

Game Dev Foundations

(work in progress...)

Tools

<https://openframeworks.cc/about/>

<https://github.com/anael-seghezzi/CToy>

<https://derivative.ca/>

<https://www.pygame.org/wiki/GettingStarted>

1) PLANE MATH:

Creating plane equation with point and normal:

Normal dot (given point - any xyz combination) = 0

Distance from point to line:

$D = |a \text{ cross } b| / |a|$

- Where a is a vector on the line and b is a vector from the point on the line to the point off the line (basically use cross product to find the DIFFERENCE between them).
- Because it is the area of the parallelogram with sides a and b we divide by the length of a to remove the width basically.

Distance from point to plane:

Point distance to plane

$D = n \cdot f$ (where f is a vector from the plane to the point off the plane (point off - point on), use the dot product to find the similarity towards the normal, which is D)

Or

Evaluate with the plane equation:

A handwritten note on lined paper showing the formula for the distance from a point to a plane. The formula is:
$$\text{distance} = \frac{\text{Plane equation} - D \text{ (evaluated w/ the point)}}{\sqrt{A^2 + B^2 + C^2}}$$
 Annotations include: 'Plane equation' with an arrow pointing to the numerator; 'D (evaluated w/ the point)' with an arrow pointing to the subtracted term; 'from plane equation' with an arrow pointing to the denominator; 'not on this plane' with an arrow pointing to the numerator; and 'from equation for plane (COUNT THE 1s!!)' with an arrow pointing to the denominator.

Plane containing point parallel to a plane:

Take normal and dot it with $x-x_0$, $y-y_0$, $z-z_0 = 0$ where the x_0 is the point not on the plane to solve for D for the other plane. Basically cause the dot product has to be zero if it is parallel.

How to tell if point is on one side or the other:

Take the dot product of the vector of (the plane to the point off the plane) with (the normal vector of the plane) and if it is > 0 it is the same side as the normal, less than 0 it is on the opposite side of the normal (0 it is actually on the plane)

Raw plane intersection:

$$R(t) = P + t\mathbf{d}$$

To solve for the intersection of ray $R(t)$ with the plane, we simply substitute $\mathbf{x} = R(t)$ into the plane equation and solve for t :

$$\begin{aligned}\mathbf{n} \cdot R(t) &= d \\ \mathbf{n} \cdot [P + t\mathbf{d}] &= d \\ \mathbf{n} \cdot P + t\mathbf{n} \cdot \mathbf{d} &= d \\ t &= \frac{d - \mathbf{n} \cdot P}{\mathbf{n} \cdot \mathbf{d}}\end{aligned}$$

Then plug back into $R(t)$ and solve for the distance (d there being the direction)

Distance between two parallel planes:

Find a point on one of the planes then do point distance to plane formula (cause they don't intersect)

PLANE/PLANE && PLANE/LINE INTERSECTIONS:

Plane/Plane Intersection:

Plane parallel, perpendicular, or angle between:

- *Parallel*: have the same exact normal scaled by D
- *Perpendicular*: The dot product of their normals is 0
- *Intersection*:
 - Find angle between two planes, or point of intersection. POI is just plug one into the other
 - Angle: $\cos(\theta) = \mathbf{n}_1 \cdot \mathbf{n}_2 / \|\mathbf{n}_1\| \|\mathbf{n}_2\|$ (don't need the length division if it is unit length)

- *Identical*: If the scalar d is scaled by the same amount of the rest of the equation to equal the same d proportionally as if parallel.

Line of intersection:

- 1) Get point: Solve for a variable by setting one variable to zero, then plug it into the other equation so solve for one. Then plug back into others to get a point.
- 2) Take normals of both planes and cross product them to get the normal of the line (cause this sticks out along where the planes intersect)
- 3) Parametric: Take the point then add t times the normal to it to get the parametric equation
- 4) Symmetric: solve for t in the parametric
- 5) Angle between: use angle formula to solve for theta with the two normals:
 - a) $\theta = \cos^{-1}((a \cdot b) / (|A| |B|))$

Parametric form:

Start with a point, then treat x,y,z all as a variable scalar t, which will be along the line you are given basically with the relative equation between them.

