Minecraft 4 Feedback

• Looks good!
Platformer

- A game that minimally involves platforms
- Not based on any game in particular
  - Super Mario 64?
  - Team Fortress 2?
- Completely up to you to make unique gameplay
Breakdown

• Week 1 (collision debugger)
  – Raycasting II
  – Collision detection

• Week 2
  – OBJ loading
  – Collision response

• Week 3
  – Pathfinding
QUESTIONS?
LECTURE 7

Non-Voxel Collisions
Non-Voxel Collisions

MOTIVATION
To Review

• Entity movement models are dependent on collisions

• Collision are different for:
  – Entity-entity collisions
  – Entity-environment collisions

• All of this controls player movement
Voxel is nice...

- AABB + block based world makes it easy to:
  - Collide
  - Raycast
  - Manipulate the world

- Great for a number of gameplay aesthetics:
  - World generation/exploration
  - Construction
...but not always great

- What if I want:
  - Slopes/ramps/curved surfaces
  - Non 90 degree angles
  - Environment objects of varying size

- In Minecraft, some of these issues can be solved with mods, some can’t
What do we really want?

• Arbitrary environment representation
  – Not restricted to a grid or size
• Arbitrary shapes in that environment
  – Allow for sloped surfaces
  – Allow for approximated curved surfaces
• We want TRIANGLES!
• What shape should entities be?
Shape: AABB

- **Pros:**
  - Simple collision test for axis-aligned worlds

- **Cons:**
  - Entities don’t have same diameter in all directions
  - Complicated collision test for arbitrary worlds
  - Entities “hover” on slopes
  - Stairs need special handling
Shape: Cylinder

• Pros:
  – Entities have same diameter in all directions

• Cons:
  – Collisions even more complicated by caps
  – Same slope hover problem
  – Same stairs problem
Shape: Upside-down cone

• **Pros:**
  – Entities don’t hover on slopes
  – Entities naturally climb stairs (kinda)

• **Cons:**
  – Still more complicated collision tests
  – Sliding like this may be undesirable
Shape: Ellipsoid

• Pros:
  – Simpler collisions than any of the others for arbitrary triangle world
  – Entities closer to the ground on slopes
  – Entities still climb stairs (if they’re low enough)

• Cons:
  – Entities “dip” down a bit going off edges
After Platformer ...

• Environment represented as an arbitrary mesh of triangles
• Entities represented as ellipsoids
• We need to build:
  – A basic mesh representation
  – Ellipsoid-triangle collisions
  – Ellipsoid raycasting
  – Triangle raycasting
  – Navigation through the world
QUESTIONS?
LECTURE 7
Raycasting II
Ellipsoid Raycasting II

ELLIPSOID RAYCASTING
Raycasting a circle

• Before we try 3D, let’s think in 2D
• Ray: position and direction
  • \( \vec{r}(t) = \vec{p} + t\vec{d} \)
  • \( \vec{d} \) is a normalized vector
• Make every circle a unit circle at the origin (simpler to raycast)
  – Translate circle center and ray origin by -(circle center)
  – Scale circle and ray origin and direction relative to radius \((1/r)\)
    • DO NOT RE-NORMALIZE the ray direction vector
• Plug ray equation into equation for unit circle at the origin:
  \[ x^2 + y^2 = (\vec{p}.x + \vec{d}.x*t)^2 + (\vec{p}.y + \vec{d}.y*t)^2 = 1 \]
• \( t \) is the only real variable left, solve with quadratic formula
  – \( t \) gives you the intersection point for both the unit circle with the transformed ray, and the original circle with the untransformed ray
    • Because we haven’t re-normalized the direction

Raycasting a Sphere

• Unit sphere at the origin: \( x^2 + y^2 + z^2 = 1 \)
  – Same transformations to both sphere and ray
• Same ray equation (3 components)
• Solve for \( t \):
  – Calculate discriminant \( b^2 - 4ac \)
    • \(< 0\) means no collision (no real roots to quadratic)
    • \(= 0\) means one collision (one root, ray is tangent to sphere)
    • \(> 0\) means two collisions (two roots)
• Plug \( t \) into ray equation to get 3D intersection
Raycasting an Ellipsoid
• Sphere intersections are way easier than ellipsoid intersections
• Squish the entire world so the ellipsoid is a unit sphere!
  – Do detection in that space, convert back
• Change of vector spaces:
  – Ellipsoid radius $R = (r_x, r_y, r_z)$
  – Use basis $(r_x,0,0)$, $(0,r_y,0)$, $(0,0,r_z)$
  – Ellipsoid space to sphere space: component-wise division by $R$!
Raycasting an Ellipsoid

