Math for Game Devs

A math course by Freya Holmér for students at FutureGames
shared publicly with permission 💖

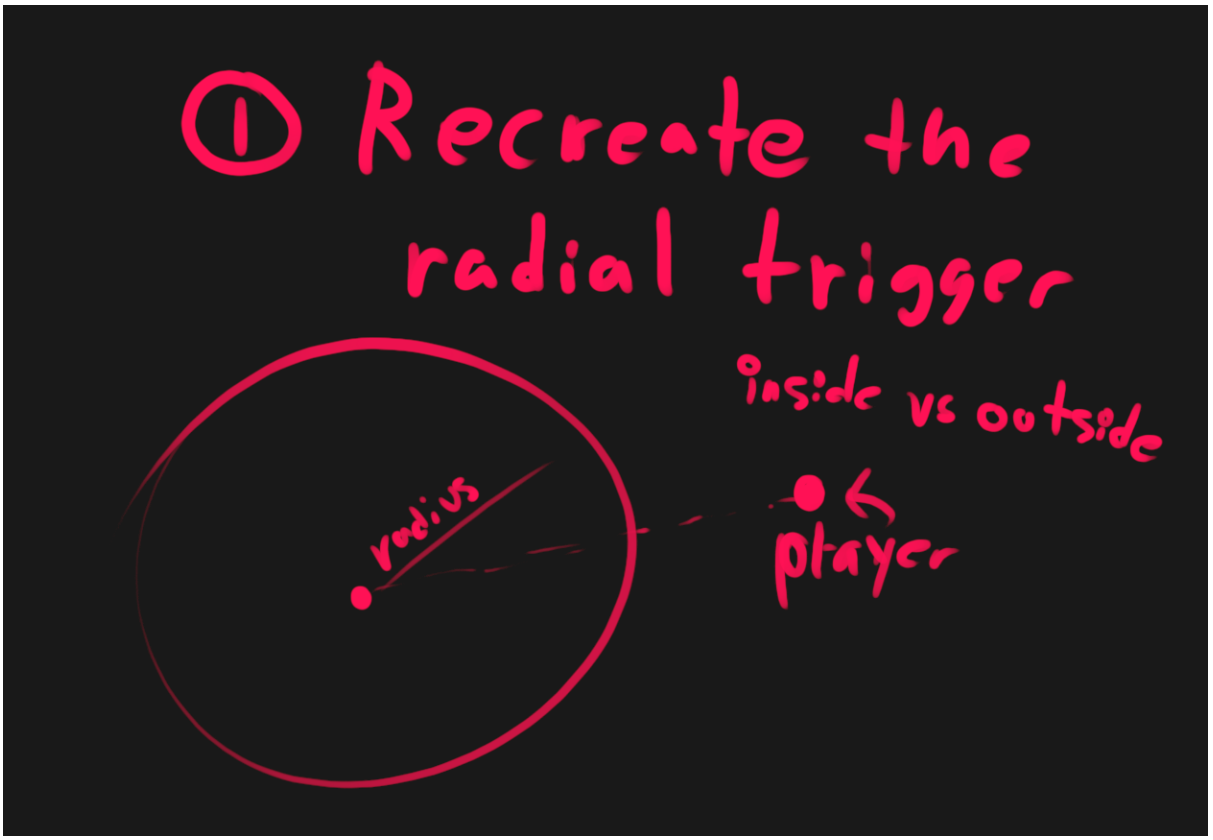
Lectures

1 • Scalars, Vectors & the Dot Product • Video
2 • Spaces, Matrices & the Cross Product • Video
3 • Trigonometry • Video
4 • Interpolation, Rotation & Point Velocity • Video

