Ray Tracing Assignment 2:
Optical Laws, Anti-Aliasing and Texture Mapping

March 8, 2022

Computer Graphics

1 Theoretical questions (1.0 point)

The answers to the theoretical questions must be submitted as a PDF file. We recommend using LaTeX, but we allow other submissions (including handwritten ones) as well. This question is parametrised by the digits of one of your group member's student number. If your student number is $s7654321$, then $g = 7$, $f = 6$, $e = 5$, $d = 4$, $c = 3$, $b = 2$, $a = 1$. For all theoretical questions please explain your answers. If applicable, show your mathematical derivations.

1. (0.3 points) Consider the following 2D scene description; see Figure 1. The origin is at $\mathbf{O} = [0, 0]$, a point light source is at $\mathbf{S} = [6 + \frac{a}{10}, 16 + \frac{b}{10}]$ and the only geometric object in the scene is a disk with radius $r = 4$ centered at $\mathbf{C} = [8 + \frac{c}{10}, 8 + \frac{d}{10}]$. The light creates a shadow of the circle on the x-axis. Give the range $[\texttt{shadow}_a, \texttt{shadow}_b]$ where the x-axis is in shadow, i.e. where it does not receive light (see also Theoretical question 3. of the RT1 assignment).

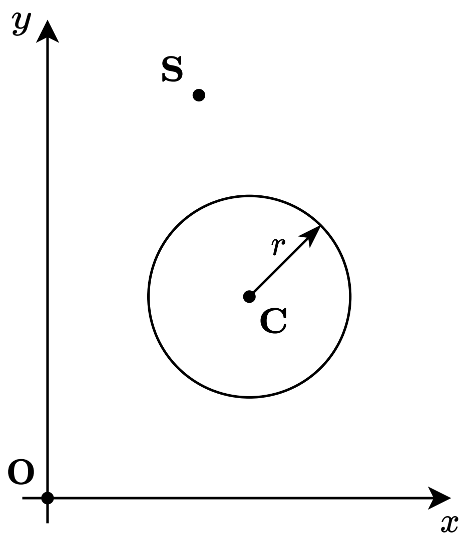


Figure 1: *A 2D scene with point light source* $\mathbf{S}$ *and a disk with radius* r *centered at* $\mathbf{C}$.

2. (0.5 points) Consider the following 2D scene description; see Figure 2. The origin is at $\mathbf{O} = [0, 0]$, the eye/camera is at $\mathbf{E} = [4 + \frac{a}{10}, 8 + \frac{b}{10}]$ and the only geometric object in the scene is a rectangle with width $w = 12$ and height $h = 4$ located (its top-left corner) at $\mathbf{C} = [14 + \frac{c}{10}, 16 + \frac{d}{10}]$. Assume that the camera is looking at the x-axis, which in this scenario is a perfect mirror. Give the range $[\texttt{ref}_a, \texttt{ref}_b]$ on the x-axis where you can see the rectangle in the mirror.

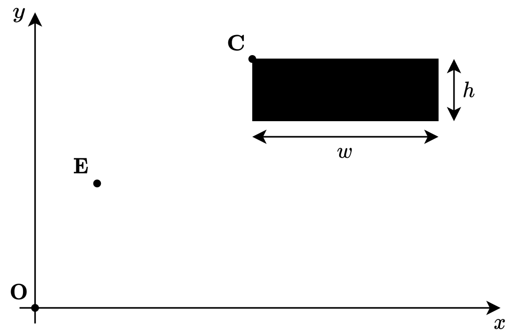


Figure 2: *A 2D scene with eye/camera* $\boldsymbol{E}$ *and a rectangle with width* w *and height* h *located (its top-left corner) at* $\boldsymbol{C}$*.*