Found by having a normal and multiplying it by t then adding a point to that.

Symmetric form:

Solve for t, so each of the variables are set equal to each other given any x,y,z.

Both parametric and symmetric form start with a point $\langle P_x, P_y, P_z \rangle + t \langle \text{Normal (or slope, normal if this is line of intersection...)} \rangle$

Plane/Line Intersection:

Plug in the x, y, z t values into the equation of the plane and solve for t

- On Plane = when t can be 1 to leave you with a number equaling another number
- Parallel = when t is removed and you get a number that isn't equal to another number
- Intersection = when you get a t value equals a number, plug that back into equation to find x,y,z

[link](#)

Line/Line Intersection:

Distance between two lines:

Given this point:

$$\vec{x}_1 = \vec{q} + r\vec{u} \text{ and } \vec{x}_2 = \vec{q} + s\vec{v},$$

You take the cross product (because the closest point must be perpendicular to both lines), and the dot it with a line connecting the two lines to determine the length of that connection distance relative to that closest perpendicular point.

$$d = \left| (\vec{p} - \vec{q}) \cdot \frac{\vec{u} \times \vec{v}}{|\vec{u} \times \vec{v}|} \right|$$

Line/Point Distance:

Given a point \vec{P} a line $\vec{x} = \vec{r} + s\vec{u}$

$$\vec{u}_0 = \frac{\vec{u}}{|\vec{u}|}$$
$$d = \left| \vec{u}_0 \times (\vec{r} - \vec{P}) \right|$$

Take distance of the line normalized times the starting part of line r minus P to represent a line from the line to the point, then take the cross product to find the difference length.

[link](#)

Point to point on Sphere (Spherical distance)

$$d = \cos^{-1} (\mathbf{P} \cdot \mathbf{Q}),$$

2) VECTOR MATH

Law of Sines:

<http://xaktly.com/MathNonRightTrig.html>

The law of sines can be used to find the measure of an angle or a side of a non-right triangle if we know:

- two sides and an angle not between them or
- two angles and a side not between them.

$$\frac{\sin(A)}{a} = \frac{\sin(B)}{b} = \frac{\sin(C)}{c}$$

Law of cosines:

Opposite side squared = adjacent squared + hypoteneuse squared MINUS 2 times adjacent times hypoteneues times cosine theta

The law of cosines can be used to find the measure of an angle or a side of a non-right triangle if we know:

- two sides and the angle between them or
- three sides and no angles.

$$a^2 = b^2 + c^2 - 2bc \cdot \cos(A)$$

$$b^2 = a^2 + c^2 - 2ac \cdot \cos(B)$$

$$c^2 = a^2 + b^2 - 2ab \cdot \cos(C)$$

Dot product:

Each component multiplied together of two vectors, or length of a times length of b times the angle between them. Measure of how similar two vectors are.

Angle between two vectors:

$$(a \cdot b) = ||a|| ||b|| \cos(\theta)$$

To find the dot product of two vectors: Multiply each component of the vector by the component of the other vector

- Derived using law of cosines because the length of a vector squared is the same as the dot product of a vector with itself (because getting the length of a vector is the same as a dot product squaring)
- Derivation is by taking $a^2 + b^2 - 2ab \cos \theta$ to find that third side, then you square it and whittle down the right hand side until all you are left with is an $a \cdot b$ on the left and no squares on the right. Its just beautiful

Cross product:

- The scale of this vector represents the difference between two vectors rather than the similarity (in the case of the dot product).
- The direction represents a line orthogonal to the two given lines.
- Largest when two vectors crossed are orthogonal.
- Perpendicular vector between two vectors, or vector that favors neither vector by being evenly between them, basically the vector that goes sideways perpendicularly into the dimension neither vector shares.
- Magnitude is the area of the parallelogram between the two vectors, so divide by the side you want to negate and you get the difference between the two (see point distance to line)