• Convert from ellipsoid space to unit sphere space
  – Don’t forget to transform to origin as well as scale

• Solve sphere equation for the new ray

• Plug $t$ into the original ray equation to get intersection point
Raycasting II – Ellipsoid Raycasting

QUESTIONS?
Raycasting to the environment

• We can raycast to ellipsoids, great
• Need some way to be able to raycast to our environment as well
• This can be used for gameplay like bullets, lasers, line of sight, etc...
• More importantly, you will use this in your sphere-triangle collision detection
Raycasting to the environment

- Our environment is made up entirely of polygons
- All polygons can be decomposed into triangles
  - Even ellipsoids are approximated by triangles when being drawn
- So to raycast the environment, raycast to each triangle, and take the closest intersection
Ray-triangle intersection

- Given: Ray casted from $\vec{p}$ in the direction of $\vec{d}$
  - Ray equation $\vec{r}(t) = \vec{p} + t\vec{d}$
- Goal: find $\vec{x}$, the point on the triangle
- There might not be a point $\vec{x}$ which exists in that triangle
- But there is a point $\vec{x}$ that exists in the plane of that triangle
  - $t$ value might just be negative (the point is in the opposite direction of the ray)
Ray-triangle intersection

- Point $\hat{x}$ on triangle plane if $\hat{n} \cdot (\hat{x} - \hat{s}) = 0$
  - Where $\hat{s}$ is any point on the plane, such as one of the vertices
  - $\hat{n}$ is the normal of the plane
- Set $\hat{x} = \hat{p} + t\hat{d}$
- Solve for $t$ in $\hat{n} \cdot ([\hat{p} + t\hat{d}] - \hat{s}) = 0$
  - That means $t = \frac{\hat{n} \cdot (\hat{p} - \hat{s})}{\hat{n} \cdot \hat{d}}$
Ray-triangle intersection

- So now we know the point P at which the ray intersects the plane of the triangle
  - But is that point inside the triangle or outside of it?
- Point P (on plane) is inside triangle ABC iff P is on the left of all of the edges (assuming that edges are defined in counter-clockwise order i.e. $\overrightarrow{AB}, \overrightarrow{BC}, \overrightarrow{CA}$)

P is to the left of $\overrightarrow{AB}, \overrightarrow{BC}, \overrightarrow{CA}$

P is to the right of $\overrightarrow{CA}$
Ray-triangle intersection

• A point $P$ is to the left of edge $AB$ if the cross product $AB \times AP$ is in the same direction as the triangle normal $-BC \times BP$, and $CA \times CP$ are the other cross products.

• Can calculate normal of a triangle with cross product of two of its edges

$$N = (B - A) \times (C - A)$$

• Now you can compare to see if two vectors are in the same direction by seeing if their dot product is positive

$$(AB \times AP) \cdot N > 0$$
Triangle Raycasting

QUESTIONS?
Lecture 7
Collisions III
The basics

• Entity represented by an ellipsoid
• World represented by a set of triangles
• Continuous collision detection
  – Analytically compute the time of and point contact, translate object to that point
  – What we did for the voxel engine
• Basic idea: formulate motion of the entity as a parametric equation, solve for intersection
  – Only works for simple motion (straight lines)
General algorithm

- Compute the line the player follows in one update
  - Kinda like raycasting start position to end position
- Do ellipsoid-triangle sweep test for all triangles and take the closest result
  - Can optimize this using spatial acceleration data structure to test relevant triangles
  - Closest indicated by smallest positive $t$ value (proportion of update taken resulting in collision)
- Compute remaining translation, sweep again
  - Cut off after a certain number of translations
  - You’ll do this next week
WARNING