3. (0.2 points) Consider the following 2D scene description; see Figure 3. Assume that the x-axis is the boundary between two isotropic media, where the medium above the x-axis (white) has refractive index $n_i = 1 + \frac{a}{10}$ and the medium below the x-axis (blue) has refractive index n_t. The origin is at $\mathbf{O} = [0, 0]$, the eye/camera is at $\mathbf{E} = [4 + \frac{b}{10}, 6 + \frac{c}{10}]$, the point where the view ray (cast from the camera) is refracted by the blue medium is at $\mathbf{P} = [12 + \frac{d}{10}, 0]$, and the refracted ray ends up at $\mathbf{Q} = [16 + \frac{e}{10}, -8 + \frac{f}{10}]$. Use Snell's law to find the refractive index n_t.

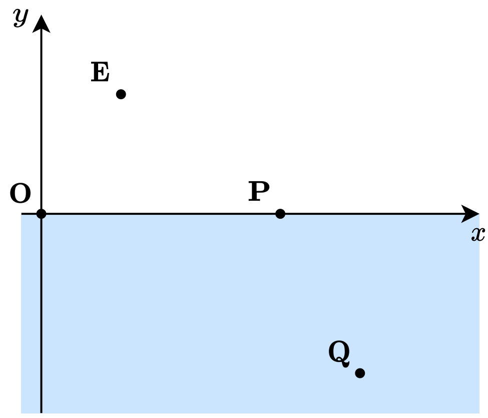


Figure 3: *A 2D scene with two isotropic media separated by the* x*-axis, eye/camera* $\boldsymbol{E}$*, point of refraction* $\boldsymbol{P}$ *and final ray position (after refraction)* $\boldsymbol{Q}$*.*

2 General instructions

In the following exercises you will be asked to repeatedly add new functionality to an existing ray tracing application. The ray tracer should always be able to render the scene files from previous (sub-)assignments and produce the correct images. In other words, adding new functionality should not break the existing functionality.

If you work with a lab partner, then both of you will need to **be in a group on Themis before submitting**. In order to do so,

- visit https://themis.housing.rug.nl,
- go to *Courses* → *2021–2022* → *Computer Graphics*,
- have one student create a group and the other one join that group.

After completing each sub-assignment, create a **zip** file containing:

- `CMakeLists.txt`,
- the `Code` folder.

Do **not** include any other folders, executables, a README file or rendered images.
Submit the created zip file to Themis.

The assignments have to be completed in order, i.e., for each sub-assignment it is a precondition that all features preceding it have been implemented. In total, 11 points can be obtained.

The previous framework cannot be used for this assignment. Please download the framework corresponding to this assignment from Nestor. The framework contains an implementation of the Phong illumination model, the `Sphere` shape and the `Quad` shape.

3 Optical laws

In this assignment you will implement a global lighting model by extending the framework with support for shadows, reflection and refraction.

3.1 Shadows (2 points)

The new framework is able to read a new (optional) parameter from a scene file: `Shadows`. This is a boolean value which it stores in the `Scene` class as `renderShadows`.

Modify the lighting calculation in the `Scene::trace` function to render shadows, depending on whether `renderShadows` is `true` or `false`.

The general approach for detecting shadows is to test whether a ray from the hit point to the light source intersects other objects. Only when this is *not* the case, the light source contributes to the lighting.
Use the provided `Scene::castRay` function to cast a shadow ray. The two elements of the returned pair can be accessed using `.first` and `.second`. The first element is either a smart pointer to the hit object, or `nullptr` in case no intersection was found.

To avoid shadow acne, use the method described in the *Illumination and Shading* lecture slides. In this implementation, use the `epsilon` value provided in `scene.h`, i.e. 10^{-3}. Make sure to take the direction of the normal into account.

Figure 4 shows a sphere casting a shadow on a quadrilateral.