Length of the cross product of a and b = length of a * length of b * $\sin \theta$

To find: line up sideways:

$\begin{vmatrix} x & y & z \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}$

Then multiply right crosses minus left crosses (both starting with the upper left letter) across down from the letters on top (imagine the grid has two more of itself on both sides):

$$(y_1 z_2 + z_1 x_2 + x_1 y_2) - (x_1 z_2 + y_1 x_2 + z_1 y_2)$$

Triple Product:

Dot product of one vector with cross product of other two. Gives you a parallelepiped box.

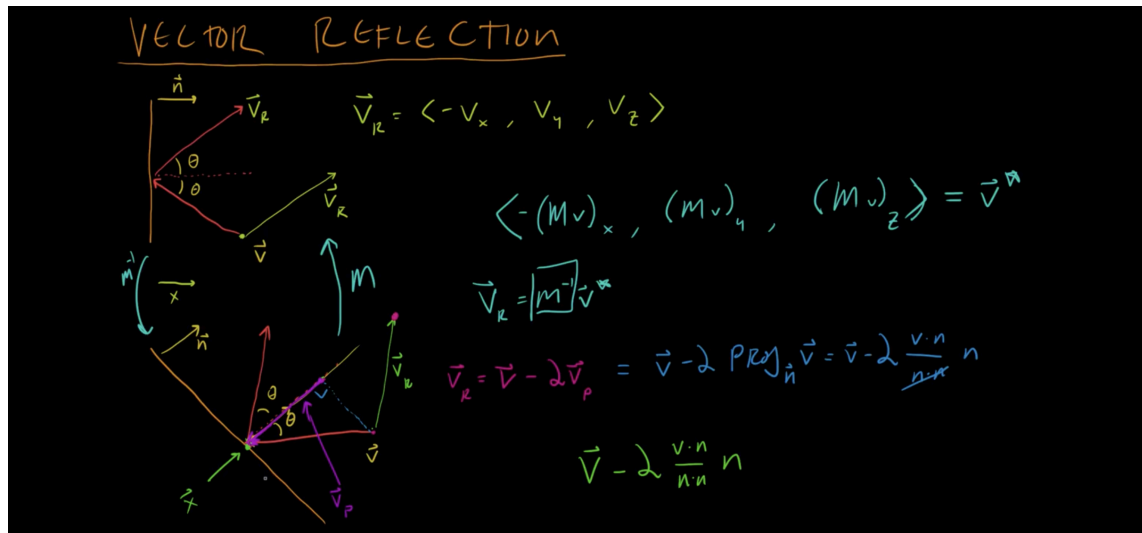
Vector Projection (onto normal):

$N \cdot (V_1 \cdot N / |N|)$ - or basically the normal direction times the dot product projection (or length)

Vector Reflection:

$$R_r = R_i - 2N(R_i \cdot N)$$

Orthogonal projection onto normal times two subtracted from the end of the approach gives the vector starting at the approach point going in reflected direction (then start at x - the impact point - and add that vector)



Magnitude:

Vector size: $\sqrt{x^2 + y^2 + z^2}$

Normalized:

Standard out of 1, each component of a vector gets divided by its magnitude

Quaternions:

A rotor that represents a rotation about an axis by a given scalar, effective through two vector reflections

- Multiplication: need to multiply from left to right (q times d = r)
- Can be smoothly interpolated (as opposed to euler angles where order can rotate things unexpectedly)
- Avoid gimbal lock (when trying to interpolate with euler angles on multiple axis's and the rotations affect each other.

<https://www.youtube.com/watch?v=eo2HNCTV78s>

<http://marctenbosch.com/quaternions/>

SLERP:

Is a spherical linear interpolation. The interpolation is mapped as though on a quarter segment of a circle so you get the slow out and slow in effect. The distance between each step is not equidistant. SLOW IN SLOW OUT

LERP:

Is a linear interpolation so that the distance between each step is equal across the entire interpolation. Like a weight towards the other value that slowly progresses there given the current distance.

