TRaX memory clarifications

Loading materials

Triangle IDs

Performance investigation

Reminder

- Always ‘svn up’ and make the sim directory before starting an assignment
- I recently fixed a bug involving triangle IDs

Materials

- Materials are 25 words each, starting at `int start_matls = loadi(0, 9)`
- Triangles have an ID to their material
  - `int shader_id = loadi(tri_addr + 10)`
  - assuming you have found “tri_addr” of the hit triangle
- int shader_addr = start_matls + (shader_id * 25)

Materials

- The diffuse color of each material is what we care about for now
- Diffuse color is stored in the material at offset 4(r), 5(g), 6(b)
-
```
Color matl (loadf(shader_addr, 4),
            loadf(shader_addr, 5),
            loadf(shader_addr, 6));
```

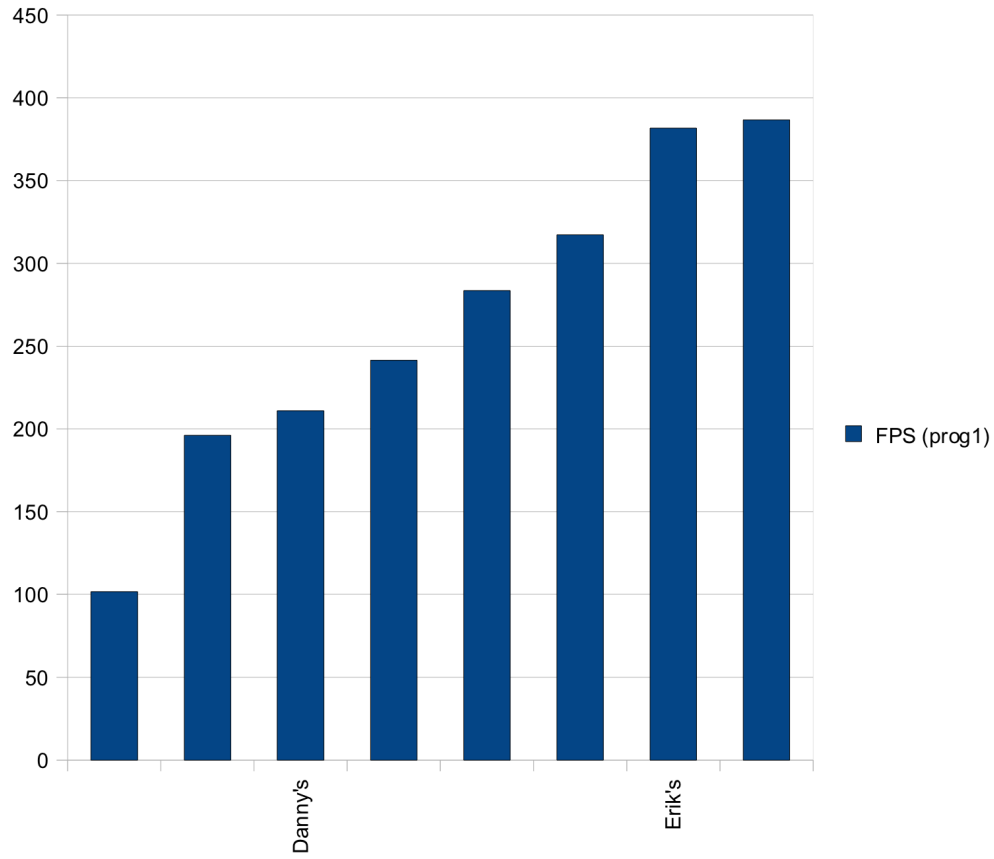
Triangle IDs

- In order to shade a triangle, we need to keep track of which one was hit
- In the HitRecord, save either:
  - triangle address
  - triangle ID (saved with tri in global mem)

Triangle IDs

- triangle address
  - My triangle constructor takes the global memory address of the triangle, so that the HitRecord can save it later
- triangle ID
  - Alternatively, you can load the triangle's ID from memory
    - ID = loadi(tri_addr, 9);
    - Then recompute address from ID for shading

Program 1 Performance



How could Erik possibly beat me?

