Worksheet 6

We would like to be able to work with scenes of higher complexity. A million pixels, a million triangles, and several different materials is a common scenario. For such scenes, brute force ray tracing is infeasible: looping through 10^{12} ray-triangle intersections is too demanding a task. Fortunately, we can use a spatial data structure to accelerate the ray tracing.

Learning Objectives

- Accelerate rendering techniques using spatial data structures.
- Use a BSP tree for efficient space subdivision.
- Algorithm and data format adaptation for the GPU.

Space Partitioning

Ray tracing is slow if we have no spatial data structure for acceleration of the intersection tests. The following exercises are about using an axis-aligned binary space partitioning (BSP) tree for space subdivision.

1. Use an axis-aligned BSP tree to find the closest intersection of a ray and a triangle mesh. Investigate the implementation of this spatial data structure in `BspTree.js`. Explain how it works and use the BSP tree instead of the simple looping over all triangles from Worksheet 5. You may hit the WebGPU limit of at most 8 storage buffers per shader stage. If this is a problem, comment out your area light shading and exploit that triangle face indices can be a `vec4u` with the material index of the face as the fourth index (the w-coordinate). Show that you can render a teapot, a bunny and perhaps even a larger object when using the BSP tree. The table below provides a view for the bunny.

eye point	look-at point	up vector	camera constant
(-0.02, 0.11, 0.6)	(-0.02, 0.11, 0.0)	(0.0, 1.0, 0.0)	3.5

2. Because some data in our triangle mesh is per vertex and some is per triangle (face), we can interleave some buffers and reduce the number of storage buffers to stay below the limit of 8. The files `BspTree_interleaved.js` and `OBJParser_interleaved.js` can help you do this. If you did not do this in the previous task, exploit that triangle face indices can be a `vec4u` with the material index of the face as the fourth index (the w-coordinate). This should help you remove a storage buffer. Construct a small `struct` for vertex attributes and use it to contain interleaved vertex positions and normals. In this way, you can remove another storage buffer. With these modifications, reintroduce your arealight shading and render the Cornell box with blocks using the BSP tree.

3. Load the Cornell box without blocks (`CornellBox.obj`). Use your `intersect_scene` function to insert a mirror sphere and a glass sphere in the Cornell box. Size, position, and material of the spheres are provided in the table below. Render this scene using the BSP tree and include shadows, distant area light illumination, and an option to do anti-aliasing by jitter sampling.

object	center	radius	material	refractive index
left sphere	$(420.0, 90.0, 370.0)$	90.0	mirror	
right sphere	$(130.0, 90.0, 250.0)$	90.0	glass	1.5

Reading Material

The curriculum for Worksheet 6 is (12 pages)

B Section 12.3. *Spatial Data Structures.*

Supplementary reading material:

- Ize, T., Wald, I., and Parker, S. G. Ray tracing with the BSP tree. In *Proceedings of Symposium on Interactive Ray Tracing*, pp. 159–166. 2008.
- Kammaje, R. P., and Mora, B. A study of restricted BSP trees for ray tracing. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing (RT '07)*, pp. 55–62, October 2007.
- Sung, K., and Shirley, P. Ray tracing with the BSP tree. In D. Kirk, editor, *Graphics Gems III*, Chapter VI.1, pp. 271–274, Academic Press, 1995.