How to properly lerp two vectors:

$a = \text{lerp}(a, b, 1 - f^{\wedge} dt)$

- This guarantees it never overshoots and maintains parity despite framerate
- If F is .25 then the interpolation value is 75% per second...

How to choose: SLERP vs LERP:

If you have a normalized vector that you want to interpolate to a different direction, USE SLERP. Slerp will rotate it spherically around the unit circle (maintaining a length of 1 - whereas lerp will basically take a shortcut and give you a non uniform vector...)

Gimbal Lock:

When two of the axes line up during a linear interpolation some of an axis's rotation will be lost or the order will mess up the way it goes. So you use quaternions to avoid rotations affecting each other.

Euler Angles:

(with z going forwards/backwards)

$$V_x = \sin(\text{yaw})\cos(\text{pitch})$$

$$V_y = \sin(\text{pitch})$$

$$V_z = \cos(\text{yaw})\cos(\text{pitch})$$

- Cosine pitch cause that is the roll effect (as pitch goes up that will go down)

Represent a direction in the form of a pitch yaw and roll.

3) PHYSICS

Moving Object:

$$s = ut + (1/2)at^2$$

S is position

U is velocity at the start

A is acceleration

T is time

(if acceleration is constant)

kinematic:

The Kinematic Equations

$$d = v_i \cdot t + \frac{1}{2} \cdot a \cdot t^2 \quad v_f^2 = v_i^2 + 2 \cdot a \cdot d$$

$$v_f = v_i + a \cdot t \quad d = \frac{v_i + v_f}{2} \cdot t$$

<https://www.physicsclassroom.com/Class/1DKin/U1L6b.cfm>

GRAVITY = 9.8 m/s

4) MATRICES

Transpose:

Swap the columns with the rows - when multiplied by the matrix gets you back to the identity if the matrix is orthogonal. AND orthogonal if its transpose is equal to its inverse.

Inverse:

Rotation -> Take its transpose

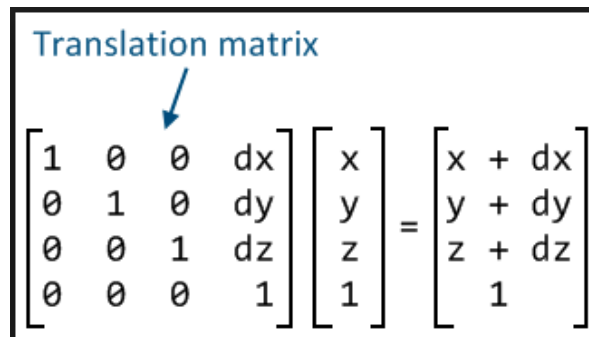
Translation -> Set each translation value to negative

Scale -> set to 1/each value

Used to transform a point into another's coordinate space. If you have world space, multiply by inverse transform of an object to get point in its space. If you have point in object's space, multiply by transform to get back to world.

[Inverse](#)

Translation Matrix:


$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + dx \\ y + dy \\ z + dz \\ 1 \end{bmatrix}$$

- Inverse is just -dx, ect...

Rotation Matrix:

[Link](#)

This basically rotates the coordinate system and then takes the dot product on each axis with the given point to represent the point's new length along each of the axes. Notice how when rotating about an axis that axis is left alone (by just multiplying by 1) and the rotation occurs in the other two.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- For each of these you are basically taking the coordinates of each of the vectors (each row going down for each axis) rotated on a unit plane given the length relative to theta...

Rotate about a point: multiply by transform matrix that gets it back to 0,0,0 -> then rotate the object, then transform to the point's space. In order: Inverse of pointToOrigin * rotation * matrixToOrigin.