Lecture 1 • Video

Scalars, Vectors & the Dot Product

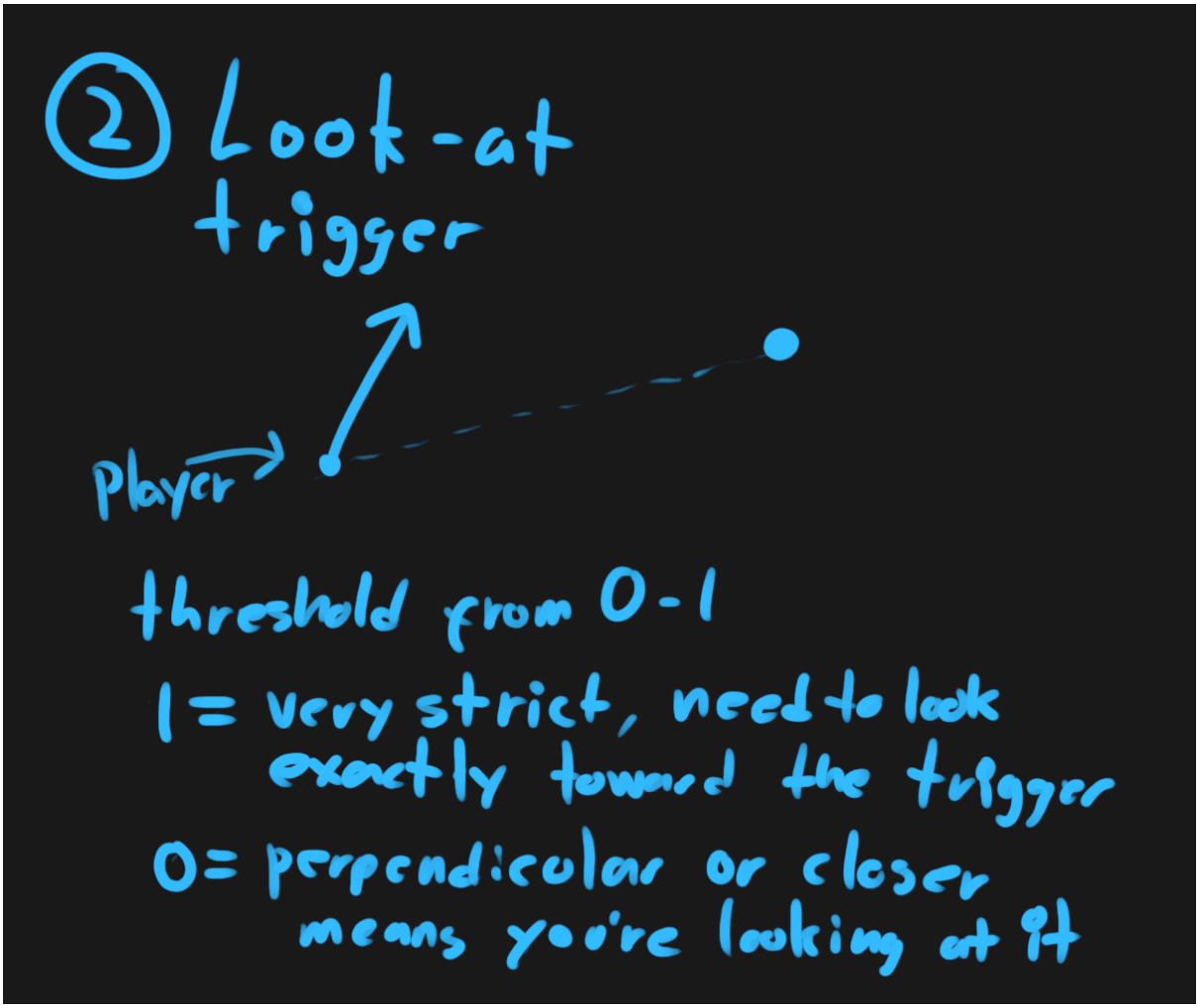
Assignment 1



Recreate the radial trigger we made on stream

- Try to not look back at the stream, see if you can remember the concepts
- Use OnDrawGizmos to draw a circle representing the radius
- Detect if an object's location is inside or outside the radial trigger
- Draw the gizmo green if inside, red if outside
- The trigger should be able to be placed at any location (ie, not just at 0,0)

