Coursework I: Ray-tracing

COMP0027 Team
Tobias Ritschel, Michael Fischer, Pradyumna (Preddy) Reddy, David Walton

October 12, 2022

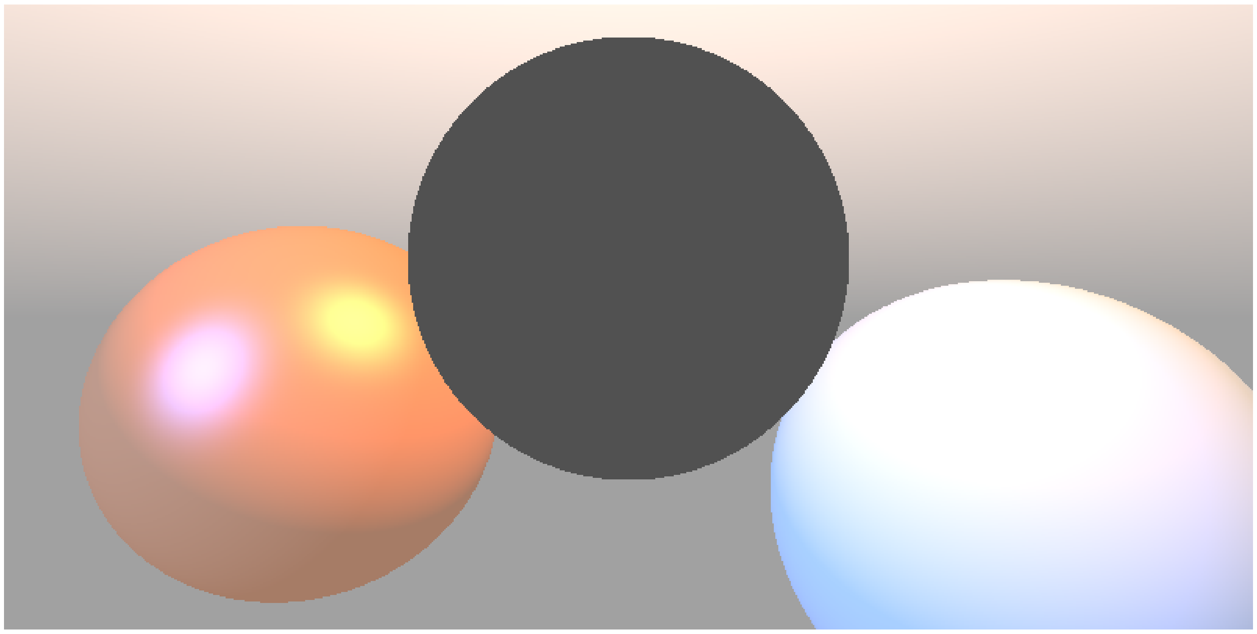
We have shown you the framework for solving the coursework at `https://uclcg.github.io/uclcg/`.

You should start by extending the respective example there. The programming language is WebGL (`https://www.khronos.org/registry/webgl/specs/latest/1.0/`) and the OpenGL ES Shading Language (GLSL) `www.khronos.org/files/opengles_shading_language.pdf`. This should run in any browser, but we formerly experienced problems with Safari and thus recommend using a different browser such as Chrome. Do not write any answers in any other programming language, in paper, or pseudo code. Do not write code outside the `#define` blocks.

