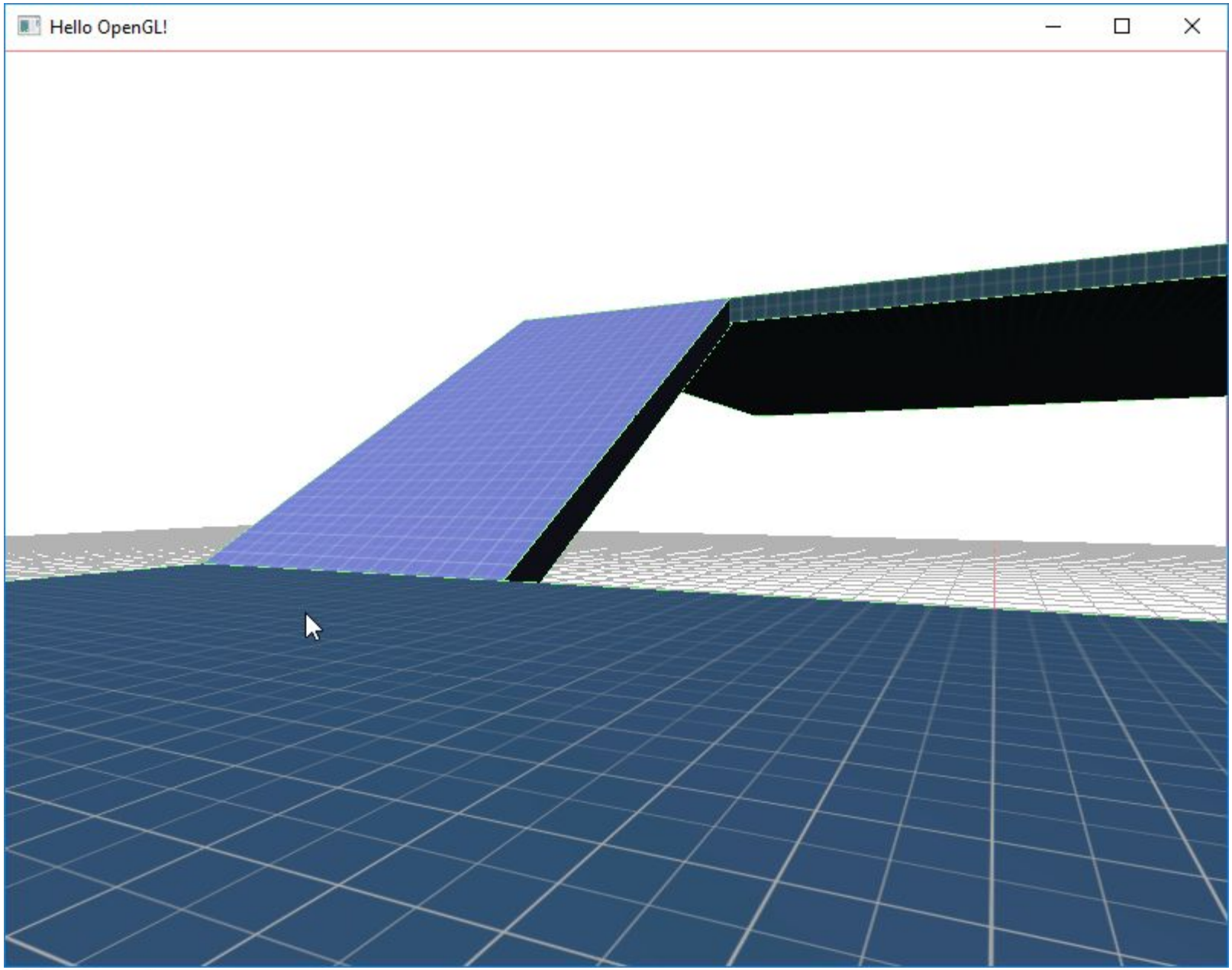
MVD: Engine Programming

06 - Levels

alunthomas.evans@salle.url.edu

laSalle ENG
Universitat Ramon Llull


Hello OpenGL!