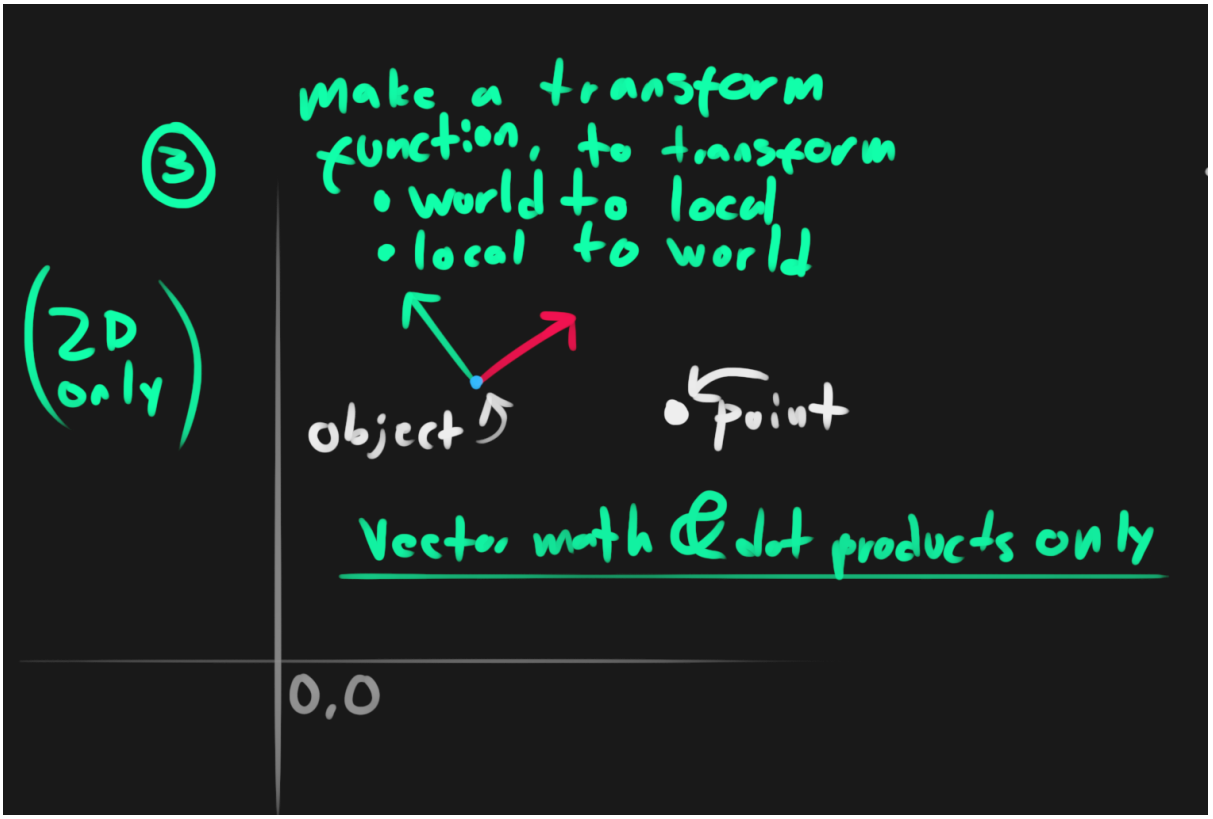
Assignment 2



Make a look-at-trigger

- Detect whether or not an object is looking at the trigger
- Have a threshold range from 0 to 1 where:
    - 1 = you have to look exactly toward the trigger to activate it
    - 0 = you have to look perpendicular or closer to activate it (facing away counts as outside)
- Draw a line gizmo green if looking closely enough, red if looking too away from it

Assignment 3



Make a function that can transform:

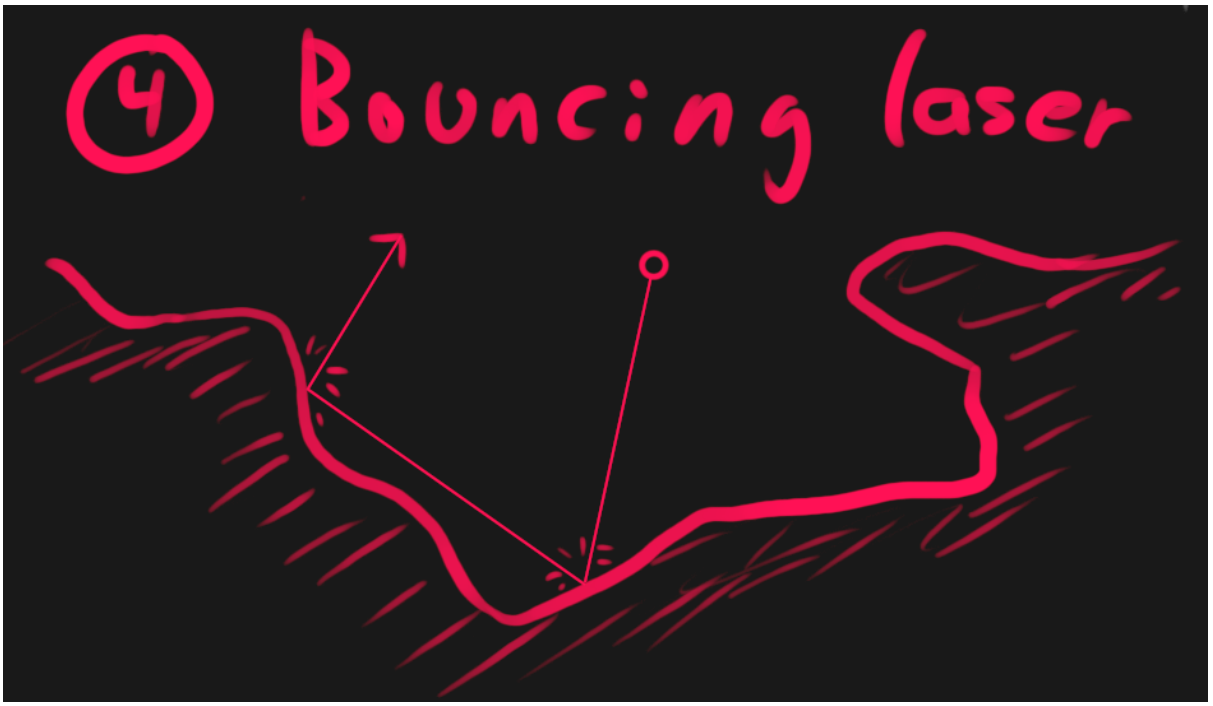
A) a world space point to a local space point
B) a local space point to a world space point
...taking rotation of the object into account

