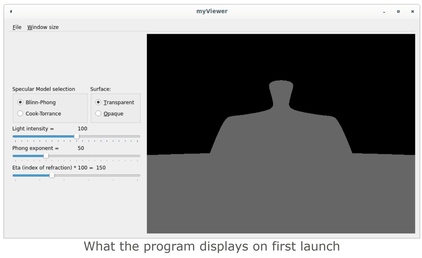
Getting started

First, clone the git repository.

Change working directory to `BVH_TP`. Create the Makefiles with: `qmake -qt5`, then compile the project with `make`. You can also use `qtcreator` if you prefere IDEs.

Compilation should produce a few warnings but no errors. Once it is done, start the program with `./viewer/myViewer`. You should get the following picture. There is a single window, with the OpenGL display on the right, parameter controls on the left, and pull-down menus:

- `File` to load different models, textures or environment maps, and also save a screenshot.
- `Window size` to change the size of the window (dragging corners also works).



What the program displays on first launch

Program structure

The `BVH_TP` folder contains:

- `trimesh2/`, an auxiliary library in charge of loading 3D models and converting them into a format suitable for display. It can read files in ply, obj, 3ds, off formats, at least. You will find several models in `viewer/models`, but feel free to download others.
- `viewer/` contains the main program code. Inside this folder, you will find:
  - `src/main.cpp`, in charge of application starting and preparing menus.
  - Files for the `OpenGLWindow` class, for creating a Qt window with an OpenGL context.
  - Files for the `GlShaderWindow` class, which derives from `OpenGLWindow`. This is where the main action is. It loads the scene, loads the shaders, compiles them, sends parameters...
  - The `shaders/` directory contains all the shaders.
  - The `textures/` directory contains all the textures and environment maps. Feel free to add more textures if you want to.
  - The `models/` directory contains some 3D models to be used with the program. Feel free to add more.

For this project, everything is in the `shaders/gpgpu_fullrt.comp` compute shader. It's a working shader, but we're going to add more features.

You won't need to edit the C++ code at first, but you might need to read it to know the name of the variables used by the shader. Later, you will need to edit it if you want to add sliders or radio buttons.

Blinn-Phong model with Fresnel and Textures

The program contains a working ray-tracer, implemented using Compute Shaders. For the time being, we're not going to look at the ray-tracing part, only at the local illumination part.

The key function is `directIllumination()`. It is called by `trace()` for each pixel of the screen that covers an object of the scene (we'll cover this in the ray-tracing lecture).

First, edit this function so that it computes local illumination using the Blinn-Phong model. Make it as modular as possible, so you can reuse code in future works, but also move code aside when you change lighting model.

You already have the point position, its color and the incoming viewing direction $\vec{V}$. You will have to compute the shading normal $\vec{n}$ and the vector to the light source $\vec{L}$.

Once you have these, you can compute:

- **Ambient lighting:**

  $$C_a = k_a \times C \times I$$

  where k_a is the ambient reflection coefficient and I is the light intensity.

- **Diffuse lighting:**

  $$C_d = k_d \times C \times \max(\vec{n} \cdot \vec{L}, 0) \times I$$

  with k_d the diffuse reflection coefficient.

- **Specular lighting:**

  $$C_s = F(\theta_d) \times C \times \max(\vec{n} \cdot \vec{H}, 0)^s \times I$$

  with the half-vector $\vec{H}$, defined as the normalized sum of $\vec{V}$ and $\vec{L}$:

  $$\vec{H} = \frac{\vec{V} + \vec{L}}{\|\vec{V} + \vec{L}\|}$$

  and where θ_d is the angle between $\vec{H}$ and $\vec{L}$. Remember that you should **never** use inverse trigonometric functions.

  F is the Fresnel coefficient, which depends from the material index of refraction, η:

  $$\begin{aligned} F(\theta) &= \frac{F_s + F_p}{2} \\ c_i(\theta) &= \left(\eta^2 - \sin^2\theta\right)^{\frac{1}{2}} \\ F_s(\theta) &= \left|\frac{\cos\theta - c_i}{\cos\theta + c_i}\right|^2 \\ F_p(\theta) &= \left|\frac{\eta^2\cos\theta - c_i}{\eta^2\cos\theta + c_i}\right|^2 \end{aligned}$$

  As a first step, work with a real η. In a second step, make it a complex number, with an imaginary component.

The colour returned by `directIllumination()` is $C_a + C_d + C_s$.

Cook-Torrance model

Once you have the Blinn-Phong model, extend it (in a modular way) so your program uses the Cook-Torrance model:

$$C_s = \frac{F(\theta_d) D(\theta_h) G_1(\theta_i) G_1(\theta_o)}{4 \cos\theta_i \cos\theta_o}$$

where *F* is the Fresnel term you have computed before (make sure it is in an auxiliary function!). *D* is the microfacet normal distribution. We'll use the GGX distribution:

$$\begin{aligned} D(\theta) &= \frac{\chi_{[0,\pi/2[}(\theta)}{\pi \cos^4\theta} \frac{\alpha^2}{(\alpha^2 + \tan^2\theta)^2} \\ G_1(\theta) &= \frac{2}{1 + \sqrt{1 + \alpha^2 \tan^2\theta}} \end{aligned}$$

The illumination should be different from Blinn-Phong. Use the boolean `blinnPhong` to switch between Blinn-Phong and Cook-Torrance. Use the `shininess` parameter for both methods.

Again, make sure the functions are well separated. Re-use code as much as possible, and anticipate having to re-use it.

If you want to do more...

If you have all the code working, here are things you *can* do:

- Implement a color picker, to let the user select the base color for the object (using Qt elements).
- Implement the Artist Friendly Metallic Fresnel (the author provides C++ code).