FPS control

Rotate camera as Free movement

Detect collision on ‘down’ collider

- translate player entity (and camera) to be always ‘FPS_height’ units above collision point

‘Forward’ and ‘Strafe’ dirs are clamped to 0 in y-axis

Only move in direction if collider in that direction isn’t colliding

FPS Jump

Need to add gravity (if down collider distance is greater than FPS_Height) apply gravity

Need to create FPS_jump_force

- when player is jumping, add translate FPS_jump_force in y axis every frame
- reduce FPS_jump_force every frame until 0

Today's first task

1) Adapt ControlSystem.updateFPS to make an FPS camera with jump

Storing/reading data

Rotations

There are three ways we can store the rotation of an object:

i) Euler angles

ii) Matrices

iii) Quaternions

Euler Angles

PROS:

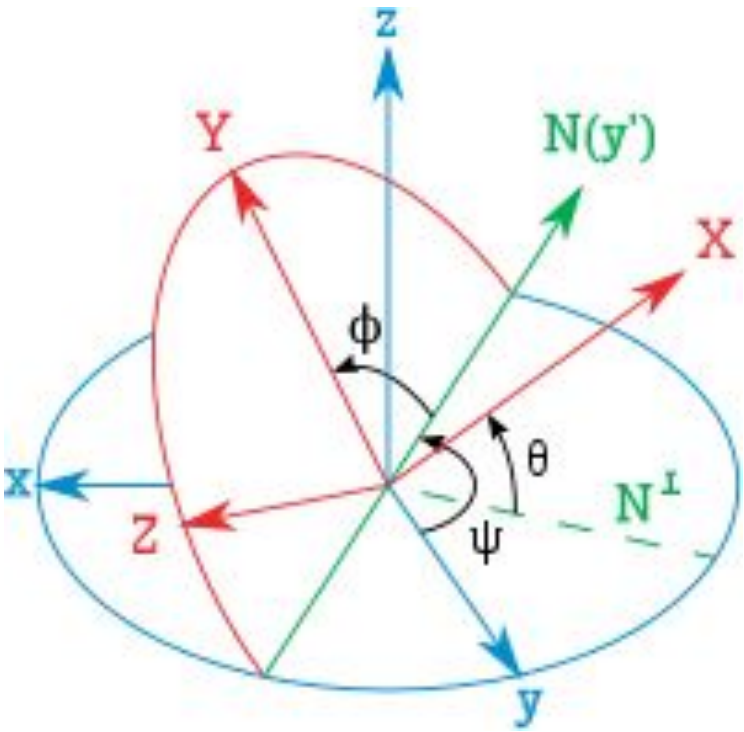
Low memory (3 floats)

Easy to understand

Simple maths to apply (sin and cos of angle)

CONS:

You have to rotate them hierarchically, one at a time = **gimbal lock!**



Matrices

PROS:

fewer operations that raw euler angles, as you can combine multiple rotations into a single matrix

CONS:

Still suffers gimbal lock

Use more memory (16 floats)

Multiple ways of interpolating between them = errors

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

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

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

Quaternions

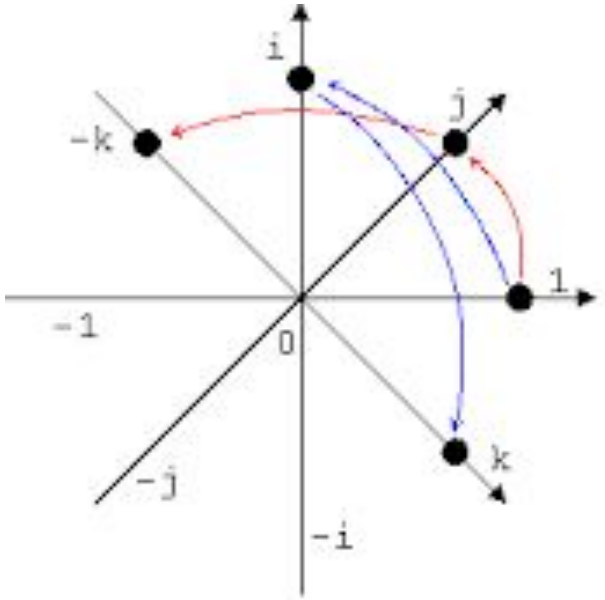
PROS:

very little memory (4 float)

no rotation hierarchy = no gimbal lock

CONS:

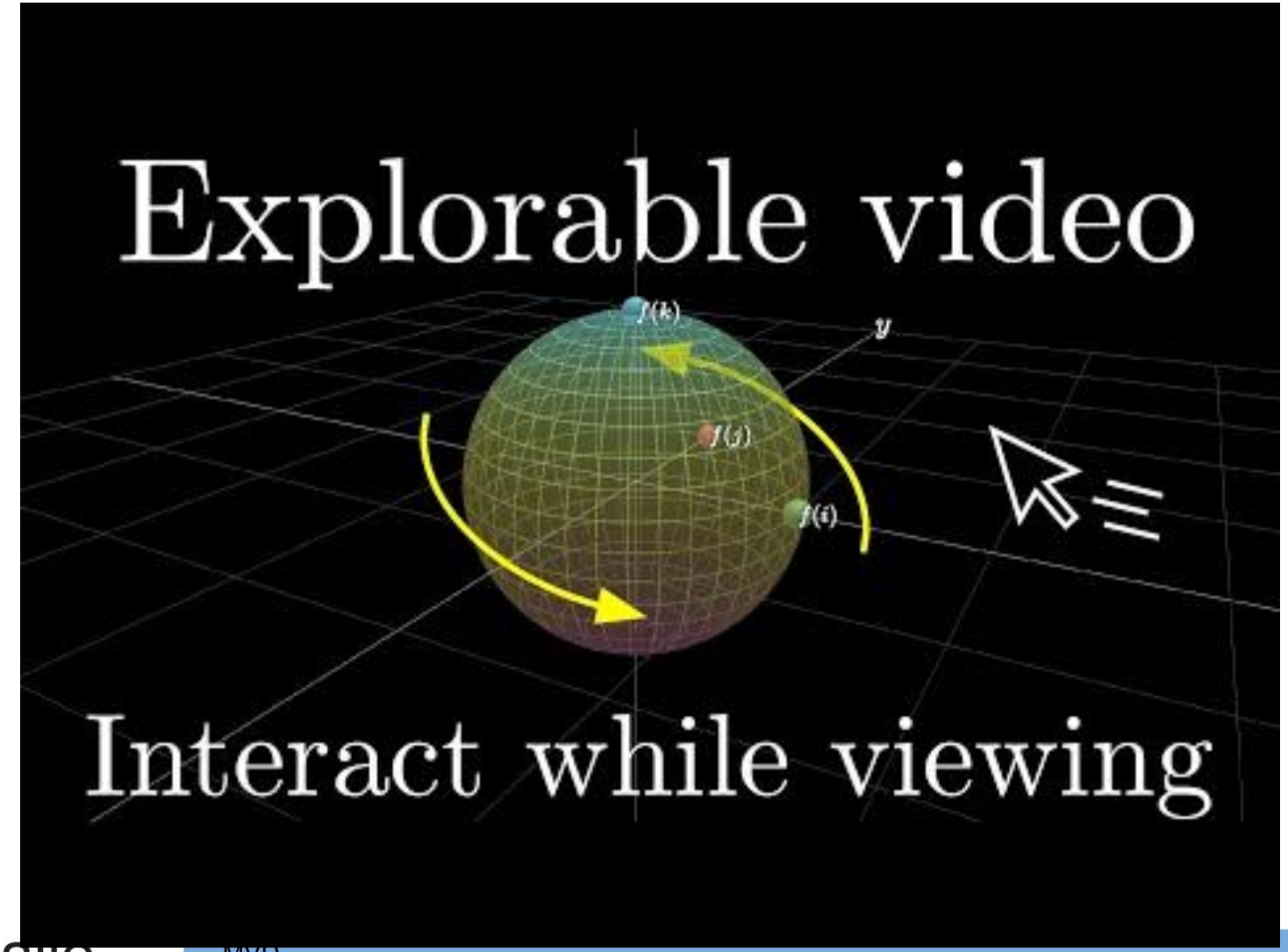
difficult to visualise how they work!



Graphical representation of quaternion units product as a 90°-rotation in 4D space

$$ij = k$$
$$ji = -k$$
$$ij = -ji$$

Quaternions explained (briefly)



Quaternion cheat sheet

Quaternion	Axis-Angle	Description
$(1,0,0,0)$	$(\text{undefined},0)$	Identity rotation
$(0,1,0,0)$	$((1,0,0),\pi)$	Pitch by π
$(0,0,1,0)$	$((0,1,0),\pi)$	Yaw by π
$(0,0,0,1)$	$((0,0,1),\pi)$	Roll by π
$(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0)$	$((1,0,0),\pi/2)$	Pitch by $\pi/2$
$(\frac{1}{\sqrt{2}},0,\frac{1}{\sqrt{2}},0)$	$((0,1,0),\pi/2)$	Yaw by $\pi/2$
$(\frac{1}{\sqrt{2}},0,0,\frac{1}{\sqrt{2}})$	$((0,0,1),\pi/2)$	Roll by $\pi/2$

