Program 1

Hardware Ray Tracing

Due: 11:59:59 PM, January 22, 2014

Before you start, make sure you update the `simtrax` directory with `svn up`, as there may be a patch. Also fully read the TRaX programming guidelines:
https://code.google.com/p/simtrax/wiki/Programming
This will save you lots of time and energy!

Implement classes for `Vector`, `Ray`, `Color`, and `Sphere` as discussed in class. You should implement all of the legal operators including arithmetic operators, dot product, cross product, vector length and vector normalize, as well as any other functions you find useful.

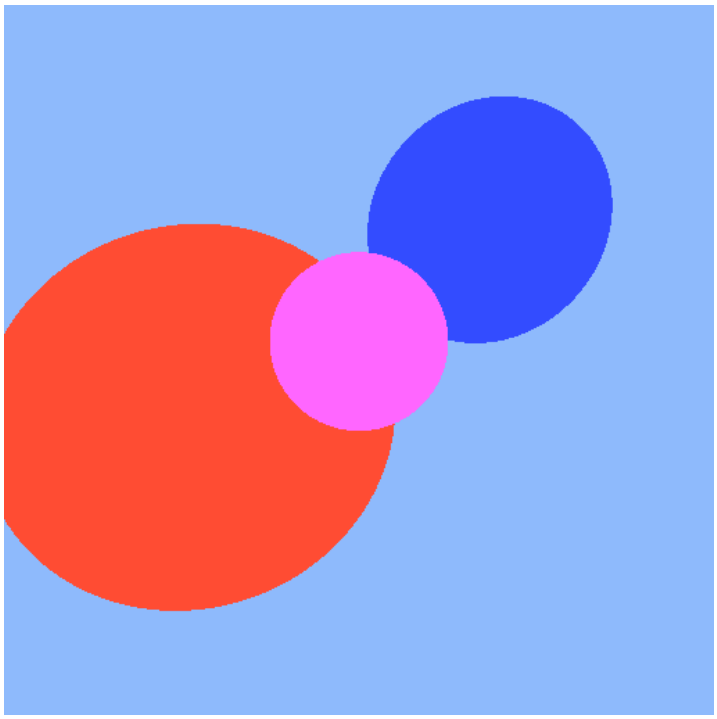
See the examples in `simtrax/samples/src/...` as a starting point. Particularly `mandelbrot` has a simple `Color` and `Image` class that you can use as a starting point for your own `Color` and `Image` classes.

Your `Sphere` class will be replaced with a more powerful version in a future assignment, but will be used here for the purpose of testing the above infrastructure. The sphere should record the center and radius and have a single method called `intersects` that returns whether a ray hits the sphere or not. For this assignment, we are not concerned with the intersection distance.

1. **Required Image(s)** (40 points): Your ray tracer should reproduce the image shown here containing three spheres (without shading). The image is 512x512 resolution.

   The image is generated by creating three spheres:

   - Sphere 1: Center: [0.0f, 0.2f, 1.1f] Radius: 1.0f
   - Sphere 2: Center: [1.4f, 1.5f, 1.2f] Radius: 1.3f
   - Sphere 3: Center: [-1.5f, -0.5f, 1.0f] Radius: 1.9f



The following code will generate the rays to test against the three spheres. This assumes your `Image`, `Color`, and `Ray` classes use the same constructors as mine. You are free to modify this code, or do away with it altogether, as long as the produced image is correct.

```
#include "trax.hpp"

int xres = GetXRes();
int yres = GetYRes();
int start_fb = GetFrameBuffer();
Image image(xres, yres, start_fb);

// atomic increment allows multiple threads to get unique pixel assignments
for(int pix = atomicinc(0); pix < xres*yres; pix = atomicinc(0))
  {
    // i, j are the framebuffer pixel coordinates
    // [i = 0,      j = 0]      is bottom-left
    // [i = yres-1, j = xres-1] is the top-right
    int i = pix / xres;
    int j = pix % xres;

    // x, y are the normalized image-space coordinates
    // [x = 0,   y = 0]   is the middle
    // [x = -1f, y = -1f] is the bottom left
    // [x = 1f,  y = 1f]  is the top-right
    float x = 2.f * (j - xres/2.f + 0.5f)/xres;
    float y = 2.f * (i - yres/2.f + 0.5f)/yres;

    // This is our implicit "camera"
    Ray ray(Vector(0.f,0.f,-3.f), Vector(x, y, 1.f));

    Color result;
    if(sphere1.intersects(ray))
       result = Color(1.f, .4f, 1.f);
    else if(sphere2.intersects(ray))
       result = Color(.2f, .3f, 1.f);
    else if(sphere3.intersects(ray))
       result = Color(1.f, .3f, .2f);
    else
       result = Color(0.56f, 0.73f, 0.99f);
    image.set(i, j, result);
  }
```

