University of Poitiers
Master Computer Science M1
Algorithmic 3D I

TP 7. Ray tracing

The goal of this last part is to design and program, in Java, a basic ray tracing.

Preliminary step: Image recording

Get class JavaTGA.java, compile it and run it. It saves a simple image using the non-compressed image format TGA. Check how it works: A buffer must first be filled corresponding to each pixel of an image, row by row. Each colour is specified in the format B,G,R (not R,G,B). Find the two imbricated loops that aim at filling the buffer. You can base your main loop for ray tracing on these loops.

Step 1. Ray casting

Ray casting is often considered as a ray tracing approach with only primary rays (no reflection for mirror effect, no transmission for transparent object) and shadow rays.

1.1) Design a class for planes, another class for spheres. Include in each class at least one constructor to create the corresponding object (with appropriate parameters to define it, for instance a centre and a radius for a sphere). These classes should inherit from an abstract class including a general abstract method to compute the (first if multiple) intersection of any represented object with a ray (a half-line) defined by a starting point P and a direction vector $\vec{v}$:

```
double getIntersection(Vec3f P,vec3f v);
```

This method should return the parameter λ such that the intersection M, if it exists, is defined as:

$$M = P + \lambda \vec{v}$$

In case there is no intersection, a negative value (or 0, but it's more hazardous) can be returned.

1.2) Create a scene by manually "instantiating" several planes and spheres. The same frame as OpenGL's one can be used (viewer at (0;0;0), viewing in the -z direction). Define also one or more light source (by defining its position and ambient, diffuse and specular terms).

1.3) Create a method *findColor* that gives the colour received by point P in direction $-\vec{v}$ by[4]:

- Computing intersections λ_{obj} of all the objects with the ray (P, $\vec{v}$),
- Select the smallest positive value λ_{min} corresponding to the nearest intersection, and
- Compute the colour of this intersection using Phong's lighting model.

1.4) By relying on the imbricated loops given in the JavaTGA class, create the main loop for ray tracing. The main parameters must be the width w and height h of the final image and the distance D where the projection plane is placed (D can be seen as a zoom factor). A pixel *(row,col)* therefore corresponds to a 3D point with coordinates:

$$x = \frac{(col - w/2)}{h}$$
$$y = \frac{(row - h/2)}{h}$$
$$z = -D$$

Note that y gets a value between -0.5 and 0.5, while x lies in a different range taking into account the aspect ratio (w/h) of the screen[5]. This 3D point gives the direction of the ray to cast from the viewer. Provide a first version of your ray tracer: cast every primary ray and get the colour of its pixel using method *findColor*.

[4]Remember that in ray tracing, rays are cast in the opposite direction of light path. That explains why the direction of light is -v et not v.
[5] Suppose an image with ratio 16:9 (it is the case of high definition TV screen: 1920x1080). Then x lies within the range -0;89 to 0,89 instead of -0,5 to 0,5 to respect the screen ratio (such a screen is wide).

1.5) Compute shadow rays so that the Phong's formula takes into account visibility of light sources. Use again *getIntersection* of all the objects of the scene to check for any intersection. Such an intersection is placed between the light source and the point where lighting is computed. By considering a direction vector defined with these two points, intersections correspond to any value for λ between 0 and 1. There's no need to search for the nearest one. Any detected intersection concludes that the considered point is not lighted (by the considered source).

Step 2. More advanced ray tracing

Change the computation of the colour of a point by introducing a reflected ray if the material of the underlying object is reflective and/or a refracted ray if this material is transparent. Formulae to compute the direction of these rays are given in the course presentation. Transform *findColor* into a recursive method, in order to find the colour of the light coming from reflected and refracted directions. The maximal depth of this recursion can be given as a general parameter of your program and an additional parameter can be given to *findColor* to control recursion.