Introduction to Game Engine Programming

By Connor Walsh

Goal

The goal of this workshop is for you to learn about the programming behind game engines.

I feel the best way to do this is to explain all of the aspects rather than just code something with little to no explanation.

What is a game engine?

- Simply, a set of tools to create a video game.
- Most people know this, so today we're going to take a deeper look at what goes on under the hood
- A lot of people tend to think that a game engine is a game engine editor like what they are used to seeing with Unity or Unreal Engines
    - However, the editor is just a piece of those engines that allows for even easier usage

Before we start...

Creating an entire game engine is a MASSIVE undertaking that mostly no one will ever fully do on their own. Typically, this would require a LARGE team and a long time.

This workshop is titled, “Introduction to Game Engine Programming”, not “Build your own Game Engine” for a reason.

That reason is, while you won’t be building an engine from scratch, I do think having an understanding of game engine fundamentals is a great way to learn more about programming.

The four main aspects of a Game Engine

1. Game logic
2. Graphics
3. Audio
4. Physics

Game logic

Since it seems most of our UAlbany IEEE members are computer science majors, this should be a fairly straightforward place to begin.

The game logic portion of an engine behaves just like any standard program you've written behaves.

Depending on what language being used (Most use object oriented languages like C++/Java) there would be multiple objects created to represent in game objects.

Graphics

Modern game engines use very low level APIs for rendering graphics. These APIs are capable of rendering both 2D and 3D graphics.

The most popular of these APIs are:

- OpenGL (Used to make Minecraft)
- DirectX (Made by Microsoft, Used by nearly every AAA developer)
- Vulkan. (Relatively new, some AAA developers have been using it)

Let's briefly touch on how OpenGL works.

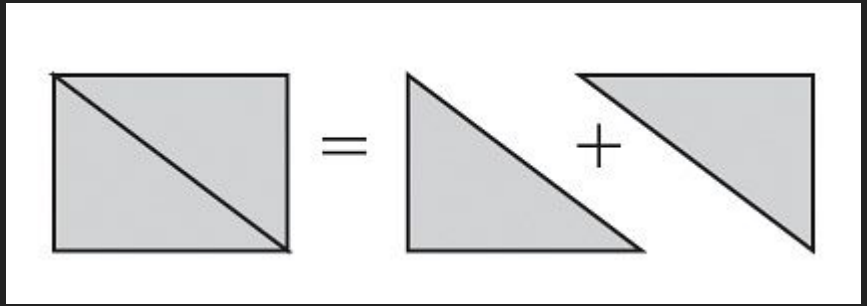
Lots and lots of triangles...

OpenGL only knows one shape, and that is a triangle.

If you want to make a rectangle, you need to use two triangles. (Incredible, I know)

Having to manage all of these triangles and their positions can be quite annoying, luckily OpenGL has a way to manage this.

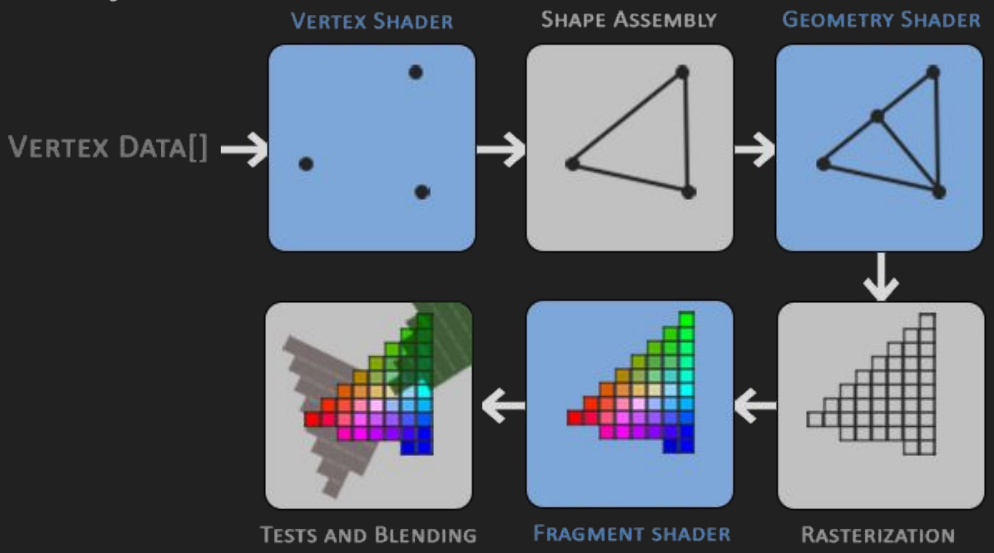
It's called a vertex buffer object.