ROTATIONS ALWAYS NEED TO ROTATE ABOUT THE ORIGIN (because you are rotating the coordinate space, so when you rotate a matrix you are also moving the object if it is not at the origin)

Rotation Inverse:

Same thing as the transpose (swapping the rows and columns)

Scale Matrix:

m_{00}	0	0
0	m_{11}	0
0	0	m_{22}

- Also scales based on the origin (so it will move it away from the origin, moving it even further if you rotated)
- Inverse scaled matrix is just 1/mn

Multiply, Scale & Rotate Matrices (Affine Transform):

[link](#)

[another link](#)

[rotate around point](#)

Used as 4x4 where the bottom right column is 0 when you don't want to translate and 1 when you do want to. Set to 0 to vectors because that is just direction and then set to 1 for points cause those need to be translated. Setting to zero because its dot product multiplication math, and so that would change the translation by zero.

How to Multiply matrices

Transform is in the 4th row, set to 1 on the bottom of 4th row (three to the right always zero) then just have the rest be an identity matrix.

of columns have to equal the # of rows...

Multiply each Column on the left by each row on the right in dot product notation (adding the products together). ORDER MATTERS (want the last one to be applied to be the farthest to the left) The result is limited by the smallest number of columns/rows. When multiplying: take the column from the first matrix and row from the second.

- EACH coordinate is the sum of the entire column from the left dotted with the entire row from the right given that coordinates column/row.

Matrix Reflection:

Multiply the axis the rotation is going over to -1, or multiply multiple axes to -1 if it is flipping across multiple axes.

Transform matrix:

- X for local to world
- If multiplied by a point, takes that point out of the transform it is relative to.

Tells you how to translate from the origin to the space of the object. (used on each vertex too because vertex knows its local coordinates but in the end it is multiplied by the matrix)

[matrix combination transforms](#)

INORDER:

- 1) Scale
- 2) Rotate
- 3) Translate

TRSv

(multiplied out)

FIRST OPERATION GOES LAST THOUGH MATHEMATICALLY (SO you actually start with translation, then times it by rotation, then times that by scale) - even though in practice that means it starts scaling at the origin.

"Put the first operation last and you get the matrix you want"

Inverse Transform Matrix:

- X for World to Local
- If multiplied by a point, takes that point into the local space of the inverse transform.

<https://www.youtube.com/watch?v=onSyW44OnxA>

Order is other way around times the inverses $S^{-1}R^{-1}T^{-1}v = M^{-1}$

5) MISC MATH

<https://www2.clarku.edu/faculty/djoyce/trig/identities.html>

SYSTEM OF EQUATIONS:

$$\begin{aligned} I_x &= t \cdot 2 \\ I_y &= t \cdot \frac{1}{2} \\ I_y &= -3I_x + 11 \end{aligned}$$

3 unknowns, plug the first two into the last (cause the first two are both in terms of t) to group those terms. Find a way to evaluate ONE term within an equation given definitions for other variables. Basically which equation contains the others, and then do the others share a similar unknown that you can then solve for...

6) OTHER WAYS TO COUNT

Conversion to decimal - Add them all up:

128	64	32	16	8	4	2	1
1	0	0	1	1	0	1	1

Conversion to Hex:

Decimal	Hex	Binary	...	Decimal	Hex	Binary
00	0	0000		08	8	1000
01	1	0001		09	9	1001
02	2	0010		10	A	1010
03	3	0011		11	B	1011
04	4	0100		12	C	1100
05	5	0101		13	D	1101
06	6	0110		14	E	1110
07	7	0111		15	F	1111

Double Check: <https://www.mathsisfun.com/binary-decimal-hexadecimal-converter.html>

ANSWER: For -2 change to binary: 00000010 take 2's complement: 11111110=FE(hex)

2's complement: swap every bit then add 1

Signed bit: In the most significant location (1 means negative 0 means positive, F means negative if hex cause F is all 1s)

Two's complement:

Use first bit as negative flag, then number minus that bit is what is left (e becomes 5 if bitflag is 8)

https://en.wikipedia.org/wiki/Two%27s_complement

<https://www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html>

<http://faculty.cs.niu.edu/~byrnes/csci360/notes/360numsys.htm>

<< left shift
>> right shift
NOT ~ (flips all the bits)
AND & (true if both bits are 1)
OR | (true if any of the bits are 1)
XOR ^ (1 if one is true and one is false, or even trues 0 odd trues 1)GOOD FOR CHECKING IF BITS ARE DIFFERENT.