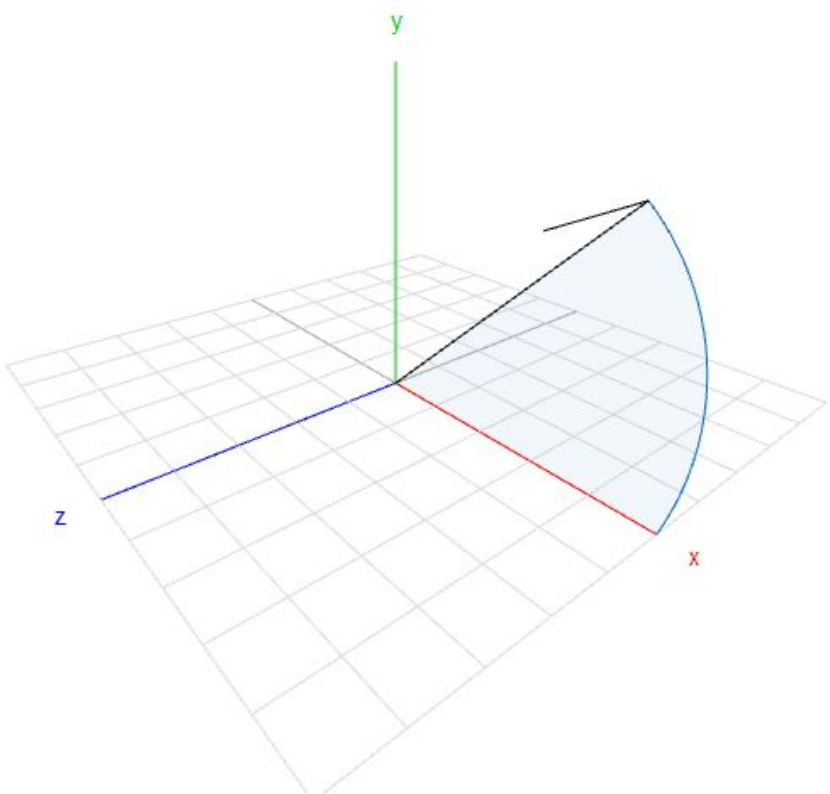
Euler to Quaternion conversion

Where alpha, beta, and gamma are euler angles in x, y, and z

$$\begin{matrix} W \\ X \\ Y \\ Z \end{matrix} = \begin{bmatrix} \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \end{bmatrix}$$

When storing rotations in a readable format, it is usually convenient to do so using euler angles. However, we convert them to quaternions before actually doing rotations

https://quaternions.online/



Quaternion

w:	0.757
x:	0.503
y:	0.348
z:	0.231

Apply Rotation

Euler Angles

XYZ - Order ▾ Degrees ▾

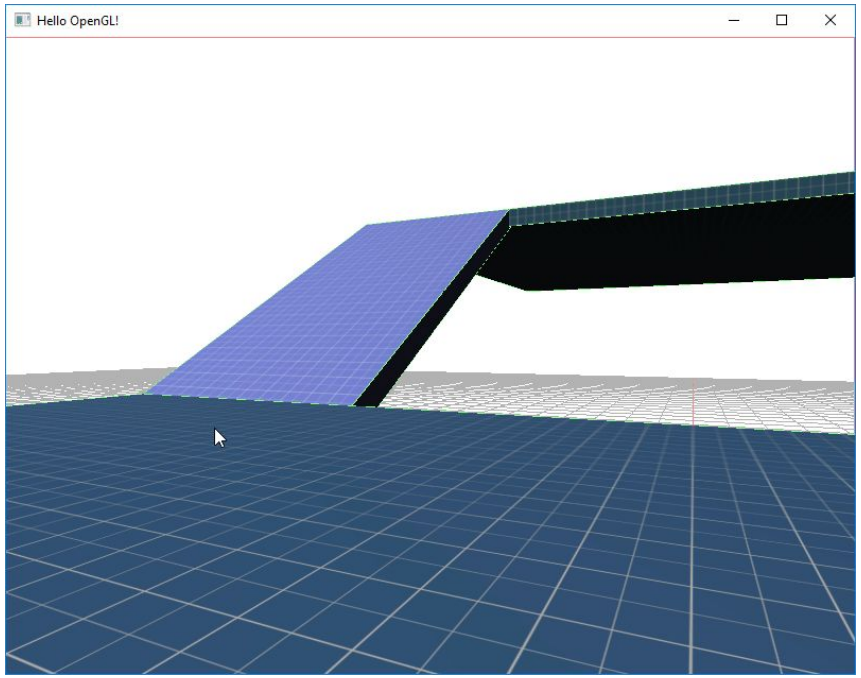
x:	67.187
y:	49.340
z:	0.000

Apply Rotation

☐ Show axis of rotation ☐ Animation

Storing scene information

What are the things we need to store?