- 2D only
- You can ignore scale if you want
- You are not allowed to use:
  - transform.TransformPoint
  - transform.TransformVector
  - transform.TransformDirection
  - Quaternions
- You are allowed to use
  - transform.right
  - transform.up
  - transform.forward
- You need to do this manually, using the dot product and vector math!

Lecture 2 • Video

Spaces, Matrices & the Cross Product

Assignment 4

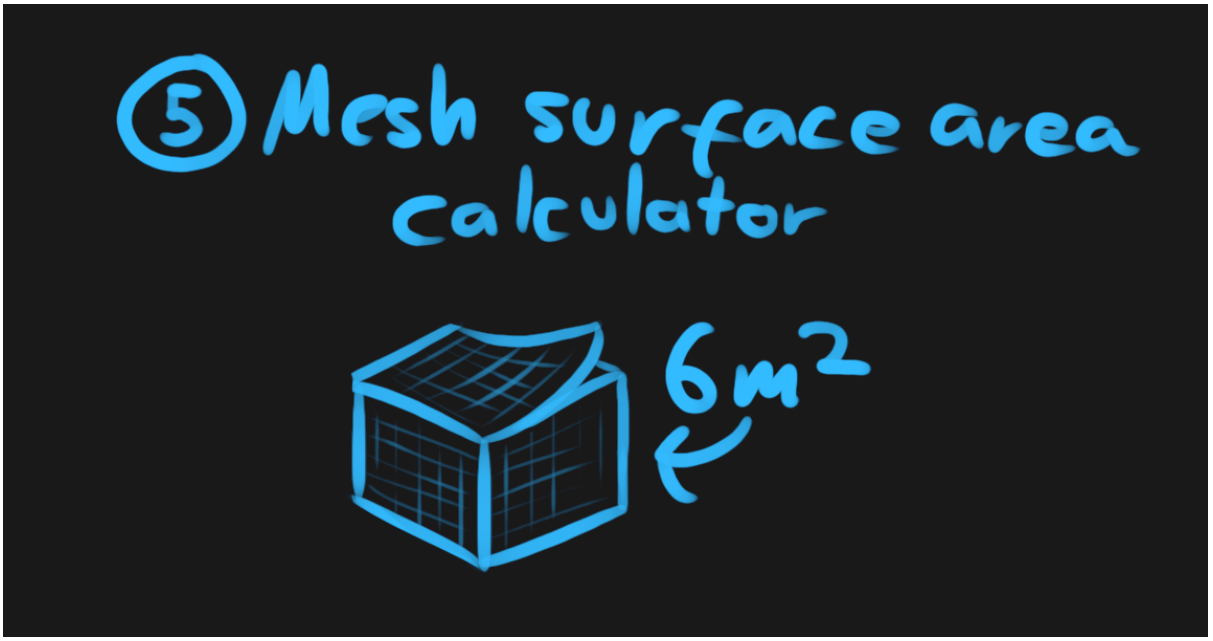


Draw a bouncing laser

- Set up a scene with some simple objects that a laser can bounce on
- Include spheres! they're fun~
- Use Phyics.Raycast and draw lines in OnDrawGizmos
- You are allowed to use `raycastHit.point` & `raycastHit.normal`
- Have an int property for max bounce count
- You don't have to animate the laser moving through the world
- Use your own vector reflection math! The goal of this is to try and figure out the vector math yourself, for calculating the outgoing direction based on an incoming vector and the surface. You can assume a mirror-like bounce
- Try to do this without looking things up - you know all the tools you need to calculate this already!
- In other words - using math libraries (other than dot/cross product) or `Vector3.Reflect` is cheating~

