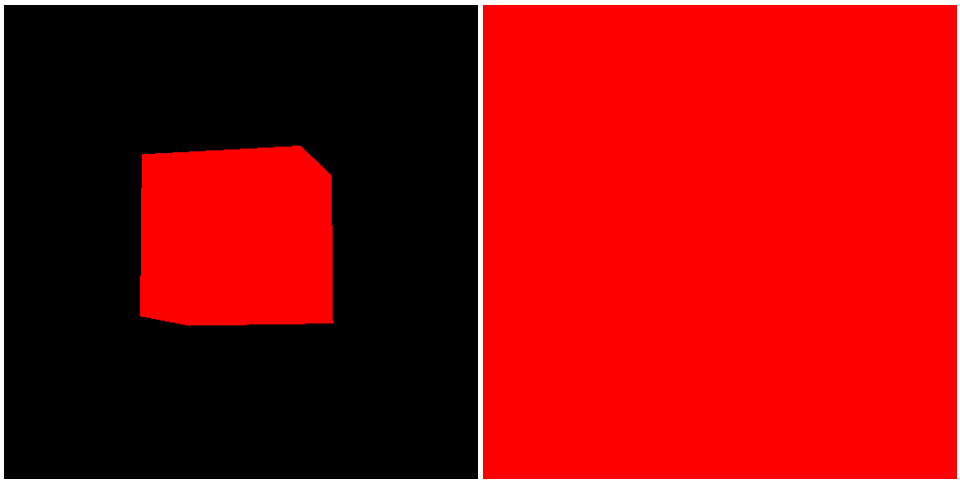
Assignment 3

Hardware Ray Tracing

Due: 11:59:59 PM, February 26, 2014

In this assignment you will implement boxes, triangles, and BVH traversal. We will also use global memory to load the scene, as discussed in class.

1. **Part 1** (10 points): Add boxes to your ray tracer. They do not need to support shading, but they do need to correctly implement ray-box intersection. We will be using boxes for bounding volume hierarchy traversal. To verify your box intersection is correct, generate the following images:



   Both images have a single box in the scene centered around the origin:

```
Box b(Vector(-1.f, -1.f, -1.f), Vector(1.f, 1.f, 1.f));
```

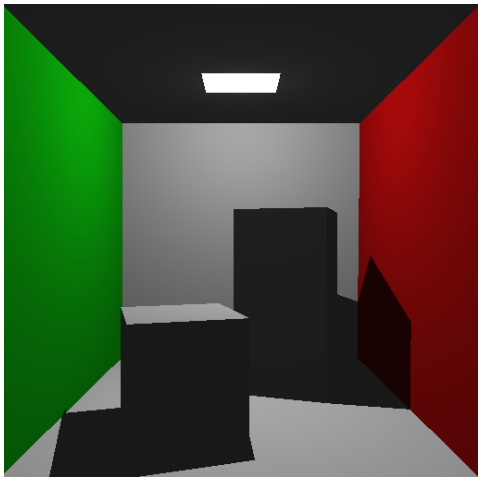
   The image on the left has the camera positioned at:

```
PinholeCamera camera(Vector(4.f, -15.f, -2.f),  // eye
                     Vector(-0.255551, 0.958315, 0.127775),  // gaze
                     Vector(0.f, 0.f, 1.f),  // up
                     0.194f, (float)xres / (float)yres);  // uLen and aspect ratio
```

   The image on the right has the camera positioned anywhere inside the box, looking in any direction (just make sure it's not looking in the same direction as the up-vector). These images don't have any shading, as they are just designed to verify that your intersection test is correct. For each pixel, if the camera ray hits the box, color it red, else color it black. If running in `simtrax`, make sure to disable usimm.

2. **Part 2: Triangles and TRaX scene memory** (10 points): Implement a triangle class in your ray tracer (but don't call it "Triangle", to avoid conflicts). Devise your own tests to first verify that your intersection is implemented correctly, similar to the box test above, then implement simple Lambertian shading from a single point light source for your triangles (just like you did for spheres). The only difference will be how normals are computed.

   Once you are convinced you can render triangles (including Lambertian shading), render the Cornell scene using the camera, light, and triangles in TRaX global memory. Don't use the BVH, just loop through every triangle in memory, as seen in lecture 10. Note that there is only 1 light source in global memory.



To load the Cornell scene for either `run_rt` or `simtrax`, use the following command line arguments:

```
--model <path to samples/scenes/cornell/CornellBox.obj>
--view-file <path to samples/scenes/cornell/cornell.view>
--light-file <path to samples/scenes/cornell/cornell.light>
--disable-usimm (for simtrax only)
```

The light's color is not defined in the view file, so just use white (1.f, 1.f, 1.f).

The ambient light color is [0.4f, 0.4f, 0.4f], `Kd` is 0.7f, `Ka` is 0.3f for every material.

See lecture 10 for information on loading triangles, the camera, and the light from global memory.

3. **Part 3: Traversing the BVH** (30 points)

   (a) The BVH is automatically built and loaded in TRaX global memory, starting at the location pointed to by `start_scene = GetBVH()`. See the lecture 10 slides for details. Adapt your ray tracer from Part 2 so that instead of looping over all triangles in memory, it traverses the BVH, and render the Cornell Box scene using BVH traversal.

   (b) Render the conference scene with the BVH. If everything so far works correctly, nothing about your ray tracer should have to change. This should be a simple matter of loading a different scene:
   Make sure to svn up the samples/scenes directory to get the conference scene.

