Worksheet 5

One of the key strengths of computer graphics techniques is that they work in general. Ray tracing is, for example, well known in numerous fields of research. The distinguishing feature of ray tracers in computer graphics is their ability to efficiently handle nearly arbitrary scenes. This set of exercises helps you load and efficiently render large triangle meshes.

Learning Objectives

- Implement ray tracing of a triangle mesh (indexed face set).
- Interpolate normals across a triangle.
- Load and render a multi-material triangle mesh.
- Render with a light source specified by triangles in a mesh.

Triangle Meshes

A common way to deal with arbitrary surfaces is using triangles. In particular, we often work with an indexed face set. This is a collection of vertices with associated positions, normals, and texture coordinates. Each vertex has an index and each face in the face set (triangle in the mesh) is defined by a vector of three indices. One way to store an indexed face set on disk is using a Wavefront OBJ file.[1] This is the file format we will use. Most 3D modelling software packages can export to this format, and we will make a couple of the classic computer graphics scenes available on DTU Learn in this format (the Stanford bunny, the Utah teapot, and the Cornell box).

1. To get started, draw the triangle in the default scene using an indexed face set. Use the library file `OBJParser.js` available on DTU Learn to load the triangle from the file `triangle.obj` and upload it to the GPU in two storage buffers: one for vertex positions (array of `vec3f`) and one for face indices (array of `vec3u`). Modify your `intersect_triangle` function to take a face index as argument instead of an array of three vertex positions and make sure that your `intersect_scene` function loops over all faces in the indexed face set and uses this modified version of the `intersect_triangle` function.

2. Switch to a scene with the Utah teapot instead of the triangle and the sphere. Set the resolution of your canvas to 800×450, remove your shader selections for the sphere, and change your `intersect_scene` function to exclude the sphere but loop over all triangles in the indexed face set. Modify the camera to

eye point	look-at point	up vector	camera constant
(0.15, 1.5, 10.0)	(0.15, 1.5, 0.0)	(0.0, 1.0, 0.0)	2.5

   Load the Utah teapot (`teapot.obj`). You can set the color of the teapot to a constant `vec3f(0.9)`. Implement a function for sampling a directional light instead of a point light. Illuminate the teapot by a directional light with the direction `normalize(vec3f(-1.0))` and emitted radiance $L_e = (\pi, \pi, \pi)$.

3. The OBJ parser also loads vertex normals. Upload the vertex normals to the GPU in a storage buffer and use the barycentric coordinates computed in your ray-triangle intersection test to obtain interpolated vertex normals as a better normal for shading of a smooth surface.

4. We will now again change the scene. Set the canvas resolution back to 512×512, remove the textured plane, and load the Cornell box with blocks (`CornellBoxWithBlocks.obj`). This scene is measured in millimetres and thus requires a different camera configuration:

eye point	look-at point	up vector	camera constant
(277.0, 275.0, -570.0)	(277.0, 275.0, 0.0)	(0.0, 1.0, 0.0)	1.0

[1] http://paulbourke.net/dataformats/obj/

The Cornell box has triangles with different materials. Each triangle has a material index. Make a struct containing material information in WGSL. To begin with, we are only interested in color and emission (both RGB[A] vectors). We will consider color the diffuse reflectance (`hit.diffuse`) and replace ambient (`hit.ambient`) everywhere in the ray tracer with emission (`hit.emission`). The data you need is loaded by the `OBJParser.js` library and made available in a JavaScript object (see `obj.mat_indices` and `obj.materials`). Make another index buffer, but this time for material indices with only one `u32` per triangle. Make a storage buffer for holding the materials. Extract the color and emission data from the loaded object and insert it into your storage buffer for the materials. Use the sum of color and emission for flat shading of the Cornell box.

5. The final task is to also upload indices pointing out triangles that are light sources (because they have non-zero emission), and then shade the scene using a point light representing these emissive triangles. The list of emissive triangles is called `light_indices` in the JavaScript object containing drawing info. Use radiometric considerations to shade the Cornell box appropriately based on the area, orientation, and position of the emissive triangles relative to the point of interest hit by a ray.

Reading Material

The curriculum for Worksheet 5 is (16 pages)

- **B** Sections 12.1–12.1.2. *Triangle Meshes*.
- **B** Sections 2.8–2.9. *Linear Interpolation* and *Triangles*.
- **B** Sections 14.7.1–14.8. *BRDF* and *Transport Equation*.