Vertex Buffer Objects

The simplest way I can describe this is OpenGL will keep one copy of the shape (i.e. a cube) inside the GPUs memory, and then whenever you want to draw this exact shape, instead of having to use the exact vertex positions again, you can just ask OpenGL for the object from memory.

After this stage, it is simply shading and determining what the final color the object will be displayed as.



Not that bad huh?

I hope I’ve bored you sufficiently on the topic of 3D graphics, so let's move on.

Audio

An audio manager is essential because in a game, there will be multiple objects emitting noise at varied distances from the player, and also music.

It is important to handle this in such a way that audio is mixed properly and does not destroy the end user’s eardrums. After all, bad audio is the fastest way to ensure an Alt-F4.

A typical, simple solution for this is to have a Singleton class as a dedicated audio manager that would be able to be passed the name of certain audio tracks to then be played.

Audio Video (For later)

https://youtu.be/UvRU25T_XOg

(By Coding Tech)

Physics (The fun part)

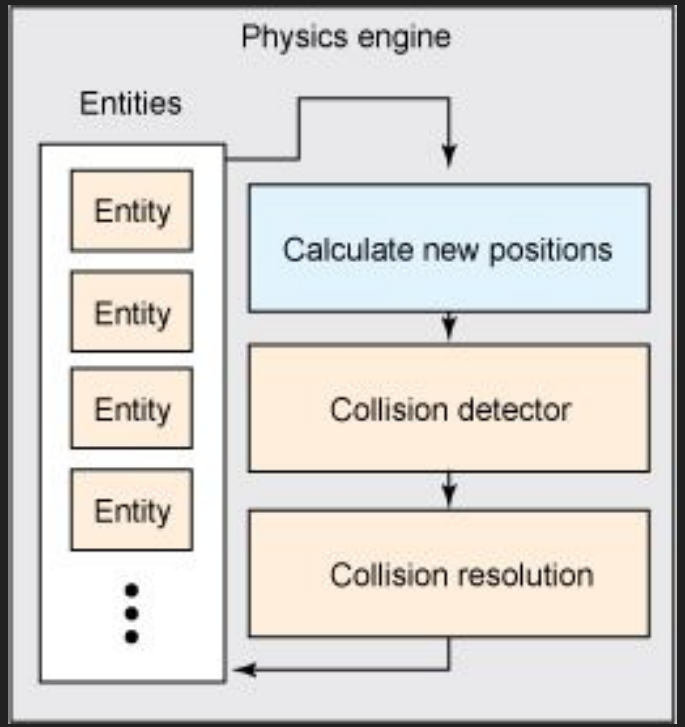
Typically, every single game object will have its own properties such as mass, collider shape, velocity, etc. This makes things a bit simpler when determining how objects should react to and with other objects.

A physics engine will typically have functions to handle the input locations and velocities and the output locations and velocities. At a super high level, it’s nothing more than calculus. (Just a ton of it)

Physics Video

https://youtu.be/-_IspRG548E

(By WinterDev)



User Input

After all, it can’t be considered a game unless there is a player controlling it…

We need code that will handle user input consistently and tell the rest of the components of the game engine what to do

But how do we accomplish this?

Game Loop

Game loops are what differentiates a game from a still image.

Video games are very resource intensive and unoptimized for our hardware.

This is because in order to be ready for user input and change on the screen, we need to CONSTANTLY loop over EVERYTHING that we are doing in the engine.