Bonus optional fun thing: make an object move along the bounced path at a fixed speed when you press play

Assignment 5

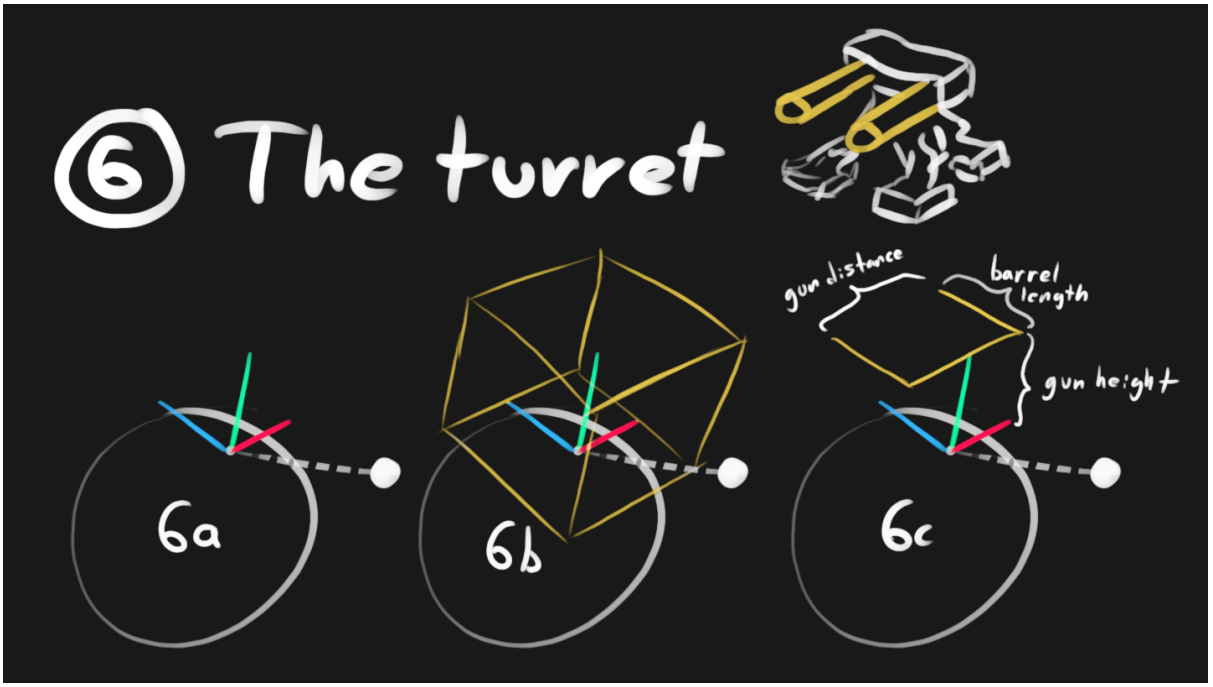


Write a function that can calculate the surface area of meshes

- Make a "public Mesh mesh;" field in a component so you can easily assign any mesh
- Check out Unity's mesh documentation to see how the data you need is stored
- To clarify - this is the total surface area of all surfaces in the mesh, not a bounding box or anything like that~
- The Unity cube for instance, has 6 faces that are all 1x1 meters, so total surface area is 6m²
- If you got this right, then you should get these values using Unity's primitive meshes
    - 1m² (quad)
    - 6m² (cube)
    - 100 m² (plane)
    - 3.105… m² (sphere)
    - 6.250… m² (capsule)
    - 7.802… m² (cylinder)

Bonus optional fun thing: same thing but calculate the volume 👀

Assignment 6



6a) Recreate the turret-placement alignment code we made in this lecture - calculate a proper orientation for a turret that is to be placed in front of the player, through a simple raycast

- Use the normal of the surface as the up vector of the "turret"
- Use the direction the player is looking to calculate what direction the turret should look
- Draw the basis vectors of the final turret placement
- You are allowed to use `raycastHit.point` & `raycastHit.normal`

6b) Create a matrix and draw a box using local space coordinates

- Construct a Matrix4x4 using these basis vectors and the position
- Unity's Matrix4x4 documentation will help a lot here!
- Use this Matrix4x4 to draw the wireframe of a cube using these local space coordinates, which will act as a stand-in for the bounding box of the invisible turret!

```
Vector3 corners = new Vector3[]{
      // bottom 4 positions:
      new Vector3( 1, 0, 1 ),
      new Vector3( -1, 0, 1 ),
      new Vector3( -1, 0, -1 ),
      new Vector3( 1, 0, -1 ),
      // top 4 positions:
      new Vector3( 1, 2, 1 ),
      new Vector3( -1, 2, 1 ),
      new Vector3( -1, 2, -1 ),
```

```
        new Vector3( 1, 2, -1 )
};
```

