CS 6958
LECTURE 10
LOADING THE SCENE

February 10, 2014
Creative!
Creative!
Creative!
Creative!
Bloopers
Bloopers
Profiling / Reordering

- Examining one top offender instruction yields:
  - Reducing stalls by up to 3 cycles
  - Increasing performance by up to 0.5%

- Automated reordering (peephole optimizer)
  - Maybe 5% performance boost

- Real offender will be when LOADs take > 100 cycles
## Branches in Profile

<table>
<thead>
<tr>
<th>Assembly:</th>
<th>Num Executions</th>
<th>Data Stall</th>
<th>Cycles</th>
<th>Stalls / Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>3126: FPDIV r22, r22, r8</td>
<td>28805</td>
<td>429893</td>
<td>14.92</td>
<td></td>
</tr>
<tr>
<td>3134: FPDIV r30, r26, r12</td>
<td>28805</td>
<td>403270</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>3144: FPDIV r25, r25, r26</td>
<td>28805</td>
<td>383632</td>
<td>13.32</td>
<td></td>
</tr>
<tr>
<td>3123: FPINV SQRT r8, r26</td>
<td>28805</td>
<td>374465</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>
Profiled Block

bneid     r3,     $0B_54
FPDIV     r22,    r22,     r8
brid      $0B_55
FPRSUB    r3,     r22,     r30

$0B_54:
FPADD     r3,     r30,     r22

$0B_55:
ORI       r8,     r0,      .5
FPMUL     r26,    r3,      r8
Execution Order 1

1. bneid r3, $0B_54
2. FPDIV r22, r22, r8
  brid $0B_55
  FPRSUB r3, r22, r30
  $0B_54:
3. FPADD r3, r30, r22
  $0B_55:
4. ORI r8, r0, .5
5. FPMUL r26, r3, r8
Execution Order 2

1  bneid  r3, $0B_54
2  FPDIV  r22, r22, r8
3  brid  $0B_55
4  FPRSUB  r3, r22, r30

$0B_54:
5  FPADD  r3, r30, r22

$0B_55:
6  ORI  r8, r0, .5
7  FPMUL  r26, r3, r8
Recap: Global Memory

- loadi, loadf, storei, storef
- Lots of data pre-loaded
- --view-file
  - Camera data
- --model
  - Geometry (.obj or .iw format)
  - BVH info (built from geometry)
  - Material info (.obj files specify a .mtl file)
- --light-file
  - Light
Recap: Data Pointers

- Light
  - loadi(TRAX_START_LIGHT)

- Camera
  - loadi(TRAX_START_CAMERA)

- Model
  - BVH/Grid
    - loadi(TRAX_START_SCENE)
  - Triangles
    - loadi(TRAX_START_TRIANGLES)
  - Vertex normals
    - ...
  - Texture coordinates
    - ...

- Materials
  - loadi(TRAX_START_MATLS)
Light Files (--light-file)

- Very simply format (too simple)

- 3 floats specifying the location of a point light

  278.0 279.5 540.0
Loading Lights From Memory

- **Old constructor:**
  ```cpp
  PointLight(const Vector& position, ...)
  ```

- **New constructor:**
  ```cpp
  PointLight(int addr)
  {
      position = Vector(loadf(addr, 0), 
                         loadf(addr, 1), 
                         loadf(addr, 2));
      ...
  }
  ```

  ```cpp
  PointLight theLight(loadi(TRAX_START_LIGHT));
  ```
In Vector.h

```cpp
inline Vector loadVectorFromMemory(const int &address) {
    return Vector(loadf(address, 0),
                  loadf(address, 1),
                  loadf(address, 2));
}
```

Similarly for Color, ...
Simple text format (details soon)