- I did some investigating…
- Program 1 consists almost entirely of sphere intersection tests
  - profile the sphere::intersect code

Sphere performance

```
bool Sphere::intersects(const Ray&
  ray)
{
   profile(0);

   ... // sphere intersect code

  profile(0);
  return;
}
```

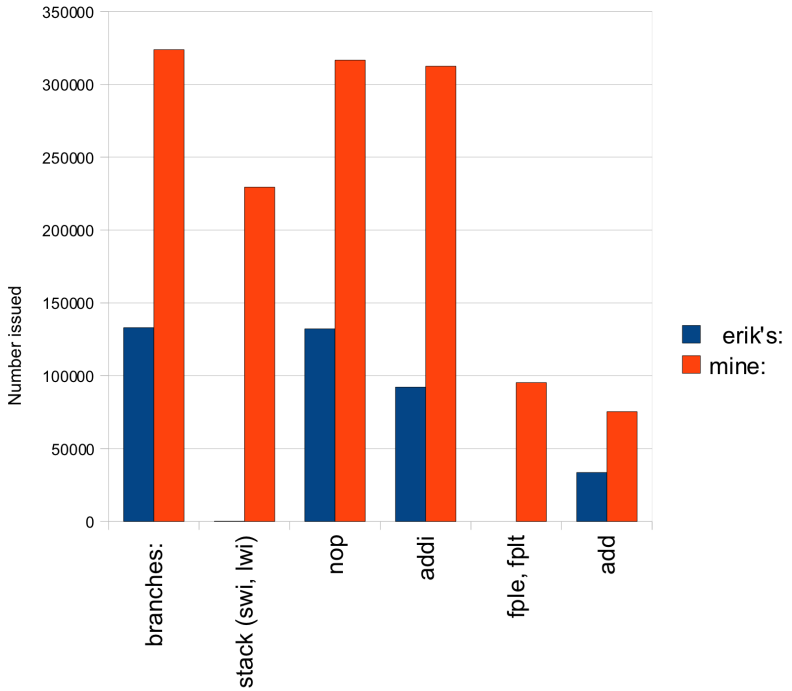
Sphere profile

	Erik's	Danny's
Numer of kernel calls	46099	46099
Total cycles spent	648200	1364732

Performance breakdown

- But our sphere code is essentially identical, something else going on?
- To be sure, I copied Erik's sphere code to my ray tracer – no difference
- Total instructions issued:
  - Erik's: 1746803
  - Danny's: 2761856

Performance breakdown



Performance breakdown

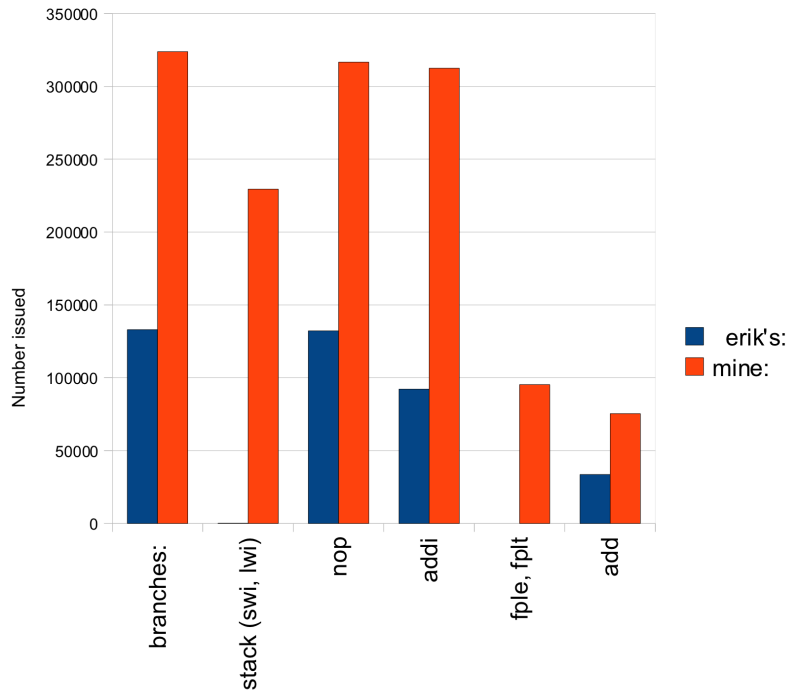
- The stack operations indicate that something of mine doesn’t want to live in registers
- Most of the work is done in the vector class

Vector.h

- Erik's
  - `float x, y, z`
- Mine
  - `float data[3]`
- Same with Color.h
  - `float r, g, b // Erik's`
  - `float data[3] // mine`

Performance breakdown

Notice the difference in `addi`, comes from array offset computation



Vector.h

- After using float x, y, z, my performance was within ~1% of Erik's