CS 6958
LECTURE 4
INTRO TO GRAPHICS GEOMETRY

January 15, 2014
Administrative

- Email me if you are not on the mailing list
- dkopta@cs.utah.edu
- teach-cs6958@list.eng.utah.edu
Geometry for Graphics
Geometry for Graphics

“If it doesn’t have any math, it probably isn’t worth doing.
If it has too much math, it definitely isn’t worth doing”

-Steve Parker
Geometry for Ray Tracing

- Point
- Vector
- Triangle
- Ray
- Plane, Sphere, Disk, Cylinder, …
  - All of these can be made with tessellated triangles

- Assumed that all of these are 3D
Vectors

- In 3D, a vector is just 3 floats
  - \( <x, y, z> \)

- Intuitive meaning: a direction and length
Points

- Also just 3 floats
  - \( <x, y, z> \)
- Intuitive meaning: a point in space
Vectors

- **Addition**
  
  \[ A + B = \langle A.x + B.x, A.y + B.y, A.z + B.z \rangle \]
Vectors

- Scalar multiplication
  - $2\mathbf{A} = <2\mathbf{A}.x, 2\mathbf{A}.y, 2\mathbf{A}.z>$

![Graph showing scalar multiplication of a vector](image-url)
Vector Properties

- If $A$, $B$, $C$ are vectors
- Commutative
  - $A + B = B + A$
- Associative addition
  - $(A + B) + C = A + (B + C)$
- Additive Inverse
  - $A + (-A) = 0$
Vector Properties

- A, B, C are vectors, r, s are scalars
- Associativity of scalar multiplication
  - \((rs)A = r(sA)\)
- Distributivity of scalar sums
  - \((r + s)A = rA + sA\)
- Distributivity of vector sums
  - \(r (A + B) = rA + rB\)
Vectors vs. Points

- If $P$ is a point, $V$ is a vector.
- What does $P + P$ mean?
  - Translation?
- What does translation mean for $V$?
- In code they’re both just a set of 3 floats.
Vectors/Points

- $P_1 = P_0 + V$
Vectors/Points

- \( \mathbf{V} = \mathbf{P}_1 - \mathbf{P}_0 \)
- Distance between \( \mathbf{P}_0 \) and \( \mathbf{P}_1 \) = \( |\mathbf{V}| \)
Points vs. Vectors

- Operators that do make sense:
  \[ P = P + V \]
  \[ P = P - V \]
  \[ V = P - P \]
  \[ V = V + V \]
  \[ V = V - V \]
  \[ V = s \times V \]
  \[ V = -V \]
  \[ V = V \times V \]
Other Operators

- A, B are vectors
  - Dot product “multiplies” them, but results in a scalar
- $\mathbf{A} \cdot \mathbf{B} = A_x \cdot B_x + A_y \cdot B_y + A_z \cdot B_z$
Useful Properties

\[ \cos(\theta) = \frac{A \cdot B}{|A||B|} \]
Useful properties

- Length of a vector $V = |V|$

  $$= \sqrt{Vx^2 + Vy^2 + Vz^2}$$

  Or: $|V| = \sqrt{V \cdot V}$

- $V \cdot V = \text{length squared}$
A vector with unit length (1.0) is “normalized”

If $V \cdot V = 1$, $V$ is normalized
Rays

- Origin $O$, direction $V$, distance $t$
- Normalize $V$ so we can parameterize distance
- $t = $ distance to object
- $|V| = 1$
- Intersection point $P$:
  - $O + tV$
Normalizing a Vector

\[ \text{unit}(V) = \left\langle \frac{V_x}{|V|}, \frac{V_y}{|V|}, \frac{V_z}{|V|} \right\rangle \]

\[ |V| = \sqrt{V \cdot V} \]

Preserves direction of \( V \)
If $A$ and $B$ are normalized vectors

$$\cos(\theta) = \frac{A \cdot B}{|A||B|} = A \cdot B$$

- We find ourselves regularly normalizing ray directions!
Cross Product

- A, B, C are vectors
- C = A x B
- “right hand rule”

\[<AyBz - AzBy, AzBx - AxBz, AxBy - AyBx>\]
Cross Product

- \( C = A \times B \)
- \( C \) is perpendicular to both \( A \) and \( B \)
- \( |A \times B| = |A| \cdot |B| \cdot \sin \theta \)
- \( |A \times B| = \) area of swept rectangle
- If \( A = B \) or \( A = -B \), then \( A \times B = 0 \)
Colors

