Program 5

Parallel Hardware Ray Tracing

Due: 11:59:59 PM, October 20, 2011

This assignment requires a new compiler intrinsic `trax_rand`. Before you start, go in to your LLVM_Trax directory and run `./update.sh`
As always, make sure to 'svn up' and make the simulator directory as well.

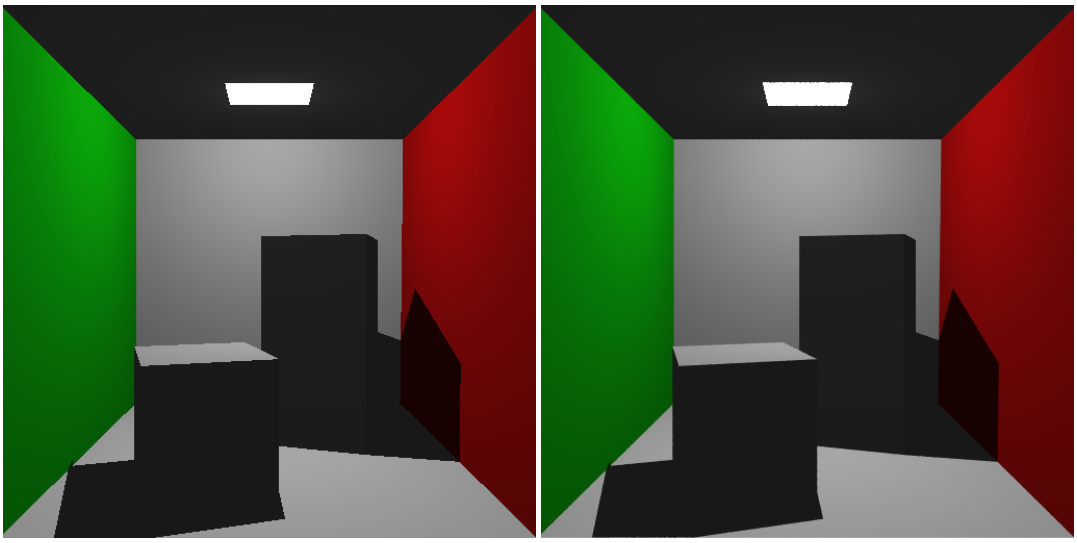
In this assignment you will implement multi-sampling and path tracing.
See the "Sampling" (10/4) and "Path Tracing" (10/6) lecture slides for more information and pseudo code.

1. **Part 1** (40 points): Multi-sampling

   Implement random pixel multi-sampling with a simple box filter for anti-aliasing. This means, for every pixel, instead of casting a ray through the center of the pixel, cast it through a random point in the area of the pixel, and do this `N` times for `N` number of samples per pixel. A box filter is simply an average. Every sample contributes energy with equal weight to the final color of the pixel (divide total color by `num_samples`). This will require random numbers. To get a random number between 0 and 1, use the intrinsic `trax_rand()` with no arguments, as in:

   ```
   float offset = trax_rand();
   ```

   To verify your sampling is working correctly, render the Cornell scene with 20 samples per pixel, and compare it to 1 sample per pixel. Your results should look something like the following:

   On the left is 1 sample per pixel, on the right is 20 samples per pixel. There is a noticeable difference in the edges of the images, specifically on the shadow of the large box. Note that at 512x512, this will take a minute to render at 20 samples per pixel, even in run_rt.

   Extra Credit (20 points)

   Instead of random sampling, implement jittered sampling, which shouldn't be much more difficult once you can find a random point within a pixel's area. Jittered sampling involves dividing the pixel in to a uniform $N * N$ grid, and placing one sample in each grid cell at a random point. This will require that you use a square number of samples. See the lecture slides for more details.

   Extra Credit (20 points)

   Implement a triangle or Gaussian filter instead of a box filter. See the slides for details.

2. **Part 2** (40 points): Path tracer

   Implement Kajia style path tracing for global illumination in your ray tracer. Since TRaX only supports a single point-light source, pure path tracing will not work. Kajia path tracing is described in the (10/6) lecture slides. Remember to terminate the ray path if it hits the background.