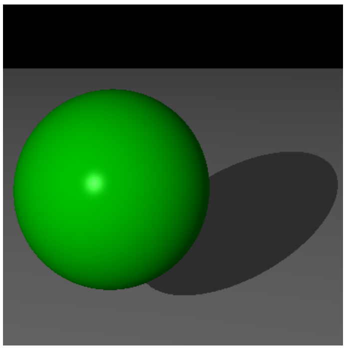


Figure 4: *The shadow of a sphere on a quadrilateral*

3.2 Specular reflection (2 points)

In the Phong illumination model, the specular component models the shininess of the object. On the surface of the object where specular highlights can be seen, the surface acts in a mirror-like fashion. This behavior can be modeled by adding support for specular reflection.

In our ray tracer, an object's material is either opaque (default) or transparent, and these two cases are handled differently. In this section, we will add specular reflections to opaque objects. Transparent objects are covered in Section 3.3.

To implement specular reflection, compute the color of the reflection as follows and add it to the `Color` as already computed according to the Phong illumination model.

1. Note how the `Scene::trace` function in the framework already distinguishes between transparent and opaque objects. Make sure the functionality to be added only applies to the latter.

2. After hitting the object, the ray should reflect off the object as if it where a (perfect) mirror. Compute the direction in which the ray is reflected.

3. Recursively trace a new ray in this direction. To avoid finding an intersection with the current object due to floating point inaccuracies, use the same method and epsilon (given in `scene.h`) used to avoid shadow acne.
   A potential problem is that the recursive ray tracing call may never terminate, for example in the case where the ray's origin is inside of a sphere. To resolve this, the `Scene::trace` function has been extended with a second parameter for the recursion depth. When calling `Scene::trace` recursively, the depth passed to it should be the current depth minus 1. If the recursion depth is defined in the scene file, it is read from there, otherwise it defaults to 0.

4. Multiply color returned by the call in the previous step by the specular coefficient of the current object. This models that the hit object is not a perfect mirror: some of the light is absorbed by the surface.

Figure 5 shows an example of specular reflections.

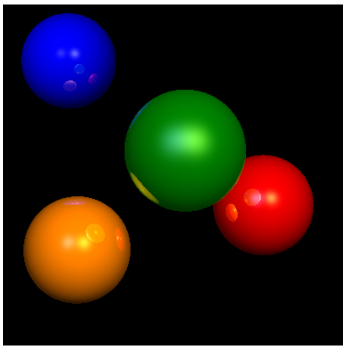


Figure 5: *Specular reflections on spheres*

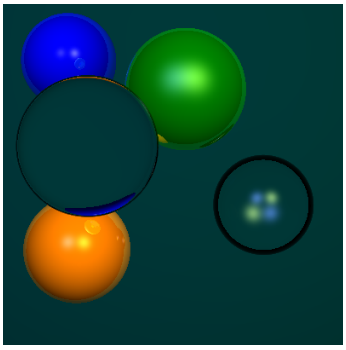


Figure 6: *Opaque and transparent spheres*

3.3 Transparent objects (2.0 points)

In addition to having a color of their own, transparent objects reflect and refract light.

In the framework, an object is interpreted to be transparent when its material in the scene file contains the attribute `nt`: the refractive index n_t for this object. If this attribute is present in the scene file, it is stored in the `nt` data member of the `Material` class and the boolean data member `isTransparent` is set to `true`. The framework already distinguishes between transparent and opaque objects.

Implement reflection and refraction for transparent objects as described by the *Illumination and Shading* lecture slides (make sure you have the latest version). Use Schlick's approximation to compute the reflectivity, and thus the ratio between reflected and refracted light. You may assume that

- only closed objects can be transparent,
- closed objects return normals that point outwards,
- the refractive index n on the outside of the objects is 1.0,
- the objects in the scene do not intersect.

To avoid finding an intersection with the current object when casting a secondary ray, use the same method and epsilon (given in `scene.h`) used to avoid shadow acne. Also ensure that the normal points in the correct direction for this.

Figure 6 shows an example of transparent objects reflecting and refracting light.