- **We will use Red, Green, Blue (RGB)**
  - Other color spaces exist

- **A color is 3 floats (but don’t use Vector!)**
  - Range is 0 – 1  (but allow them to go higher)

- **Operators:**
  - Color * Color
  - Color * scalar
  - Color - Color
  - Color + Color
Images

- An image is a helper class
  - 2D array of pixels
  - Pixel != Color (must clamp to 0 – 1 first)
- Knows where the framebuffer is
- Has function to set pixel, given (color, i, j)
  - Convert i, j coordinates to address in framebuffer
Framebuffer is “upside down”

Real world
Our ray tracer
OpenGL
Taught since 2nd grade

Television
Raster Images
Other 1950’s technology
2D arrays
Framebuffer is “upside down”

\[ \text{start}_\text{fb} + ((xres*3) \times yres) \]

\[ \text{start}_\text{fb} + ((xres*3) \times yres) + (xres \times 3) \]
Geometric Queries

- Back to the original question:
- Ray tracing: determine if (and where) rays hit an object

May be interested in the point itself, or just the distance
Geometric Queries

- Ray-Plane
- Ray-Sphere
- Ray-Triangle
- Ray-Cylinder, Disk, Ring, Torus, Mobius Strip
- Ray-Box
- Ray-implicit function
Ray-sphere intersection

- Points on a sphere are equidistant from the center.
- Our measure of distance: dot product.
- Equation for sphere:

\[(\vec{P} - \vec{C})(\vec{P} - \vec{C}) - r^2 = 0\]

Assume intersection point P exists.
Ray-sphere intersection

- Now represent $P$ as a function of the ray
  - Assume intersection point $P$ exists

$$\vec{P} = \vec{O} + t\vec{V}$$

- $t$ is what we need
Ray-sphere intersection

- Using both equations:

\[
\left( \vec{P} - \vec{C} \right) \cdot \left( \vec{P} - \vec{C} \right) - r^2 = 0
\]

\[
\vec{P} = \vec{O} + t\vec{V}
\]

- Substitute:

\[
(O+tV-C) \cdot (O+tV-C) - r^2 = 0
\]

\[
t^2V \cdot V + 2t(O-C) \cdot V + (O-C) \cdot (O-C) - r^2 = 0
\]
Ray-sphere intersection

t^2V \cdot V + 2t(O-C) \cdot V + (O-C) \cdot (O-C) - r^2 = 0

- Quadratic equation with:
  - a = V \cdot V
  - b = 2(O - C) \cdot V
  - c = (O - C) \cdot (O - C) - r^2

- Roots:

\[
\frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad \frac{-b - \sqrt{b^2 - 4ac}}{2a}
\]
Ray-sphere intersection

- If the discriminant is negative, the ray misses the sphere
- Otherwise, there are two distinct intersection points (the two roots)
Ray-sphere intersection

- Which root do we care about?
- Be careful!
  - Don’t want rays behind the camera showing in our image
Assignment 1
Assignment 1

- **Implement:**
  - Vector
  - Color
  - Image
  - Ray
  - Sphere
class Ray {
public:
    Vector org;
    Vector dir;
    Ray() {
    }
    Ray(const Vector& origin, const Vector& direction) :
        org(origin), dir(direction) {
    }
}
TRaX Programming

- [ ] https://code.google.com/p/simtrax/wiki/Programming

- [ ] See simtrax/samples/src/... for examples

- [ ] Always include “trax.hpp”

- [ ] void trax_main() instead of int main()
Guidelines/Restrictions

- The TRaX compiler can be a bit fragile
  - Doesn’t quite support full C++

- However, these restrictions are likely to result in faster code
No double data types

- **Single-precision float only!**
  - Using double *may* cause compilation to fail