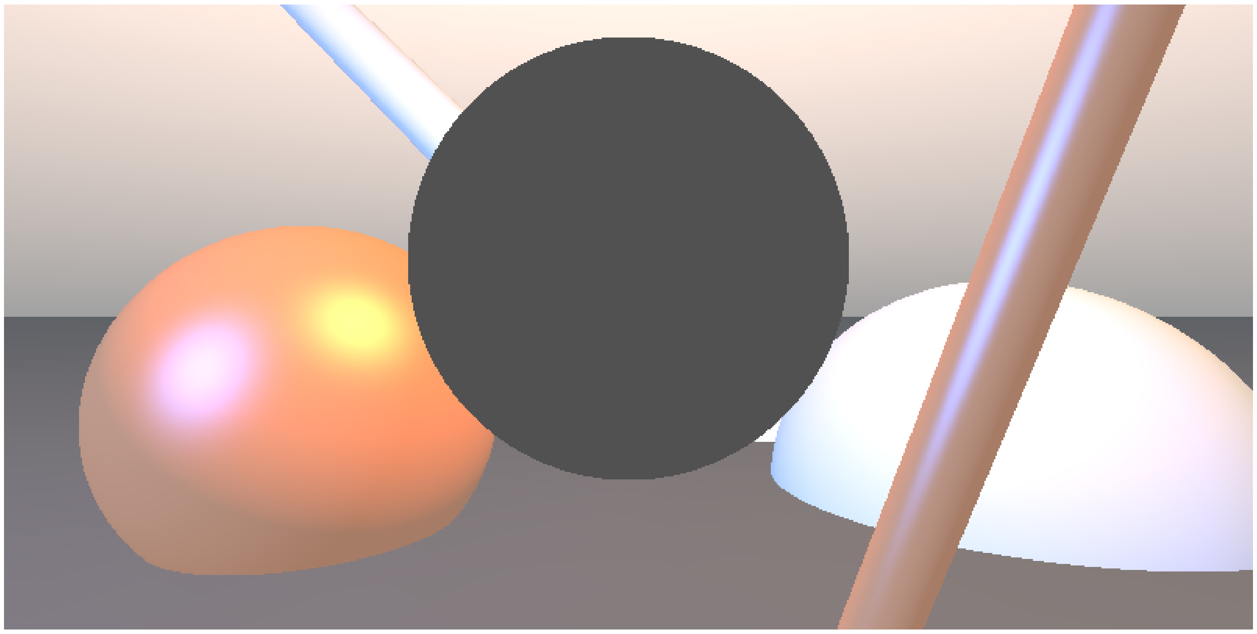
Remember to save your solution often enough to a `.uclcg` file. In the end, hand in that file via Moodle.

The total points for this exercise is **100**.

Please refer to Moodle for the due dates.