4 Anti-aliasing

Ray tracing can be regarded as sampling a geometrically described scene. As with all sampling, aliasing effects can occur, which manifest in the image as jagged edges. There are a number of anti-aliasing techniques to mitigate these effects, one of them being supersampling.

4.1 Super-sampling (1.5 points)

Supersampling is a method of anti-aliasing in which the image is rendered at a higher resolution and then downsampled.
Modify the `Scene::render` function to support supersampling by casting multiple rays for each pixel and averaging the resulting colors. Note that the rays should go through the center of each (sub)pixel. This is shown in Figure 7 for 1×1 supersampling on the left, which is identical to regular sampling, and 2×2 supersampling on the right.

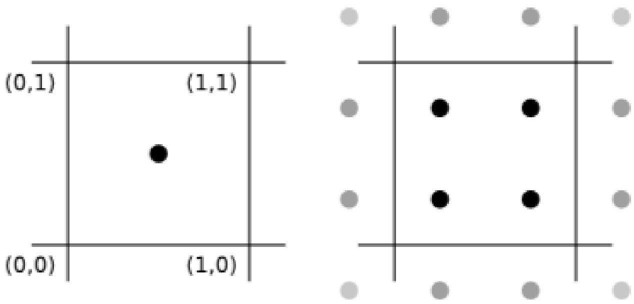


Figure 7: *Supersampling*

The supersampling factor is read from the scene file (the `SuperSamplingFactor` attribute) and stored in the data member `supersamplingFactor` in `Scene.h`. When the supersampling factor is not defined in the scene file, its value defaults to 1.
An example of 4×4 super-sampling is shown in Figure 8 (`scene01-ss.json`).

5 Texture mapping

This section covers texture mapping for spheres.

5.1 Fixed texture (1.5 points)

Observe how the `Material` class has been extended with the new data members `bool hasTexture` and `Image image`. Both variables are read from the scene file, if present. The `Sphere` class has been extended with an axis and an angle. These are only relevant for Section 5.2.

Implement texture mapping for spheres as follows.

1. Complete the `Sphere::toUV` function by mapping the x,y,z-coordinates of the hit to u,v-coordinates for use with a texture. Use the mapping as defined in the *Textures* lecture slides. Do **not** use the definition in the text book, as it produces different results.

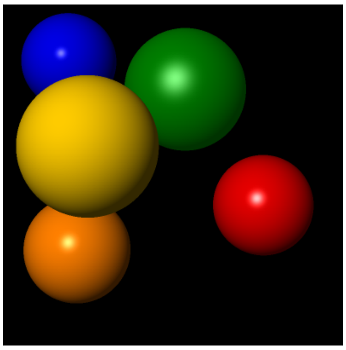


Figure 8: *An example of supersampling*

2. Modify the `Scene::trace` function. If the hit object has a texture, then use the `Object`'s function `toUV` to obtain the u,v-coordinates corresponding to the hit and obtain the color from the the `Object`'s texture as such:
   `material.texture.colorAt(u, 1.0 - v)` where $u, v \in [0, 1]$.

   Note: In a PNG file, the origin $(0, 0)$ is the top-left corner, whereas the origin for the obtained texture coordinates is the bottom-left corner. This explains why the v-coordinate has to be inverted.
   If the object does not have a texture, then use the material color as before.

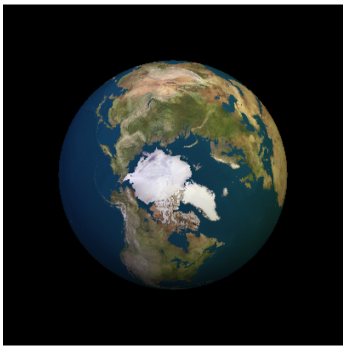


Figure 9: *The Earth texture, mapped to a sphere without rotation*

For testing your texture mapping code you may want to use `bluegrid.png` instead of a larger and more complex image.

