CSE168: Rendering Algorithms
Assignment 2
Due Thursday, May 6 at 6:00pm

Henrik Wann Jensen
Toshiya Hachisuka

Introduction

This assignment involves implementing a bounding volume hierarchy and making the ray tracer fast for scenes with a large number of triangles.

To test your implementation we have provided four benchmark scenes (teapot, bunny1, bunny20, and sponza). The code for setting the scenes are available on the class web page as `assignment2.{h|cpp}`. You should render each of these scenes with Miro. Note that if you use your own renderer then you must make sure that the camera, the image resolution, the shading, and the sampling (1 sample through the center of each pixel) matches the implementation in Miro.

For reference here are the number of ray-triangle tests for the four scenes unaccelerated at a low resolution of 32x32:

Scene	**Triangles**	**Ray/Triangle Intersections**
Teapot	577	590,848
Bunny1	69,452	71,118,848
Bunny20	1,389,021	1,422,357,504
Sponza	66,454	68,048,896

Renderings of these scenes with no shadowing are shown in Figure **??**. All your (accelerated) images should be rendered at 512x512 with 1 sample per pixel.

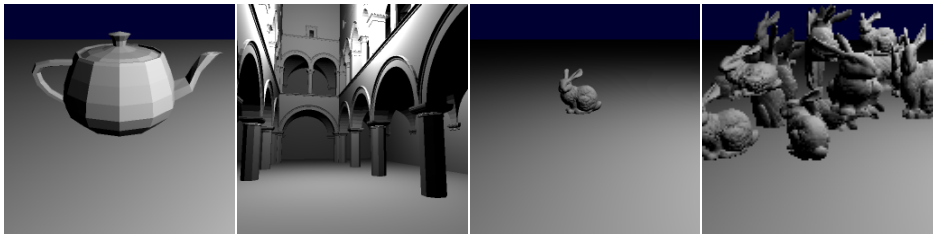

Figure 1: The four benchmark scenes for this assignment rendered in Miro with no shadow rays.

Task 1: Bounding-Box Intersection: 10%

Implement a fast slab-based bounding box intersection routine.

Task 2: Build Bounding Volume Hierarchy: 35%

Build a BVH using a top down approach. Make a global box with all triangles. Use the surface area heuristic (SAH) to perform the local split into sub boxes. Use two child boxes per box and one triangle per leaf box.

Task 3: Intersect Bounding Volume Hierarchy: 25%

Traverse and intersect the BVH. Report rendering statistics for the four provided scenes. Your statistics should include (at a minimum): total number of BVH nodes, number of BVH leaf nodes, total number of rays, number of ray-box intersections, and number of ray-triangle intersections.

Task 4: Examining the Bounding Volume Hierarchy: 25%

Compute statistics for all 4 scenes for different parameters of BVH. Report number of ray/box intersections and number of ray/triangle intersections for the different versions of BVH as follows: Test two versions of the bounding volume hierarchy, one that uses 1 triangle per box and one that uses up to 4 triangles per box. If possible report timings as well for this assignment.

Task 5: Add Shadow Rays: 5%

Add shadow calculation by tracing shadow rays towards the light sources. Render the 4 scenes and report numbers for ray/box and ray triangle intersections using the best parameters found in task 4.

Hacker points:

SSE Box intersection: + 5 %

Add SSE version of the box intersection. You can choose to implement either a single ray-single box intersection or single ray-multiple boxes intersections.

Optimize the BVH: + 10 %

Render Sponza with the optimized BVH. To get full credit, you should do either one of them by improving the BVH construction code:

- Get less than 2 triangle intersections per ray on average.
- Get faster rendering time.

Non-axis aligned boxes: + 20 %

Implement non-axis aligned boxes with pairs of slabs to get a tighter fit for the objects. Try getting faster rendering time compared with the axis aligned boxes in the Sponza scene.

Turnin

Your archive should contain eight images, the code, and intersection statistics.

Render the four provided scenes with and without shadows (eight images in total) at 512^2 resolution. You can also include any other interesting scenes you have rendered so we can include them in the gallery! However, do not include any obj files in your submission. Only rendered images should be included.

Report intersection statistics and include a README report as either ASCII text or a PDF file which includes these statistics. You should also submit your code.

You should package up all source files and project files into a `zip` archive. Do not include any of the provided `.obj` files or any generated files from compilation (i.e. clean your project). You should only need to include the `.cpp`/`.h` files, the `.sln` and `.vcproj` Visual Studio files, plus the Makefiles if any. If your code archive is larger than 1MB, you might be doing something wrong. Note that the grading itself is done on-site, so just sending the code is not enough for get grading.

Your archive should be named according to your first initial and last name. For instance, John Smith should submit `jsmith.zip`. Convert all your images to PNG format before submission and include them in your archive. Prefix your images using the same naming convention, e.g. `jsmith_sponza_noshadows.png`.

Email your archive to cse168-turnin@graphics.ucsd.edu before the deadline and before you get graded.