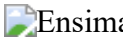


Computer Graphics II: TP2

- 
- 
- 
- 
- 
- 

Enseignants :

Nicolas Holzschuch
- Retour à la page du cours

TP2: Multiple bounces for ray-tracing

» TP1
» **TP2**
» TP3

- Spécifications GLSL
- OpenGL 4.4 API Quick Reference Card

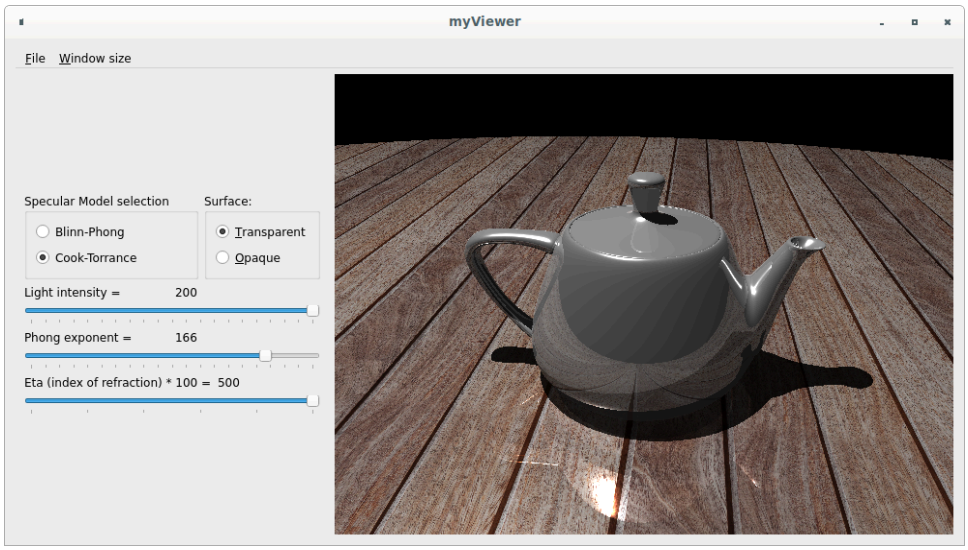
Multiple bounces with ray-tracing

We are going to extend the ray-tracer we did with practical 1 (if you haven't finished practical 1, finish it first).

Our goal is now to have multiple bounces. There are several things that are required:

- We need to compute both local and indirect illumination.
- Local illumination *must* check if the light source is visible, in order to create shadows.
- Local and indirect illumination use different directions when evaluating the BRDFs.
- Local illumination uses the full BRDF.
- Indirect illumination uses only the specular part of the BRDF, namely the Fresnel coefficients.

At the end of the practical, we should get something like this:



Our goal for this practical

Shadow rays

Our first goal is to have working shadow rays. Edit directIllumination() so that it checks whether the light source is visible (using shadow rays) before computing the full lighting model. If the light source is invisible, there is only ambient lighting.

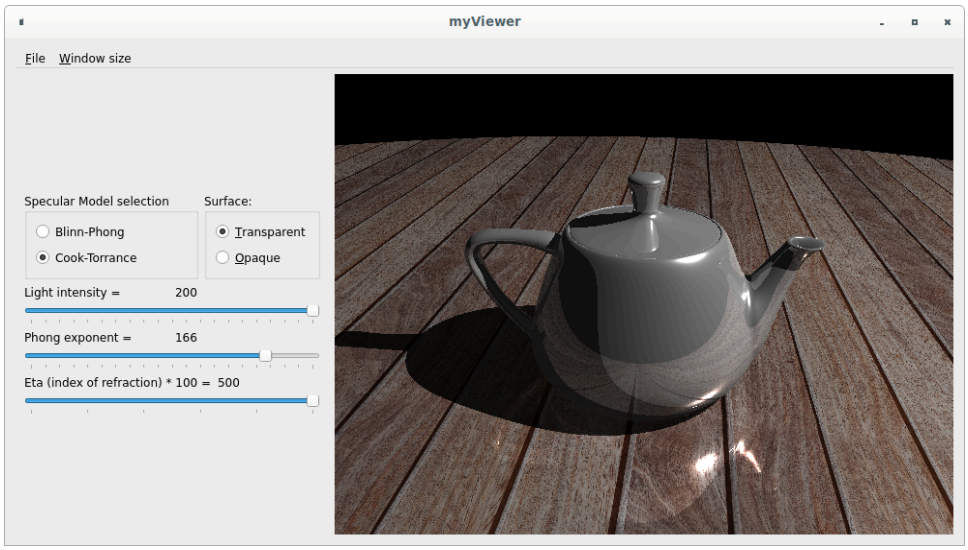
You will likely see issues with self-shadowing, especially on curved objects: the surface is made of triangles, even though it *looks* curved. This causes a staircase effect on the shadow boundary. You can fix it using a combination of:

- Check that the light source is in the positive half-space of the shading normal first. Otherwise the point is in shadow.
- Change the starting point of the shadow ray slightly to avoid self-shadowing

Indirect illumination

We are now ready for indirect lighting. Since we do not want to recurse, we are going to store ray intersections in an array, follow the rays, then compute indirect illumination.

- Define the maximum number of bounces. Allocate an array of struct trace_state_t to contain the intersections encountered by the ray (origin, direction, point hit, normal, color).
- Forward tracing: for each ray, compute the intersection, store the intersection, compute the bouncing ray, iterate until the ray does not intersect anything or until you have reached the maximum number of bounces.
- Backward color computing: you can now compute the ray color. Start by computing the color for the last point hit (the one further from the eye), using direct illumination and a shadow ray, then move down the stack. At each point, you have to compute the direct lighting using a shadow ray, and combine it with indirect lighting (what you have computed so far).
- Remember to use a different BRDF for direct and indirect lighting, since the directions are not identical.



Another view, with curved shadow on a curved surface

Contact