Notice a few things about the code above:

- `atomicinc` is an intrinsic instruction for the TRaX architecture which atomically (thread-safe) returns a unique consecutive integer, starting at 0. This is the mechanism for assigning work to threads.
- Your code can not use the `double` data type at all. This includes immediate values. Notice that all of my immediate values (even integers that would be cast to float), are specified as being float, such as `2.f`.
- It is recommended that you put all of your code in a subdirectory of `simtrax/samples/src`. Copy the Makefile from an existing example.

2. **Analysis** (40 points)

   Before using the real simulator, make sure your program is generating the correct image by using the functional simulator (`run_rt`).

   Generate an analysis document. This can either be a separate document (pdf preferred), or simply inline on your assignment web page. Your document should address the following:

   For all of the experiments below, use a resolution of 128x128 (default).

   We will start by running our ray tracer on the simplest TRaX configuration with just 1 thread, 1 instruction cache, and 1 of each functional unit. Use the following command-line arguments for `simtrax`:

   - `--num-thread-procs 1`
   - `--num-icaches 1`
   - `--num-icache-banks 1`
   - `--config-file ../samples/configs/tiny.config`

     The above 4 items are the only ones we will modify in our analysis. Do not remove or modify the below:
   - `--l1-off`
   - `--l2-off`
   - `--disable-usimm`
   - `--no-scene`
   - `--load-assembly <path to your ray tracer rt-llvm.s>`

   Since we are not using main memory, we will turn off the L1 (`l1-off`), L2 (`l2-off`), and DRAM(`disable-usimm`) so that they don't affect chip area or energy. We use `no-scene` since the scene (3 spheres) is hard-coded in your program.

   (a) **Performance / Area**

   Experiment with adding more threads by increasing `--num-thread-procs`. Plot the performance (FPS) and performance per area (FPS / total area) using 1-8 threads, without changing any other parameter.

   - At what point does adding more threads become not worth the extra area?
   - Judging by the reported area of each component on the chip, why does performance / area increase at all when changing from 1 thread to 2? In other words, if two threads roughly doubles the FPS, why doesn't 2 threads double the area?

   (b) **Bottlenecks**

   For this section you will modify the instruction caches (`--num-icaches`, `--num-icache-banks`), and/or the **number** (not latency) of functional units (`--config-file`). The number of icache banks must be a power of 2, and $<= 64$ (valid options are: 1, 2, 4, 8, 16, 32, 64). When modifying the config-file, do not change the unit latencies (2nd column), only change the number of them available (3rd column). See the comments in `tiny.config`.

   - Using 8 threads, which component of the chip appears to be causing the most stalls (preventing instructions from issuing)?
   - Without reducing the number of threads, try to eliminate this bottleneck by modifying either the instruction cache(s) or functional units, but not both. Try to do this in an area-efficient way (keep in mind the total chip area reported, don't go overboard). How did you remove the bottleneck? What is the new performance per area of your chip?
   - After removing this bottleneck, is a new bottleneck revealed? If so, try to remove it. How did you remove the bottleneck? What is the final performance per area?

   When trying to optimize performance, keep in mind that you can't get rid of data dependence stalls with this simple chip. See the lecture 4 slides, and

   https://code.google.com/p/simtrax/wiki/Output for tips and information on performance analysis.

3. **TRaX Output** (10 points) Hand in the output of the simulator generating the required image at 128x128 for both the baseline single-thread TRaX processor, and your chosen best performance/area configuration found in the previous section.

4. **Code listing** (10 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.

5. **Creative Image(s)** For this assignment only, the creative image is optional. You can't get too creative with circles. Create an image of your choosing using unshaded spheres and our fake "camera" model. Note that since we are not using main memory yet, your spheres will be on each thread's local stack; use only a small number of them (less than 10).

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 image
   (b) Link to source code
   (c) Analysis (see above)
   (d) 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.