Storing scene information

What are the things we need to store?

- Resources
  - Shaders
  - Meshes
  - Materials
  - Textures
- Entities
  - Components

Storing scene information

What are the different ways we can store information? In which format?

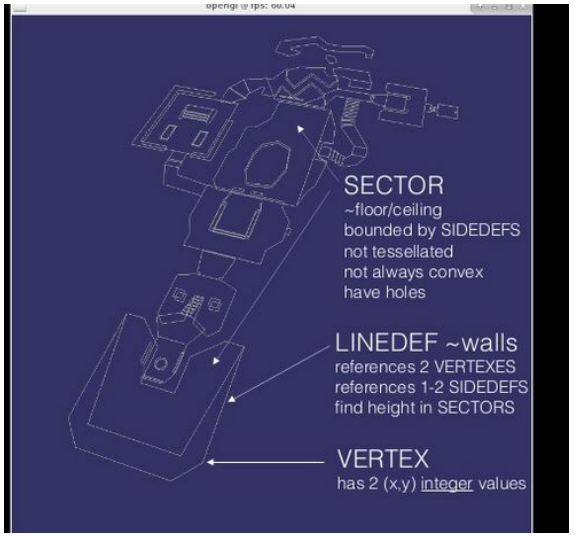
Storing information

What are the different ways we can store information? In which format?

- Binary
  - header with information regarding size (in bytes) of different elements
  - ‘blob’ with data
- Text
  - Plain text
  - XML
  - JSON

A very early game file format

e.g. Doom WAD (*"Where's all the data?"*) structure - everything in a a single binary file



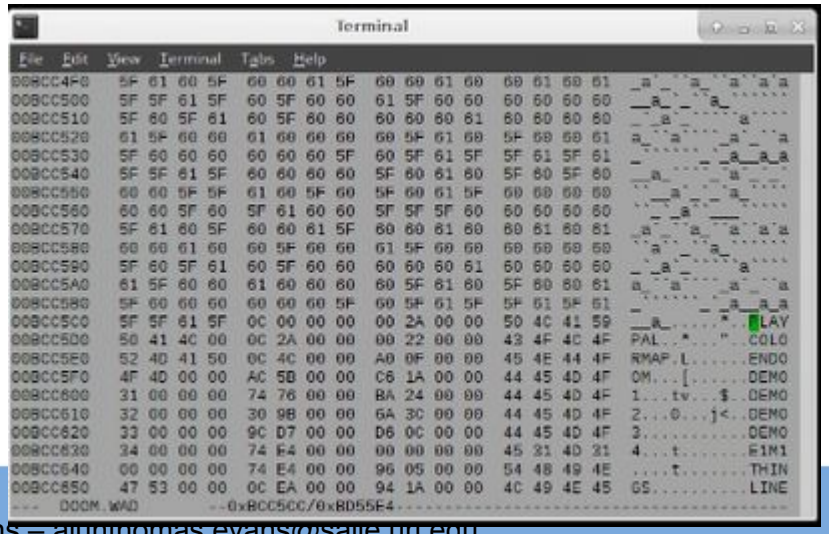
It was reverse engineered (and then, later, officially 'released')

http://www.gamers.org/dhs/helpdocs/dmsp1666.html

Binary file reading

Binary files can be of variable sizes, and you can't read them line by line.

How do you read a binary file of varying size?



Binary file reading

Binary files can be of variable sizes, and you can't read them line by line.

1) Read header (know size from data format)
2) use size information in header to read required bytes, in correct order, allocating heap memory to store it

Text file reading

Just like .OBJ files

read line by line

Binary or Text, or both

Binary is good for storing ‘big’ data like meshes and textures.