7) DATA STRUCTURES

- 1) Linked List: has a value and a pointer to the next element in the list, Double linked list has a pointer back to previous
- 2) Stack: LIFO, put items on the top then pop them off to get lower down.
- 3) Queue: FIFO, first one in is the first one to be popped off
- 4) Set: No order but no repeated values -> functions for merging, or finding difference between sets
- 5) Map: Key Value
- 6) Hashtable: takes an object and converts it to a number with hash function to use on a map - can chain but idea is to have few collisions
- 7) Binary Search: Each node's left descendants are less than it and the right descendants are greater than it. So you can just check if greater than or less than and keep going down.
- 8) Trie: A tree where each node contains a letter and branches to infinite number of nodes and a boolean indicating whether it is the end. Used to spell words where the end is a complete word based on previous nodes
- 9) Binary Heap: Good for sorting min/max overall but if you need to find specific values use a binary search tree. Tree is structured with the greatest on the left or the least on the left, so it is easy to access the biggest/smallest but not search for particular values...
- 10) Graphs: can be represented with an adjacency matrix, with columns and rows that have 1's or 0's if those columns go to those rows, therefore showing both directions. Searched by breadth or depth first...
 - a) Directed - both ways
 - b) Undirected - just a link

8) SEARCH ALGORITHMS

Binary Tree vs Binary Heap:

Tree has each value being < or > on the way down, whereas heap is ordered with min/max on upper left, and partitioned each insert to maintain that.

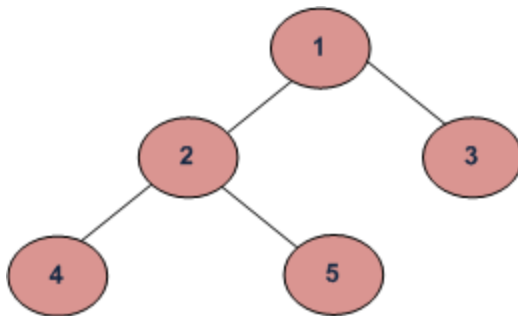
Breadth first vs depth first search:

- If you know a solution is not far from the root of the tree, a breadth first search (BFS) might be better.
- If the tree is very deep and solutions are rare, depth first search (DFS) might take an extremely long time, but BFS could be faster.
- If the tree is very wide, a BFS might need too much memory, so it might be completely impractical.
- If solutions are frequent but located deep in the tree, BFS could be impractical.
- If the search tree is very deep you will need to restrict the search depth for depth first search (DFS), anyway (for example with iterative deepening).

Breadth: Level order: 12345

Depth:

- Inorder (Left, Root, Right) : 4 2 5 1 3
- Preorder (Root, Left, Right) : 1 2 4 5 3
- Postorder (Left, Right, Root) : 4 5 2 3 1



9) SORT ALGORITHMS

- Selection Sort: Start at the left, find the least (or greatest) amount then put it at the beginning, then start one index over and do the same thing, then move another index until you reach the end.
- Bubble sort: continue looping through swapping the index with the next if ($>/<$). Keeps looping through until it does a pass where there were no indexes that needed swapping (cause otherwise it wouldn't know its done)
- Insertion sort: When number is ($>/<$) number behind it, loop through numbers behind it bumping them all one index until it is no longer ($</>$) then set at the empty index of what got bumped up.
- Quick sort: keeps a pivot index that represents the last number it moved down, then continues up the array and swaps all the way back to the pivot by comparison, or further if it has to.
- Merge sort: goes through splitting the array into sections until there is only one in each container then combines them based on which are greater than/less than.