• There is A LOT of vector math we’re about to get into
• You DO NOT need to understand all of it
  – Though it may help with debugging
• This is not a math class
  – Don’t memorize the derivations
  – Don’t re-invent the wheel
Collisions III

ELLIPSOID-TRIANGLE COLLISIONS
Ellipsoid-triangle collisions

• Analytic equation for a moving sphere:
  – Unit sphere moving from $A$ at $t = 0$ to $B$ at $t = 1$
  – Location of center: $A + (B - A)t$
  – Point $P$ on the sphere at $t$ if $||[A + (B - A)t] - P||^2 = 1$

• Solve for $t$ in unit sphere space
  – Value stays the same in ellipsoid space!

• Split collision detection into three cases:
  – Triangle interior (plane)
  – Triangle edge (line segment)
  – Triangle vertex (point)
Sphere-interior collision

- Intersect moving sphere with a plane
- If intersection is inside triangle, stop collision test
  - Interior collision always closer than edge or vertex
- If intersection is outside triangle, continue test on edge and vertices
  - NO short circuit
Sphere-interior collision

- **Sphere-plane intersection:**
  - Same thing as ray plane using the point on the sphere closest to the plane!
  - Given plane with normal $N$, closest point is $A - N$
    - We assume that the sphere starts “above” the triangle
    - Don’t care about colliding a sphere starting below the triangle, this should never happen
Sphere-interior collision

- Point P on plane if
  \[ N \cdot (P - S) = 0 \]
  - Where S is any point on the plane, such as one of the vertices
- Set \( P = (A - N) + (B - A)t \)
- Solve for \( t \) in
  \[ N \cdot ((A - N) + (B - A)t) - S) = 0 \]
  - That means
    \[ t = -\frac{N\cdot(A-N-S)}{N\cdot(B-A)} \]
- This says when the sphere hits the plane
  - May not be in the triangle!
  - Repeat your point-in-triangle test!
QUESTIONS?
Sphere-edge collision

• Sphere vs. edge is the same as sphere vs. line segment
  – Intersect moving sphere with the infinite line containing the edge
  – Reject intersection if it occurs outside the line segment

• How do we collide a moving sphere with a line?
  – Really just finding when sphere center passes within 1 unit of line
  – If we treat the line as an infinite cylinder with radius 1, and the motion of sphere center as ray we can use ray-cylinder intersection
Analytic sphere-edge collision

- Area of parallelogram formed by two vectors is the length of their cross product.
- Defining the surface of an infinite cylinder with vectors:
  - Given two points $C$ and $D$ along cylinder axis.
  - Point $P$ on surface if $\| (P - C) \times (D - C) \|^2 = \|D - C\|^2$
    - $\|D - C\|$ is area of gray parallelogram.
    - $\| (P - C) \times (D - C) \|$ is area of green parallelogram.
    - Area of parallelograms is equal.
    - This means that their height is equal, which means that the distance of $P$ to the line segment is 1.

Green parallelogram area is equal to gray rectangle area if $P$ is on cylinder surface.
Analytic sphere-edge collision

• Set \( P = A + (B - A)t \)
• Substitute into previous equation:
  \[ \|([A + (B - A)t] - C) \times (D - C)\|^2 = \|D - C\|^2 \]
• Solving for \( t \), you get a quadratic \((at^2 + bt + c = 0)\) where
  \[ a = \|(B - A) \times (D - C)\|^2 \]
  \[ b = 2((B - A) \times (D - C)) \cdot ((A - C) \times (D - C)) \]
  \[ c = \|(A - C) \times (D - C)\|^2 - \|D - C\|^2 \]
• Solve using quadratic equation, use lesser \( t \) value
Analytic sphere-edge collision

- Discard intersection if not between C and D
  - Will be handled by vertex collision test
- To check if intersection is between C and D:
  - Get vector from C to intersection point P
    \[ P - C \]
  - Project this vector onto cylinder axis
    \[ (P - C) \cdot \frac{D - C}{\|D - C\|} \]
  - Check if projection is in the range \((0, \|D - C\|)\)
    \[ 0 < (P - C) \cdot \frac{D - C}{\|D - C\|} < \|D - C\| \]
  - Optimized by multiplying by \(\|D - C\|\):
    \[ 0 < (P - C) \cdot (D - C) < \|D - C\|^2 \]
 QUESTIONS?
Analytic sphere-vertex collision