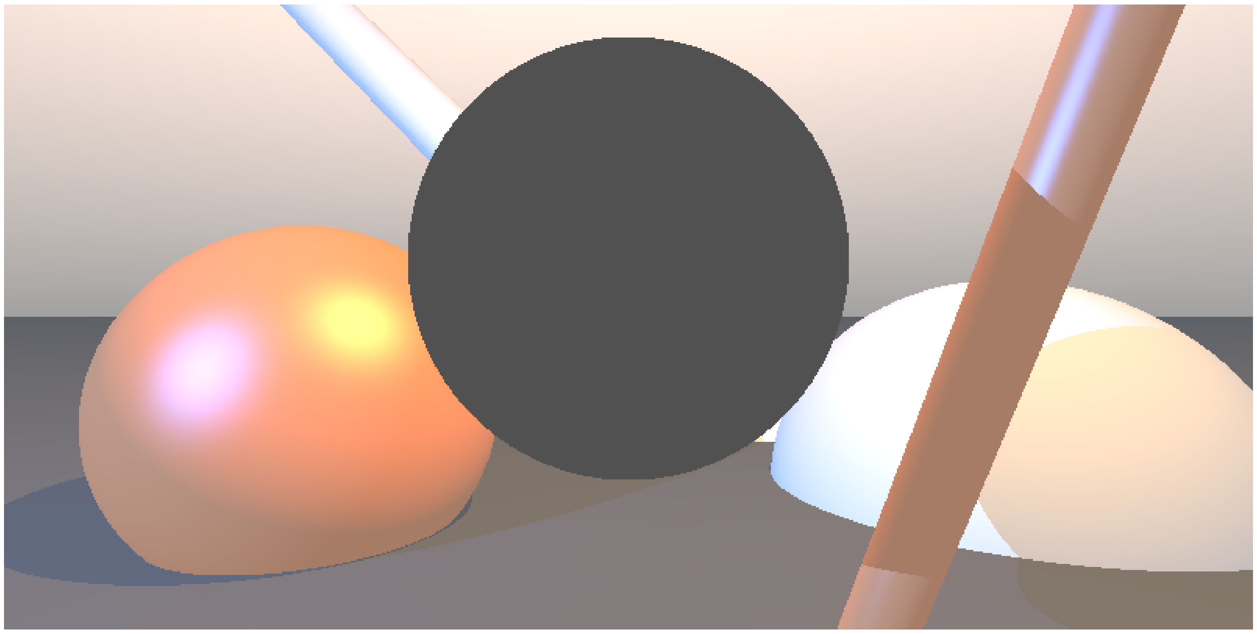

Above, the scene from this coursework, rendered with the simple initial ray-tracer you are asked to extend.

1 New primitives: Plane and cylinder (20 points)



We have added definitions of planes (a normal **n** and a distance r to the origin) and cylinders (an orientation **o** and a radius r) to the scene. You are asked to code the ray-plane (**5 points**) and ray-cylinder (**10 points**) intersections, and explain how they work (**5 points**). Pay attention on how `Sphere` is intersected. Stick to the same function signature for `Plane` and `Cylinder` that contains `HitInfo`, including normals and material.

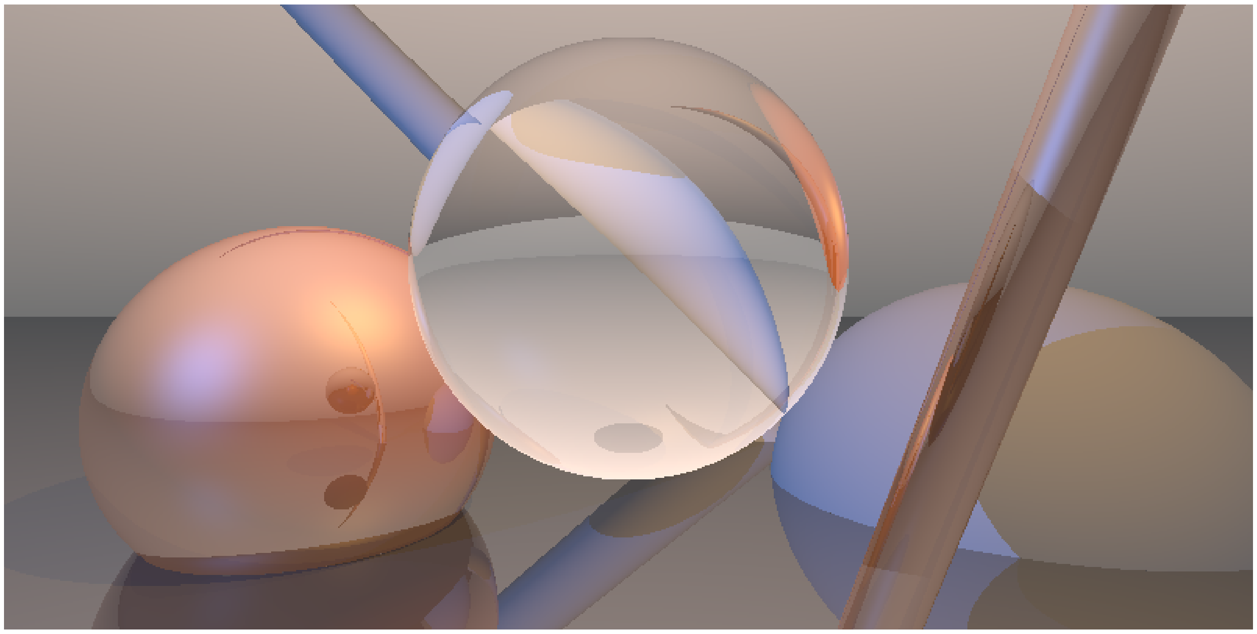
2 Casting shadows (10 points)



We have discussed how shadows, reflections and refractions work and how recursions can be unrolled into loops under some conditions. The framework contains the skeleton of such a traversal, but without the code for shadows, reflection and refraction.

First, add shadow tests to the shading (**10 points**).