- Heap sort: data organized into a complete binary tree where heapify procedure is called going down the tree to maintain the highest at the top of the tree then rearrange the low nodes.
- Tree sort: traverses a tree to sort it according to BST with each sub node being > on the right and < on the left...

https://www.youtube.com/playlist?list=PL9xmBV_5YoZOSbGAXAP1q1BeUf4j20pI

Big O

Largest constraint only
Constants dropped

N
Nlog(n) - find then sort
N² more expensive than nlogn

10) C++

Std: Standard library - lists, vectors, smart pointers...
Stl: Standard template library - sets, queue, stack, ect...

Unique_ptr - can only be one
Shared_ptr - can be multiple
Weak_ptr - doesn't hold strong reference

& vs && (rValue vs lValue, rValue "&&" means that the variable can't exist outside scope of function param)

<https://www.geeksforgeeks.org/pointers-vs-references-cpp/>

Heap (Pointers, "new" keyword when declaring object)- dynamic memory available to the application, where new objects are created and stored, needs to be managed and objects need to be deleted in C++ otherwise the memory will get out of control. Need to be malloc and deleted or freed...

Overall -> pointers themselves are allocated on the stack, the objects they reference are allocated on the heap... Member variables of the heap allocated objects are also on the heap...

Stack vs Heap:

Stack (just variables)- Static memory allocation, fixed size available to applications (not reference types). Created within functions or as variables that are automatically destroyed when the scope of that function ends.

Compilation Stages:

Preprocessing - removes macros, puts files where they need to be.

Compilation - converts raw c++ files to first assembly and then machine (bitcode). Useful to only recompile what u need rather than link every step of the way

Linker - takes definitions from other libraries and assures they are linked to the code. Replaces references to undefined objects with actual memory addresses.

11) GAME TECH

Mipmap:

Image with different resolutions built into the file so that it can be rendered at different distances from the same texture. Providing LOD within a single file. Stored as a pyramid with the images getting smaller by 2^n . So stored as 256 pixels then goes to 128, 64, 32, ect...

Anti-aliasing Techniques:

FXAA: Smooths edges of pixels on the screen, side effect is it has some edge reduction (because they are smoothed)

MSAA: Blows up image and renders it bigger than it needs to be then shrinks. Only works for forward rendering because all that light effect to be included needs to happen before.

Virtual function:

Managed with a VTable, which stores addresses to the functions for each of the inherited versions of the functions or address to the base class if it isn't overridden.

Early vs late binding:

Early for static types where definitions are bound at compile time where as late binding allows for polymorphic lookups at runtime dynamically.

Game Loop:

- Update loop takes into account lag - time between last frame, to determine how many updates simulation needs to take.
- Updates at a fixed time step while less than a certain amount of lag so that floating point precision isn't affected.
- Subdivides updates in for the physics state between update steps if CPU faster than GPU
- Then renders and regardless of processor speed results should be deterministic.

Tunneling:

Fast moving objects shooting through others without triggering collision cause moving too fast with physics simulation: <https://www.aorensoftware.com/blog/2011/06/01/when-bullets-move-too-fast/>

Update with delta time based on how long the render/update loop is taking to make sure bullets take the same amount of time and are simulated correctly. Use lag to update physics step by step and account for small floating point changes. So you use a small fixed update on the physics step that guaranteed will be smaller than the timestep with a render and update of physics, but it also needs to be bigger than the time it takes to update physics. Then you pass (lag / fixed timestep) to determine delta time for rendering and to be able to render between updates (because render is called and while it's doing that the object should be in between the current update and the next)

https://gafferongames.com/post/fix_your_timestep/

Do this by accumulating the render frames and then having a set dt that consumes that accumulated time, and then interpolate any remaining accumulated time smaller than the timestep into the next frame before rendering again.