   ```
   --model <path to samples/scenes/conference/conference.obj>
   --view-file <path to samples/scenes/conference/conference.view>
   --light-file <path to samples/scenes/conference/conference.light>
   --disable-usimm (for simtrax only)
   ```

4. **Analysis** (30 points)

   Make sure to `svn up` the root simtrax directory before starting the analysis.

   Now that we are using global memory, the data caches (L1 and L2) become an important factor on performance. We will focus on the L1 in this assignment. For the experiments below, use the `conference` scene from above, with usimm disabled, rendered at 128x128 resolution, unless otherwise specified. For realistic results, we will limit all caches (icache, L1, L2) to at most 16 banks in all experiments. Do not modify the line size of the L1 or L2 data caches.

   - Single TM configuration

     We will start with a pre-determined single TM configuration that is known to have good performance per area:

     ```
     --num-thread-procs 32
     --num-icache-banks 16
     --num-icaches 2
     --config-file <path to samples/configs/bigcache.config>
     ```

     We will also use the following simulator parameters:

     ```
     --ignore-dcache-area
     --disable-usimm
     --l2-off
     ```

     The `bigcache.config` configuration file has a good mix of functional units for 32 threads, but has a larger than necessary L1 data cache. We will ignore the data caches' area for now. We will also ignore the L2 cache.

     (a) What is the performance per area achieved using your ray tracer and this configuration on the conference scene?

     (b) What is the reported L1 hit rate?

     (c) Calculate the L1 hit + hit under miss rate. This is given by $(Hit\ under\ miss + L1\ hits)/L1\ accesses$. Judging by this, do you expect many data dependence stalls caused by `LOAD` instructions? Explain.

     (d) Now run the same simulation with `--l1-off`. How does this affect the issue rate and performance per area?

     (e) Finally, turn the L1 cache back on, and run the simulation without `--ignore-dcache-area`. What is the the performance per area? Comparing this overly large L1 cache to no L1 cache at all, is it worth the area?

   - L1 cache configuration

     Using the same simulator parameters specified in the previous section, except for `--ignore-dcache-area`, try to achieve higher performance per area by modifying the capacity and/or number of banks of the L1 cache. The cache given in `bigcache.config` may be larger and have more banks than necessary. Keep an eye on the L1 hit rate and data dependence and resource conflict stalls caused by `LOAD` instructions. If the hit rate is low, data dependence stalls will likely go up. If there are a lot of L1 bank conflicts, `LOAD` resource conflicts will go up. Keep in mind a good perf/area configuration will likely still have a fairly high number of data dependence stalls for `LOAD`. Do not modify the line size of the L1 or L2 data caches.

     During your experiments, make sure that you don't see the warning:

     `WARNING: Unable to find area and energy profile for specified L1.`

     If this happens, the simulator is unable to create that particular cache, either due to an incompatible ratio of banks/capacity, or an invalid number of banks or capacity. The capacity must be a power of two between `256` and `2097152` (words), and the number of banks must be a power of two, at most `16`. However, not all combinations are valid. If a particular combination doesn't work try another. See Lecture 12 for details.

     (a) What is the capacity and number of banks for the best L1 configuration you found?

     (b) What is the performance per area?

(c) Using this configuration, run with `--profile`, similar to Assignment 2. Find the `LOAD` instruction that causes the most data dependence stalls. On average, how many stall cycles per execution does this instruction cause?

(d) Judging by the assembly code around the `LOAD` instruction you identified, what type of data is this likely loading (material, box, triangle, other)? Hint: think about how many loads (words) a box requires vs. a triangle. There should be that many loads either all in a row, or with just a few other instructions in-between them.

- Bigger scenes

  Finally, use your best performance per area configuration found in the last part to render the Hairball scene:

  ```
  --model <path to samples/scenes/hairball/hairball.obj>
  --view-file <path to samples/scenes/hairball/hairball.view>
  --light-file <path to samples/scenes/hairball/hairball.light>
  --profile
  ```

  (a) How is the L1 cache hit rate affected compared to the conference scene? How is the issue rate affected?

  (b) Use the profile to find the `LOAD` that causes the most stalls. How does the average number of stalls for this `LOAD` compare to that on the conference scene found above?

5. **TRaX Output** (5 points)

   Include the simtrax output using your best performance per area TM configuration on both the conference and Hairball scenes.

6. **Code listing** (5 points): Link your source code to your web page (preferably .tar). All of the code that you use should be included. You will not be graded on the quality of your code or comments, only on the presence of your source. We will verify that the code you hand in will produce the image(s) turned in.

7. **Creative Image(s)** (10 points): Generate a creative image using any .obj model you want. A google search will produce plenty of results. See `http://graphics.cs.williams.edu/data/meshes.xml` for starters.

What to turn in:

By midnight on the due date, you should send e-mail to teach-cs6958@list.eng.utah.edu with the following information:

1. **URL:** A pointer to a web page containing the following information:

   (a) Required images

   (b) Creative image(s)

   (c) Link to source code

   (d) Analysis (see above)

   (e) TRaX output (see above)

2. **Time required:** How many hours did it take you to complete this assignment?

3. **Difficulty of assignment:** Was the assignment difficult or not? Feel free to expound or to be brief.

You will not be graded on these last two items. They will be used to help improve the class in future assignments and in future years.