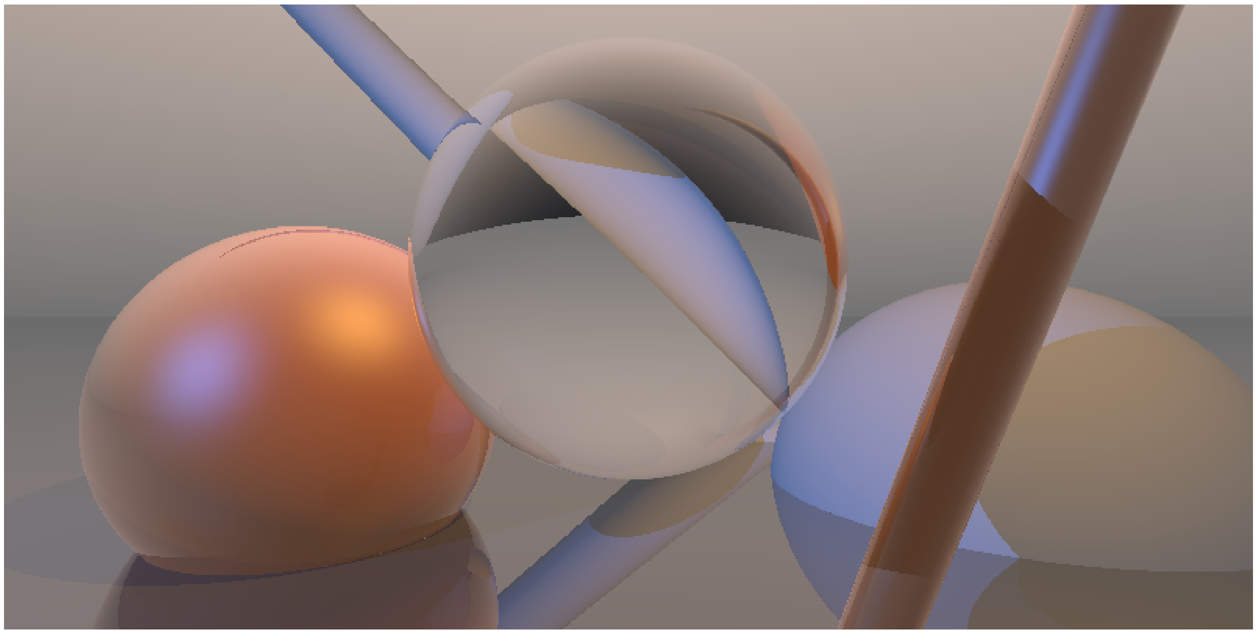
3 Adding reflections and refractions (24 points)



The code already contains the loop to perform the ray traversal iteratively, what remains to be added is code for computing the reflection direction (**12 points**) and refraction direction. Refraction direction is subdivided into two tasks: 1) the main code flow for bouncing rays and including the refractive constant (*10 points*) and 2) using the `enteringPrimitive` flag properly for keeping track of IORs (*2 points*) for a total of (**12 points**).

Note how objects far away enough behind the glass sphere appear mirrored. Although not shown here, objects that are behind and near the glass sphere would just appear distorted.