278 -549 274 0 1 0 0 0 1 0 -1 1 -1 1 2

- Eye
- Gaze
- Up

Image plane dimensions

Aperture

Film plane distance

Simulator builds and loads camera using this info
Loading the Camera

- `start_camera = loadi(TRAX_START_CAMERA)`

- `eye[x, y, z]` is at `start_camera + [0..2]`
- `(deprecated):` `start_camera + [3..8]`
- `up:` `start_camera + [9..11]`
- `gaze:` `start_camera + [12..14]`
- `u:` `start_camera + [15..17]`
- `v:` `start_camera + [18..20]`
- `Advanced info:` `start_camera + [21..33]`
Recommend a constructor that takes an address

PinholeCamera::PinholeCamera(int addr) {
    eye = loadVectorFromMemory(addr + 0);
    up = loadVectorFromMemory(addr + 9);
    gaze = loadVectorFromMemory(addr + 12);
    u = loadVectorFromMemory(addr + 15);
    v = loadVectorFromMemory(addr + 18);
}

PinholeCamera camera(loadi(TRAX_START_CAMERA));
Triangles

- Stored as 11 words:
  - p1[x, y, z] (address + 0..2)
  - p2[x, y, z] (address + 3..5)
  - p3[x, y, z] (address + 6..8)
  - Deprecated (address + 9)
  - material ID (address + 10)
Triangle Constructor

- **Recommend:**
  ```
  Tri(int address)
  {
    load 3 vertices
    load mtl ID
    ID = address;
  }
  ```
  Don’t call your class “Triangle”!

- **Save the address as the ID**
  - Save this ID in the HitRecord
  - In case we need it later (depends on implementation)
Triangle Normals

```cpp
inline Vector normal() const {
    Vector edge1 = p1 - p3;
    Vector edge2 = p2 - p3;
    Vector n = Cross(edge1, edge2);
    n.normalize();
    return n;
}
```

- Don’t compute normal unless you need to shade this triangle
- Consider saving e1 and e2 during intersection
  - Then pass HitRecord to normal()
Avoid Memory Traffic

- Should we load the material ID when checking for intersection?

- In BVH traversal:
  
  if leaf node
  
  for all primitives in leaf
  
  Tri t(address)
  
  Load in constructor? (save in HitRecord)
  
  Likely to be cached anyway

- In shade():

  mtl_ID = loadi(hitrecord.getObjID(), 10)
  
  Or load only when needed?

  mtl_id at tri address + 10
Avoid Memory Traffic

- Save as much as possible in HitRecord

- Recommend:
  - e1, e2 (computed during intersection) → use to compute normal
  - Material ID

- Alignment
  - 11 words is very odd size for 16-word cache lines
  - Memory loader doesn’t do any padding for triangles
Better(?) Triangle Normals

```cpp
inline Vector normal(const HitRecord& hr)
const
{
    Vector n = Cross(hr.edge1, hr.edge2);
    n.normalize();
    return n;
}
```
Before we get into BVH traversal, a simpler example

- use start_triangles, and num_triangles

- Simply loop through every triangle, loading them from memory
Traversing the scene

```cpp
int start_tris = GetStartTriangles();
int num_tris = GetNumTriangles();
for(int i=0; i < num_tris; i++)
{
    Tri t(start_tris + (i * 11));
    t.intersect(hitRec, ray);
}
```