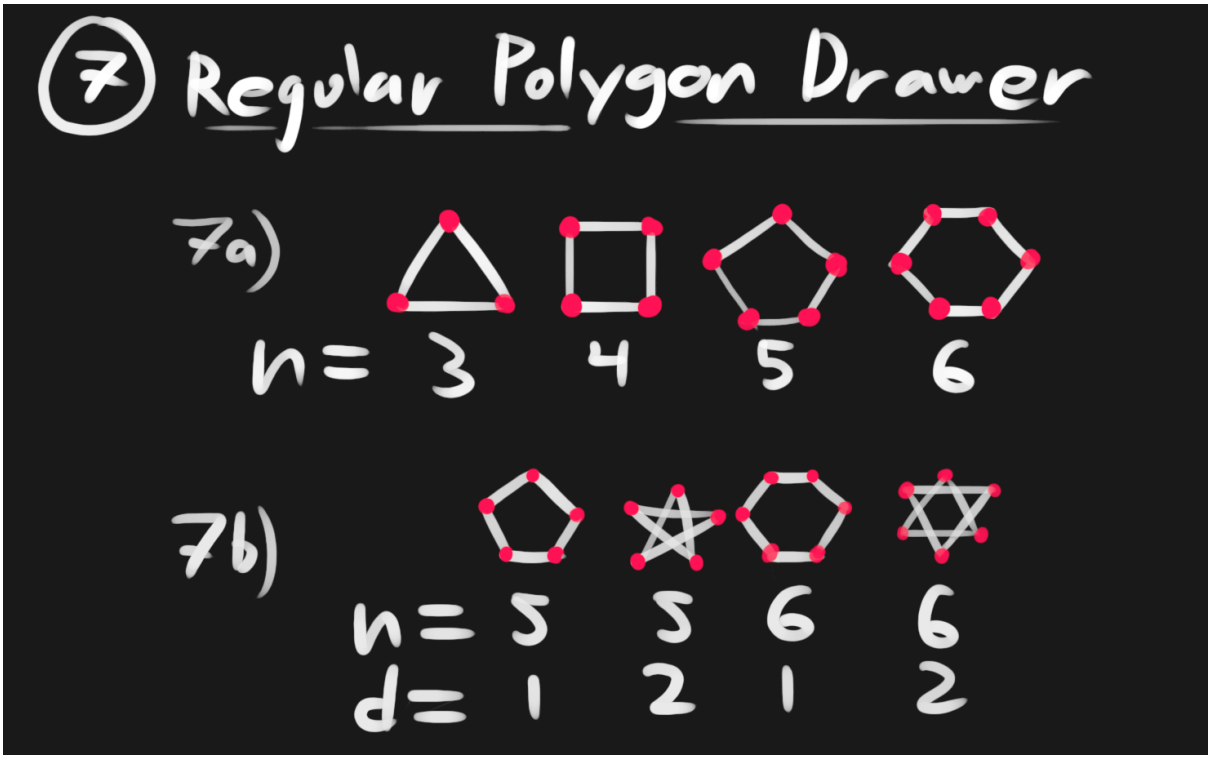
6c) Draw lines representing the "guns" of the turret!

- The turret has two guns
- Expose tweakable parameters for:
    - Gun height (distance from ground to barrels)
    - Distance between the guns
    - Gun barrel length
- Some useful default values are height = 1.3, separation = 0.3, length = 0.8
- Draw the gun barrels!

Lecture 3 • Video

Trigonometry

Assignment 7



7a)

- Make a regular polygon drawer
- Regular polygons are polygons like:
    - n = 3 (equilateral triangle)
    - n = 4 (square)