5.2 (Bonus) Rotated texture (1 point)

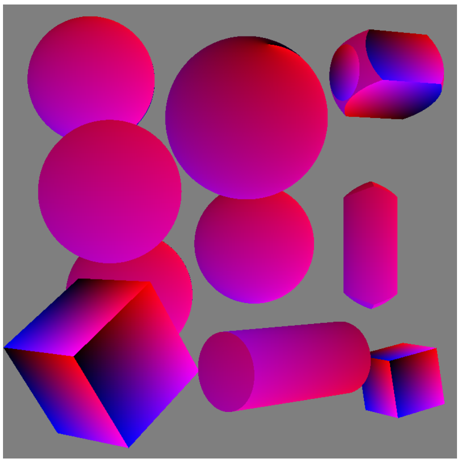
The `Sphere` class has been extended to store a rotation as defined by an axis and an angle. Both are read from the scene file, if these attributes are defined there.
In `sphere.cpp`, implement rotation for this shape. The sphere should rotate `angle` *degrees* counterclockwise around the axis given by `axis`. The axis is given as a (possibly not normalized) vector.

Hint: Consider using quaternions.

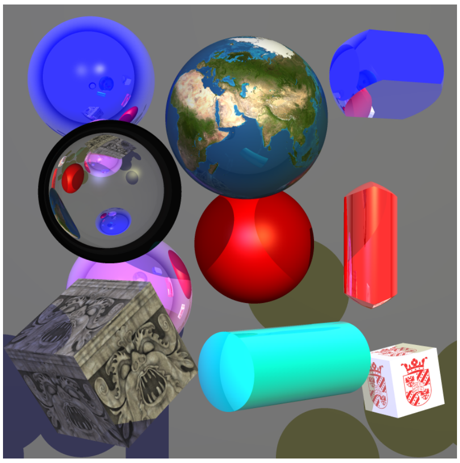
Figure 10: *The Earth texture, mapped to a rotated sphere*

6 Ideas for the competition

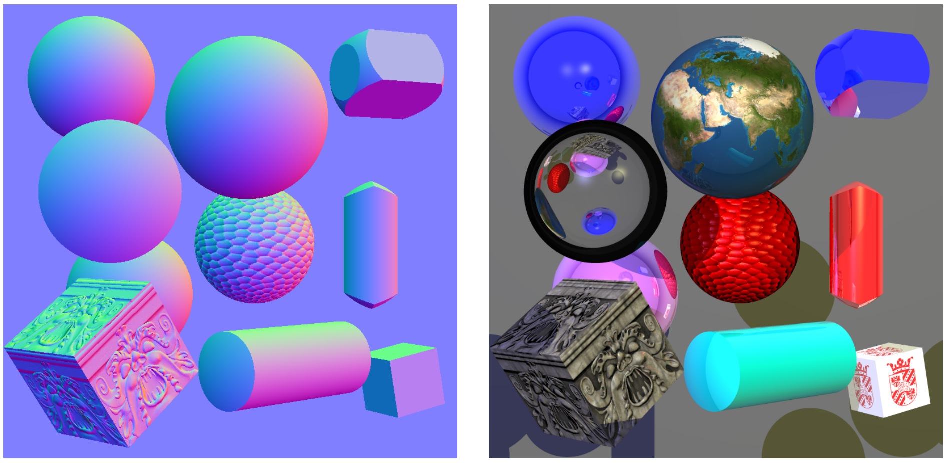
- You'll notice that the specular reflections from the scene are not blurred, like the light sources. One way to do something about this is to sample along multiple rays (around the reflection vector) and average the results. Note that for a correct result you should be careful about selecting your vectors and/or the way you average them. Hint: if you take a normal average you should select more rays in those areas where the specular coefficient is high.
- Implement normal/bump-mapping. Various examples of this technique are shown in Figure 11.



Texture coordinate buffer



Textured objects



Normal buffer   Textured objects

Figure 11: *Normal/bump maps*