EQS query: for AI to find and understand its environment

Reciprocal collision avoidance: <http://gamma.cs.unc.edu/ORCA/>

Voxel: like a pixel, but a volume in 3d.

Networking

https://gafferongames.com/post/reliable_ordered_messages/

Udp versus tcp:

Udp doesn't require order, older packets need to be dealt with when more recent come in..
Better for mp though cause resending lost packets in tcp is slow

Latency versus throughput vs bandwidth

Latency: delay

Throughput: actual amount of data transferred (some lost, or not fully utilized)

Bandwidth: max width of pipe for data.

12) PRACTICE PROBLEMS

1. You are given a projectile launch location, its launch speed, as well as a target's current position and its' current velocity. Write a function that will return the velocity of the projectile that you estimate will hit the target. State any assumptions you have to make.

```
using UnityEngine;
using System.Collections;

public class SurviosTestSpec1 : MonoBehaviour {

    void Start () {
        // example call
        Debug.Log("Shoot projectile at this velocity: "
            + FindVelocityToHitTarget(new Vector2(0,0), 2f, new Vector2(10,0), new Vector2(0,1f)));
    }

    Vector2 FindVelocityToHitTarget(Vector2 pPos, float pSpeed, Vector2 tPos, Vector2 tVelocity) {

        float sDistance = (tPos - pPos).magnitude;
        float relativeSpeed = tVelocity.magnitude / pSpeed;

        // relative speed and starting distance determines target distance they can meet
        float tDistance = sDistance * Mathf.Tan (Mathf.Asin ((relativeSpeed)));
        // just nice to know
        Debug.Log ("Time to hit target: " + tDistance / tVelocity.magnitude);

        // the angle the projectile needs to intersect at that distance
        float theta = Mathf.Atan (tDistance / sDistance);
```

```

        // return velocity
        Vector2 pDirection = new Vector2 (Mathf.Cos (theta), Mathf.Sin (theta));
        return pDirection * pSpeed;
    }
}

```

My assumptions are that this is in 2d space, the starting position of the target is at a 90 degree angle relative to the starting position of the projectile, the speed of the projectile is greater than the speed of the target, and that the velocities of both objects remain constant (i.e no changing course and no frictional or gravitational forces in play).

What is 2^8 ? Why is this number important?

What is -2 in hex? (Literally just memorize, they don't care if you know how to calculate it)

What is the dot product? How does this relate to angles? What is the cross product?

Given two vectors, how can you tell what direction the normal is facing?

Given a plane, how can you tell which side a point is on?

What is the equation for a plane? ($Ax + By + Cz + D = 0$)

Given this equation, what does A,B,C represent? How can you tell if a point lies on this plane?

Do you know what a Quaternion is?

Do you know what a HashMap is? How do you map a String/Object to a value?

What is a virtual function?

Given a point and a plane, how do you find the distance between the two?

Given a point and a plane, how do you find a line that is parallel to the plane that passes through the point?

Test Prep:

<https://gist.github.com/vasanthk/485d1c25737e8e72759f>

<https://cin.ufpe.br/~fbma/Crack/Cracking%20the%20Coding%20Interview%20189%20Programming%20Questions%20and%20Solutions.pdf>

<https://www.interviewbit.com/courses/programming/>

13) GOOD KNOWLEDGE

Reality is broken

Accelerated C++

c# programming yellow book

<https://swift.org/documentation/>

Learning Game AI By Example

Game Design: A book of lenses

Game Feel

DESIGNER NOTES PODCAST <3

Microinteractions

Rules of Play

Nudge

Everything bad is good for you

Hooked: habit forming products

Values At Play

The New Games Book

Homo Ludens

The Grasshopper

The Art of Computer Game Design

Video Game Storytelling

A Theory of fun

Masters of Doom

Blood Sweat and Pixels

<https://www.youtube.com/watch?v=CBjr4S24074>

14) Other Resources

[GameUI](#) (reference website)