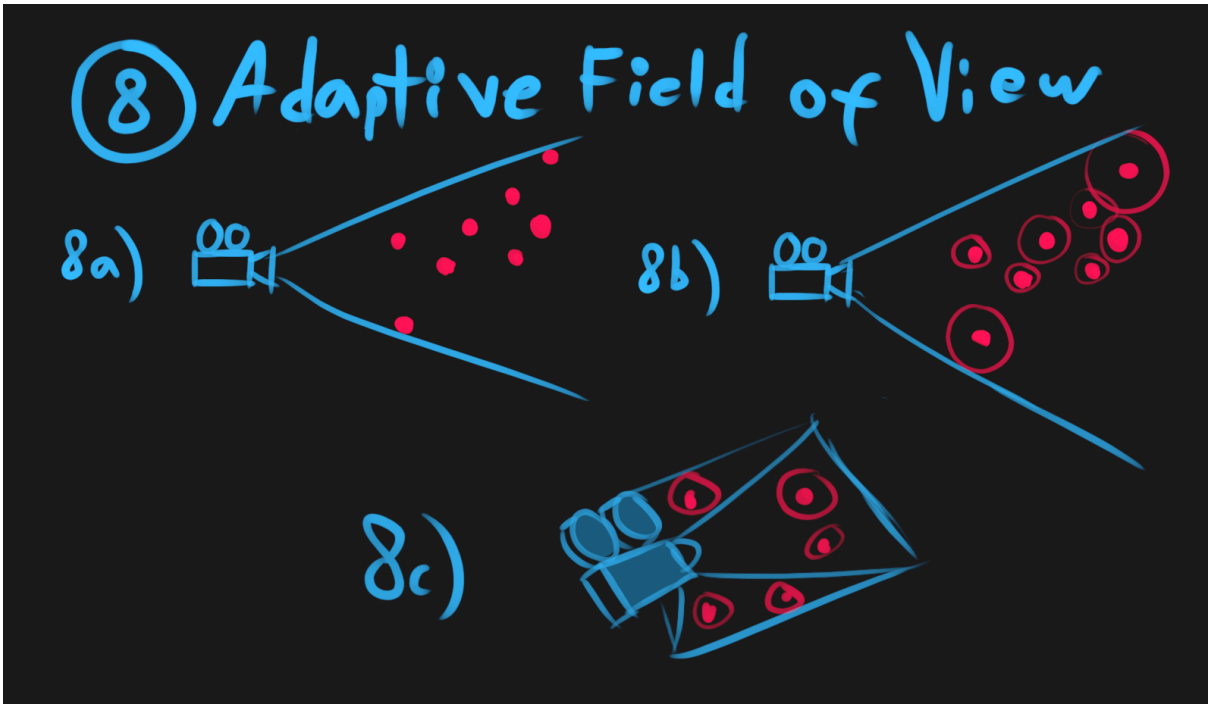
    - n = 5 (pentagon)
    - n = 6 (hexagon)
- You should be able to set a value **n** that specifies the number of sides. Any values should be valid, as long as n > 2

7b)

- Add a density value **d** that allows you to “skip” vertices when connecting them with lines, allowing you to draw not only pentagons, but pentagrams, and not only hexagons, but hexagrams
- When **d** = 2, it means lines connect two points away, instead of one point away, etc

(optional advanced thing) also calculate the area

Assignment 8



8a)

- Extend the field of view script we made on stream!
- Make a camera that will auto-adapt its (vertical) field of view, in order for a *set* of points to all be visible, not just a single point

8b)

- Give each point their own radius, and visualize this
- The field of view should adapt so that all radii of all points fit within the field of view

8c)

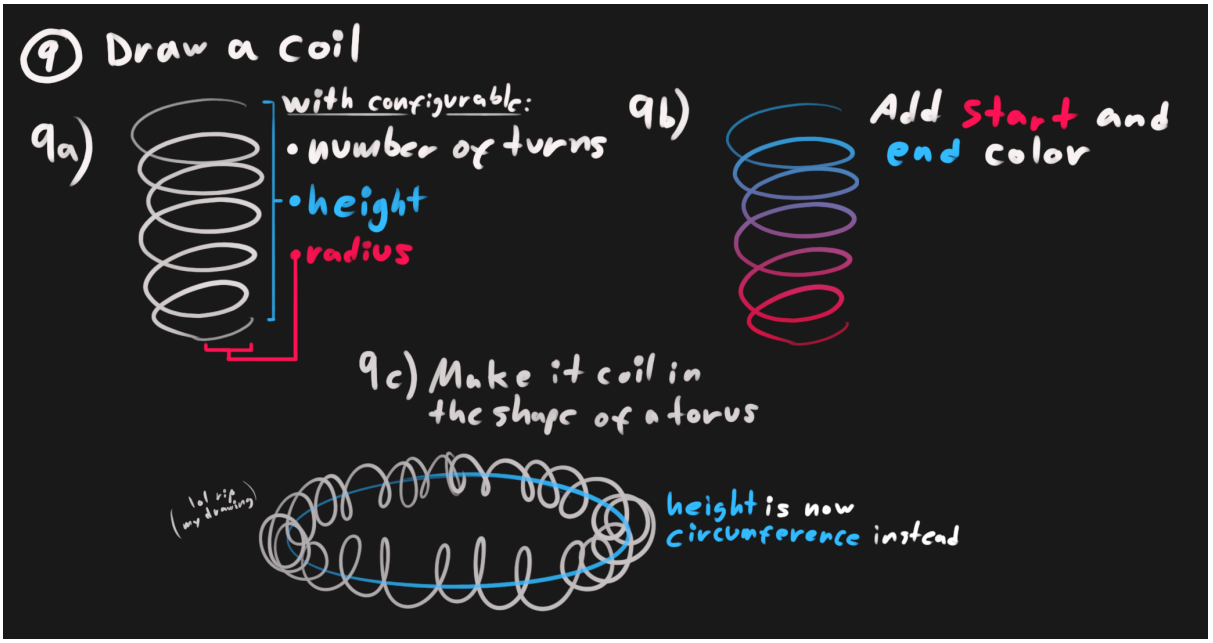
- Make this work in 3D!
- The camera should be able to have any rotation and position
- You can ignore horizontal field of view, and only focus on vertical