- Collision test against a triangle vertex $V$
- How do we collide a moving sphere against a point?
  - We know how to do a ray-sphere intersection test
  - Moving sphere vs. point is equivalent to sphere vs. moving point
    - Where the point moving in opposite direction
Analytic sphere-vertex collision

- Point P on sphere if $\|P - A\|^2 = 1$
  - Set $P = V - (B - A)t$
  - Solve $\|[V - (B - A)t] - A\|^2 = 1$ for $t$

- Looks like $at^2 + bt + c = 0$ where
  
  $a = \|B - A\|^2$
  $b = -2(B - A) \cdot (V - A)$
  $c = \|V - A\|^2 - 1$
QUESTIONS?
LECTURE 7
Tips for Platformer 1
Tips for Platformer 1

COLLISION DEBUGGER
“No, I don’t need a debugger”

- Physics/collision bugs are the hardest type of bugs to track down
- It will be much easier for you to find your mistakes in a controlled environment than for you to make them in your own code
- It’s easier to test to make sure you’ve done it correctly
How does it work?

• You can move around two of the ellipsoids here
  – The green ellipsoid represents an entity at the beginning of the tick
  – The red ellipsoid represents an entity at the end of the tick (without collision)

• The other two ellipsoids are determined by the placement of the first two
  – The first orange ellipsoid represents where the entity will end up via colliding with the green triangles
  – The second orange ellipsoid represents where the entity slides to after hitting the surface
Where do I put my code?

• We recommend that you put your raycasting and collision code in separate files (besides the support code files)

• In order to change the position of the ellipsoids within the debugger …
  – Modify their positions in the view.cpp paintGL function (at the very top)
Collision Data

• Your collision code should return a struct, minimally containing:
  – t-value in [0,1]
  – Normal
  – Point of contact

• You may want to put “fancier” stuff in later
More stuff

• About two sided triangles …
• Don’t worry about colliding ellipsoids with triangles that they are already “inside”
• Don’t worry about colliding ellipsoids with triangles that they are “below”
C++ Tip of the Week

PARAMETRIZED INHERITANCE
// (Parent varies at compile time)
// call Dad::doThis, Kid::doThat,
// Kid<Parent> f3; f3.doThat();
// the compiler just wrote 3 "Kid" classes for us

## Parametrized inheritance

```
// call Dad::doThis, Kid::doThat,
// Kid<Parent> f3; f3.doThat();
```

- Haven’t really talked about template classes
- Kinda like generics in Java
- But the thing in the parantheses is just text replaced by the compiler when given actual argument
- Can be used for things like double dispatch.
  - Don’t need to cast things for collision callbacks
LECTURE 7
C++ Anti-Tip of the Week
C++ Anti-Tip of the Week

OPERATOR OVERLOADING
Wait, Operator Overloading?

• In C++, you can tell basic operators to work with classes (or enums!)
  – The basic arithmetic operations are commonly overloaded (+, -, *, /)
    • ++, --, <<, and >> are also often overloaded
  
• GLM overloads many operators to make vector math convenient
Operator Overloading

- There are many legitimate uses of operator overloading
- But it can be very easy to misuse it
- In general, only use it to objectively make code clearer (to anyone who reads it)
  - even if `myColor%(BLUE->RED[-7])` makes sense to you
Operator Overloading

• You can even overload the function operator () for classes
  — Then you can call objects of that class like functions
  — But you could just give that class a named function, and call that function from your objects

• You can overload the assignment operator = for classes too
Operator Overloading

• The only operators you can’t overload are:
  :: . (dot) ?: (ternary) sizeof

• Meaning you can overload pretty much everything else:
  % ^ | & ~ > < == ! [] () new -> delete

• [Link to FAQ on Operator Overloading](https://isocpp.org/wiki/faq/operator-overloading)
C++ Tip & Anti-Tip of the Week

QUESTIONS?
PLAYTESTING!
Sign up for Platformer1 Design Checks!