- **Can be tricky to avoid**
  - `float pi = 3.14` ✗
  - `float pi = 3.14f` ✔
  - `float f = 5;` ✗
  - `float f = 5.f;` ✔
No dynamic memory

- `new/malloc, free/delete`
  - We will see how to deal with this when we start using main memory

- **No destructors!**
  - Compiler will fail

- Any variables you declare are on the per-thread stacks
  - Including arrays
  - Remember tiny stack space
Global variables

- Avoid them

- Primitive types (float, int, ...) should be OK

- class/struct types will cause compiler to fail
Inheritance/Templates

- Avoid them

- Polymorphism is slow
Pass By Reference

- Do this whenever possible
- Copying arguments is slow
  - Especially for large data types
- Pass by const reference keeps the original data safe
Standard libraries

- They don’t exist for TRaX
  - No one has compiled them for TRaX

- Only include “trax.hpp” and your own source files
  - trax.hpp gives you some standard library functionality
  - printf
  - Various math routines
Some of the useful functions (more as we need them)

- invsqrt(float f)
- sqrt(float f)
- min(float a, float b)
- max(float a, float b)
-GetXRes()
-GetYRes()
-GetFrameBuffer()
Implementation

- For Vector, Color
  - Overload all operators that make sense
  - Vector operator*(float s) const;
  - Vector operator+(const Vector& v) const;
  - ...

- Up to you if you want to distinguish Point/Vector

- Other useful Vector ops
  - cross
  - dot
  - normalize
  - length
Compiling

- The easiest thing to do:

  - Put your new programs in a subdirectory of:
    - `simtrax/samples/src`

  - Copy an existing Makefile in to your directory

  - Run make
Writing to the Framebuffer

- See mandelbrot sample
- `int start_fb = GetFrameBuffer();`
  - “pointer” to framebuffer
  - Main memory pointers are just integers (untyped)
  - No array indexing! []

storef(red, start_fb + (i * xres + j) * 3, 0);
storef(green, start_fb + (i * xres + j) * 3, 1);
storef(blue, start_fb + (i * xres + j) * 3, 2);

3 floats per pixel
Simtrax Configuration

--num-icaches
--num-icache-banks

Bank 0
Instruction Cache
Bank 1

Thread PC RF Stack RAM
Thread PC RF Stack RAM
Thread PC RF Stack RAM

FUs
- Int Add
- FP Mul
- FP Inv
- ...

--num-thread-procs
--config-file
Analysis

- What is holding performance back?

- For now, this is mostly determined by
  - “Issue Statistics”
  - “Module Utilization”
Issue Statistics

- Issue Rate:
  - Want this as high as possible

- iCache conflicts
  - Tracked separately from “resource” conflicts

- thread*cycles of resource conflicts
  - FU conflicts, L1, L2, DRAM

- thread*cycles of data dependence
  - Nothing you can do about this (for now)
  - Includes RF conflicts
Default Areas (square mm)

- FPADD: 0.003
- FPMIN: 0.00072
- FPCMP: 0.00072
- INTADD: 0.00066
- FPMUL: 0.0165
- INTMUL: 0.0117
- FPINV: 0.112
- CONV: 0.001814
- BLT: 0.00066
- BITWISE: 0.00066
Module Utilization

FP AddSub: 41.41
FP Compare: 12.63

...  

- Does not show every module!
  - Only the large ones
  - You will likely need to increase the number of smaller units as well (Bitwise for ORI, BLT for branches)

- A utilization of 40% is pretty high, try to drop it down
Instruction Caches: Multiple Banks

- Address → bank mapping is strided

- Bank 0
  - Instruction Cache
  - Bank 1

  ✔ Thread a: PC = 0
  ✔ Thread b: PC = 1

- Bank 0
  - Instruction Cache
  - Bank 1

  ✔ Thread a: PC = 0
  ✗ Thread b: PC = 2 (bank conflict!)
Instruction Caches: Multiple Caches

- Consumes much more area, but potentially reduces conflicts

- icache 1 ✔ Thread a: PC = 0

- icache 2 ✔ Thread b: PC = 2
Instruction Caches

- Instruction caches are actually “double pumped”

- Each bank can service 2 requests every cycle