(optional advanced thing) Take horizontal field of view into account to make sure no points are ever outside either horizontally or vertically

Lecture 4 • Video

Interpolation, Rotation & Point Velocity

Assignment 9



9a)

- Draw a coil / spring shape with Gizmos.DrawLine!
- You should be able to set:
    - Number of turns / full revolutions around the center axis
    - Height
    - Radius

9b)

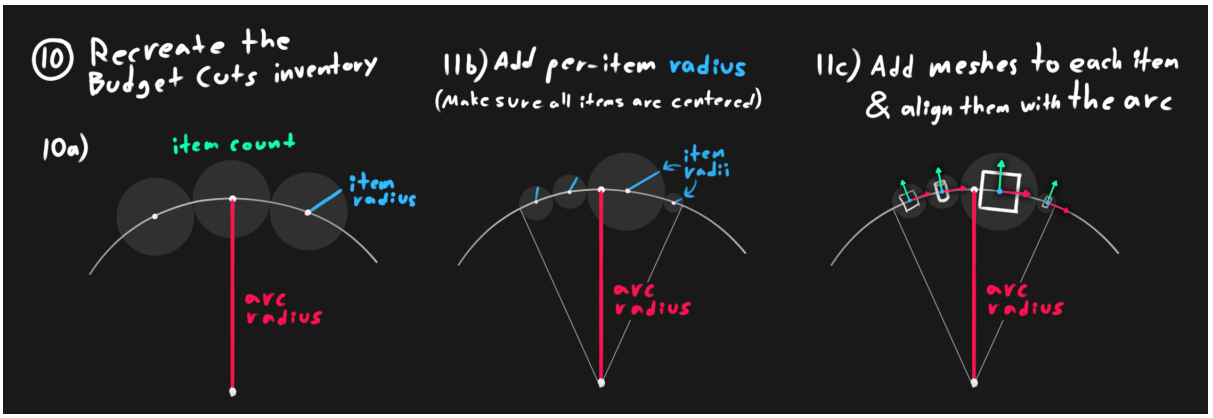
- Add a start color and an end color
- Colors between should be a linear blend between start and end

9c)

- Make it coil in the shape of a torus
- Height is now Circumference of the torus instead

(optional advanced thing) Make it coil around a 3D bezier curve instead of a torus

Assignment 10



10a)

- Recreate the inventory from Budget Cuts!
- In Budget Cuts, items are placed in an arc in front of your VR controller
- You should be able to configure:
    - The number of items / item count
    - The radius of the arc that items are placed along (arc radius)
    - The item radius
- The item positions should then be placed so that each item lies in this arc, as well as having their circles/radii touching each other with no overlap
- Items are centered so that they are grouped in front of the player
- You can draw these as circles and arcs using the Gizmo and handles drawing functions, it does not have to be an actual functional inventory
- Draw all of these in the local space of a game object, which will act as a stand-in for a VR controller transform

10b)

- Make each item have a configurable radius
- They should all still be centered geometrically
    - in other words, centering on the middle item is generally not actually centered when they all have different radii

10c)

- Add meshes to each item
    - For simplicity, you can child objects of the inventory transform be considered an item
- These meshes should be positioned on the arc, and have a rotation that aligns them to the arc

(optional advanced thing) Make it so that adding/removing items while the game is running animates them smoothly into position