4 Fresnel **(10 points)**



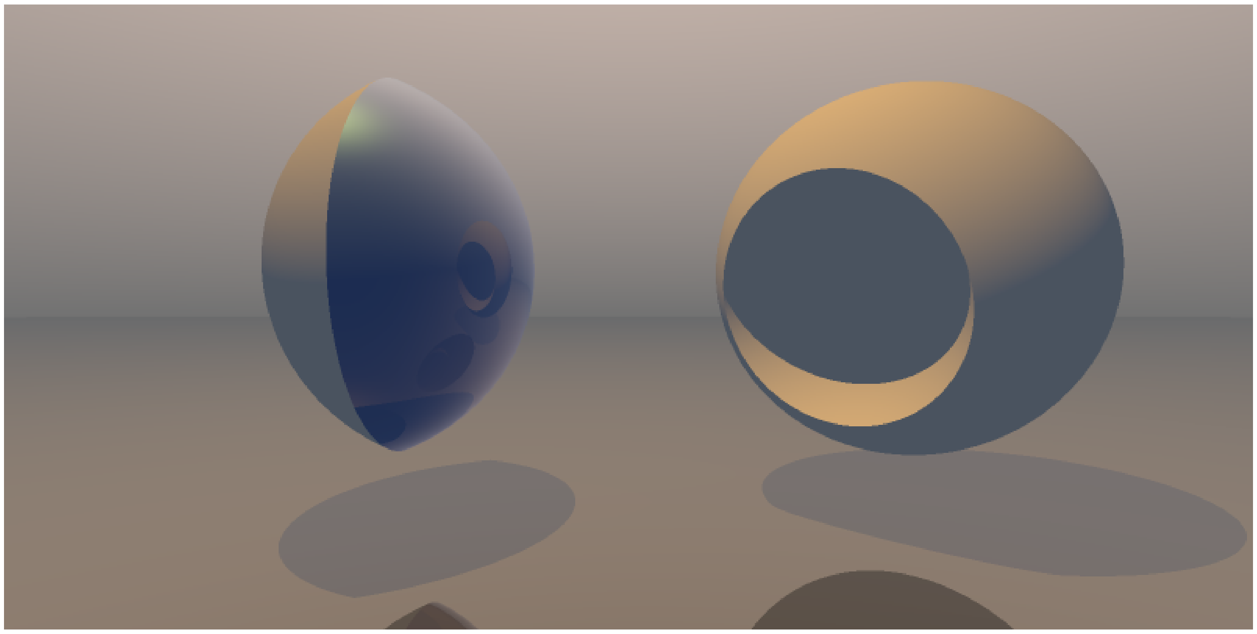
The image with reflection and refraction looks okay, but the glass likely does not look like glass as it shows reflection and refraction at equal strength all over the sphere. Such an image is shown above. Here, and in reality, reflection and refraction strength vary so that w_{reflect} and w_{refract} sums to one or less, so here we simply assume $w_{\text{reflect}} = 1 - w_{\text{refract}}$. This weighting depends on the view direction and the normal at the hit point. The reflection is typically strong on grazing angles close to the edge of the sphere, while the refraction is strongest in the centre. Do some research to find out how those weights could be computed and tell us what you used (**5 points**) and implement it (**5 points**) in the function `fresnel`. A simple implementation such as using a single dot product or the approximation by Schlick could be a good starting point.

5 Boolean Moon over UCL city **(26 points)**

We will now implement an advanced primitive: A "boolean" that forms the combination of two shapes A and B. The combination can either be the logical `and`, or the `minus`. When implementing this, set `soloBoolean` to `true` to have a clean sheet.

For the logical `and`, the shape is defined as the set of all points that are inside both A and B (depicted on the left side). Implement the intersection code for this in `intersectBoolean` (**10 points**).

For `minus`, the shape is defined as the set of all points that are on B but not in A plus the points that are on A and inside B, i.e., $B - A$ (right side of the figure). Change `intersectBoolean` to handle both types of booleans and flip the type to `minus` to test (**13 points**).



You are not allowed to change code outside defines. Add a comment to `intersectBoolean`, explaining how you would change the `HitInfo` structure to be more useful for boolean ops and why (**3 points**).

6 Reality check (10 points)

Ray-tracing is a model of what happens to light in the real world. The next two tasks ask you to reflect on that relation. You are asked to provide both of the below answers as as single-page, properly formatted (preferably TeX-typeset) PDF document with clear images (**2 points**).

Please provide example images (**2 points**) and the parameters $k_\mathrm{d}, k_\mathrm{s}$ and n (RGB diffuse, RGB specular gloss) (**2 points**) of three materials that are markedly different and can be rendered using the Phong BRDF model in our ray-tracer.

Now that you implemented and understood all the above, provide two photos of light transport or materials that cannot be realized with the code (not the geometry, that is limited to planes/spheres/cylinders/booleans) above (**2 points**). Finally, explain in two sentences, each, what prevents you from rendering those two (**2 points**).