   For this assignment, remember to turn off the ambient term! The only light added to your image should be directly from the light source, and from the indirect global illumination rays.

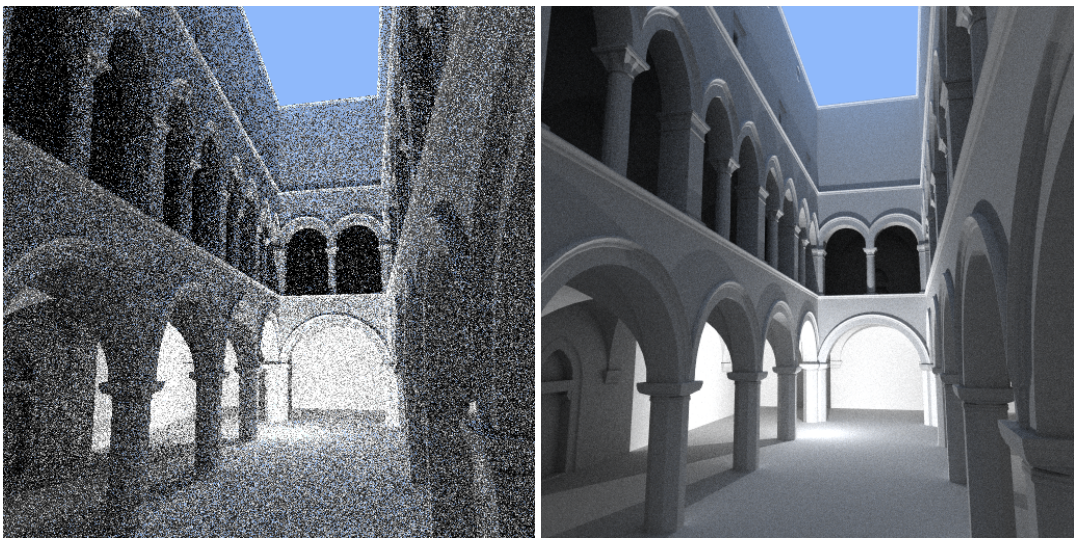
   The test_models directory has been updated with the Sponza atrium model "sponza.obj". To render the Sponza scene in run_rt, add these lines to the Makefile, and comment out the other 3 scene lines:

```
#VIEWFILE=\"${SIMDIR}/views/sponza_obj.view\"
#MODELFILE=\"${SIMDIR}/test_models/sponza.obj\"
#LIGHTFILE=\"${SIMDIR}/lights/sponza.light\"
```

   Note that there is no material file for sponza.obj. TRaX will automatically load a default material and assign it to every triangle, so you shouldn't need any special code to handle this. The background color is [0.561f, 0.729f, 0.988f].

   Render the Sponza scene with at least 20 samples per pixel and a max ray depth of 5. You may use more samples per pixel as you desire, however, this will take up to an hour with only 20 samples, even in run_rt.

   Before you start your final rendering, make sure your shading looks reasonable first by rendering with fewer samples. As a reference, below is the scene rendered with one sample per pixel (left) and 100 samples per pixel (right).



   If your image is brighter than this, you probably forgot to turn off the ambient term.

   Required TRaX output

   Render the sponza scene with 15 samples per pixel and a max ray depth of 3 on the full TRaX system, and attach the output to your handin web page. The TRaX memory loader supports options for specifying num_samples and ray_depth, if you don't want to hard-code them (`loadi(0, 16)`, and `loadi(0, 17)`).

```
./simhwrt --load-assembly <path to your assembly>
--model test_models/sponza.obj
--view-file views/sponza_obj.view --light-file lights/sponza.light
--num-cores 20 --num-thread-procs 32 --num-l2s 4
--num-icaches 2 --num-icache-banks 16
--num-samples 15 --ray-depth 3
```

3. **Creative Image** (20 points): Generate a creative image of your choice using path tracing.

4. **simhwrt output**: Link all output files mentioned above to your web page.

5. **Code listing**: 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.

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

7. **Diffuculty**: What was the hardest part of this assignment, why?

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.

By midnight on the due date, you should send e-mail to teach-cs6965@list.eng.utah.edu with a URL for your assignment web page.