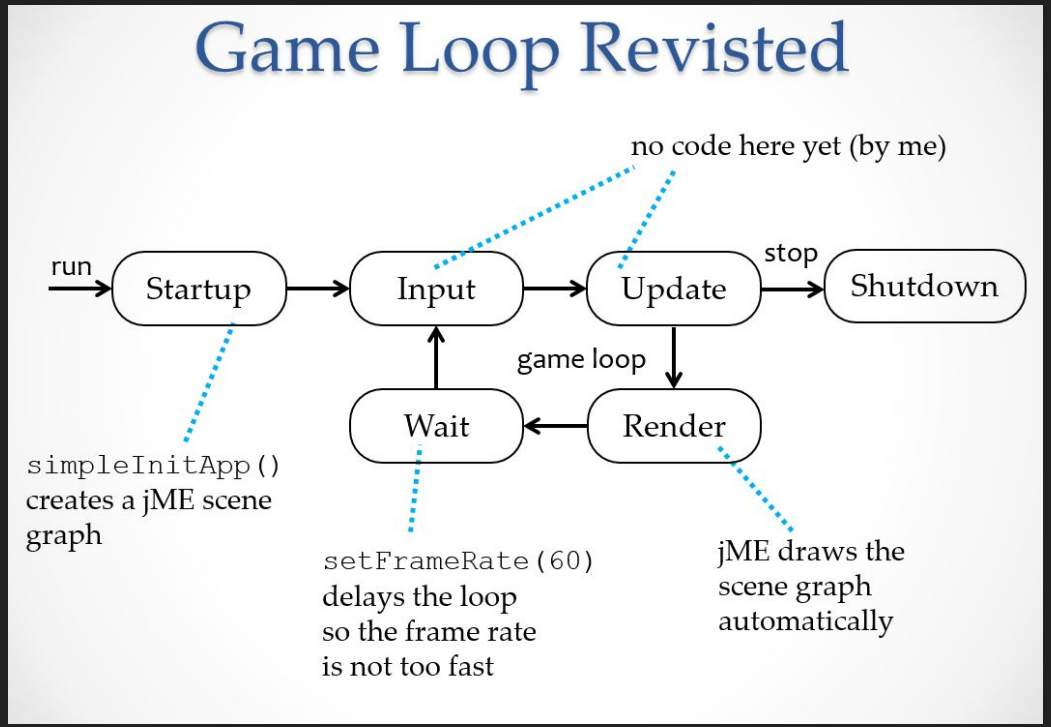
Game Loop

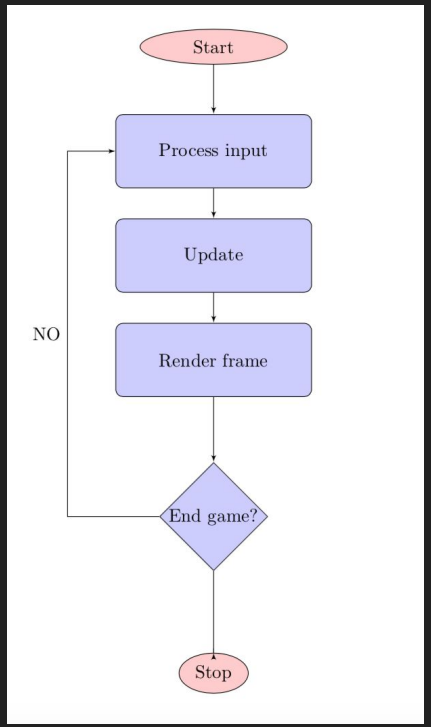
Let’s say a game runs at 60 frames per second (Pretty low for my tastes), then we would need to:

- Read user input
- Calculate physics
- Re-render the entire scene
- Determine what audio should be playing
- and Evaluate game logic

60 times every second, non-stop, until the user quits the game.

Game Loop


Game Loop Revisted
no code here yet (by me)
run
Startup
Input
Update
stop
Shutdown
game loop
Wait
Render
simpleInitApp()
creates a jME scene
graph
setFrameRate(60)
delays the loop
so the frame rate
is not too fast
jME draws the
scene graph
automatically


Start
Process input
Update
NO
Render frame
End game?
Stop

Videos!

https://youtu.be/DKrdLKetBZE - TheHappieCat

https://youtu.be/xNCeypH72AQ - Linus

https://youtu.be/vtWdgtMo1T4 - The Cherno (Not watching)

Demo Time

I’ve been working on a 3D engine this past summer using OpenGL and C++.

The code is available on my GitHub if you want to take a look

- https://github.com/connorwalsh21/CW3D

I’ve spent most of my time focusing on the graphics, so as of now, there is no audio or physics. (This stuff is quite difficult)

Next Steps!

If you enjoyed graphics, you should look into:

- OpenGL
  - OpenGL is quite difficult, but still much easier than DirectX or Vulkan
- Shaders

If you enjoyed audio, you should look into:

- Low level audio libraries for C++ (Or any language of your choice)
- Audio file design

Next Steps!

If you enjoyed physics, you should look into:

- Taking more physics classes (Seriously)
- Writing a physics simulation in an established engine such as Unity for practice

Summary

I hope you all learned more about the programming behind all the game engine “magic” and walk away with a new interest.

If you are very interested, on the next slide, I will have a list of things you should study in order to get to the next level with these technologies.

Thank you for coming!

Stay tuned next semester for a potential follow-up workshop!