- Recall that Tri constructor takes address, generates loads
Materials are 25 words each, starting at:

```c
int start_matls = GetMaterials();
int shader_id = hitRec.getMtlID();
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 offsets:
  - 4(r), 5(g), 6(b)

- Color diffuse(loadf(shader_addr, 4),
  loadf(shader_addr, 5),
  loadf(shader_addr, 6));
Assignment 3 (part 1)
Recap: BVH

- Facilitates fast ray-geometry intersection
  - Try to avoid obviously missed geometry
Recap: BVH

- Facilitates fast ray-geometry intersection
  - Only traverse branches that the ray hits
  - Basically depth-first traversal (conditional)
Traversing the BVH

- No recursion!
  - Stack frames will quickly outgrow local memory space
  - Function calls are expensive

- But tree traversal is inherently recursive

- Use a software-managed stack
  
  ```c
  int stack[32]; // holds node IDs
  int sp = 0;    // stack pointer
  ```
current = root ID (0)
while(true)
    if(ray intersects current node)
        if(interior node)
            push rightChildID
            current = leftChildID
            continue;
        else
            intersect-test all triangles in leaf
    if(stack is empty)
        break;
break;
current = pop stack
Box::intersect(HitRecord& hit, const Ray& ray) const{
    float tnear, t2;
    Vector inv = 1.f / ray.direction();
    Vector p1 = ( c1 - ray.origin() ) * inv;
    Vector p2 = ( c2 - ray.origin() ) * inv;
    Vector mins = p1.vecMin( p2 );
    Vector maxs = p1.vecMax( p2 );
    tnear = max( mins.x(), max( mins.y(), mins.z()) );
    t2 = min( maxs.x(), min( maxs.y(), maxs.z()));

    if(tnear < t2) ...
        Make sure to account for inside hits!
BVH layout

```cpp
int start_bvh = GetBVH();
```

<table>
<thead>
<tr>
<th>box corner (3 floats)</th>
<th>box corner (3 floats)</th>
<th>child ID</th>
<th>num children</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_min</td>
<td>c_max</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

(-1 indicates interior node)

Single BVH node (8 words)

```
start_bvh
```

```
start_bvh + 8
```
Sibling nodes are next to each other in memory

Right child’s ID is always left_id + 1

start_bvh + (2 * 8)  start_bvh + (13 * 8)
As with all data held in global memory, recommended:

```cpp
BVHNode::BVHNode(int addr) {
    box.c1 = loadVectorFromMemory(addr + 0);
    box.c2 = loadVectorFromMemory(addr + 3);
    num_children = loadi(addr + 6);
    child = loadi(addr + 7);
}
```
Leaf Nodes

- Implied differently:
  - `num_children > 0`
  - `child_ID = address of node’s first triangle`
    - Not ID of first triangle!
    - Leaf node’s triangles are consecutive in memory
Leaf Nodes

<table>
<thead>
<tr>
<th>box corner (3 floats)</th>
<th>box corner (3 floats)</th>
<th>child</th>
<th>num tris</th>
<th>Remaining BVH nodes</th>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_min</td>
<td>c_max</td>
<td>682</td>
<td>2</td>
<td>...</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>T2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

682 (address, not ID!)
Example

```c
inline void intersect(HitRecord& hit, const Ray& ray) const {
    int stack[32];
    int node_id = 0;
    int sp = 0;
    while(true){
        int node_addr = start_bvh + node_id * 8;
        BVHNode node(node_addr);
        HitRecord boxHit;
        node.box.intersect(boxHit, ray);
        if(boxHit.didHit())
            // and so on...
    }
}
```
left_id = node.child;
if ( node.num_children < 0 ) // interior node
{
    stack[ sp++ ] = left_id + 1;
    continue;
}
// leaf node
tri_addr = left_id;
for ( int i = 0; i < node.num_children; ++i)
{
    // intersect triangles
}
// ... finish outer loop, manage stack
BVH Implementation

- My bvh class contains just a pointer to start_bvh

```
auto start_bvh = _start_bvh;
```

- Nodes are loaded one at a time as needed
inline void intersect(HitRecord& hit, 
const Ray& ray) const

- Note that this hit record passed in is for the final hit triangle (or none if background)

- Don’t use the same one for testing against boxes!
for each pixel...
  Ray ray;

camera.makeRay(ray, x, y);

HitRecord hit;

scene.bvh.intersect(hit, ray);

result = shade(...);
Program 3 (part 2)
Performance

- Remember, there are some optimizations:
  - Traverse down closer child first
  - Don’t traverse subtree if closer triangle already found
  - The pseudo-code I’ve shown doesn’t do this
- Can be tricky!
  - What if node intersection occurs inside box?