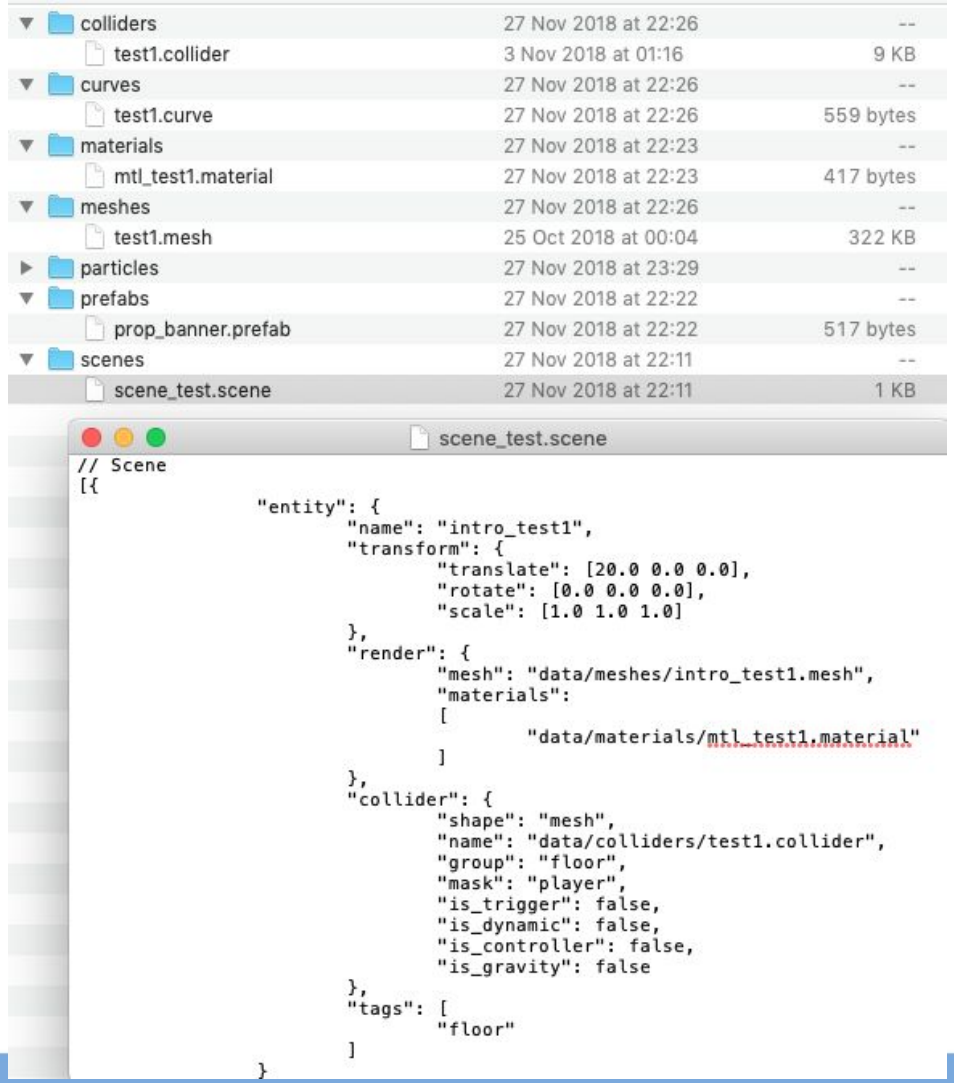
Text is better for storing scene structure, material information etc.

Why? Because you can read it and edit it with a text editor. Plus, it’s small, so quick to load anyway

Directory structure for data storage

This is what you will be developing with Alberto over the next few weeks.

Exporting from Max, importing into C++ engine



Creating a (temporary) json structure

Let's create a single JSON scheme to store our resources and define a level

RapidJSON

We *could* write our own JSON parser.

But in this case we're going to cheat and use a library, called RapidJSON

File reading example

http://rapidjson.org/md_doc_stream.html

Parser class

General approach:

1. Read all resources and store on GPU
2. Create a dictionary (std:unordered_map) for each resource, to associate it with a string id

3. Read all entities and components
4. Add to ECS using the dictionaries created in step 2

Today's task

1) Adapt ControlSystem.updateFPS to make an FPS camera with jump

2) Create a different level with assets already in directory

3) Download external assets and load them from the JSON

   a) (Test obj/json/tga loaders to find edge-cases and bugs!)

This week's homework

Preparation for christmas project

https://www.gamedev.net/articles/programming/artificial-intelligence/the-total-beginners-guide-to-game-ai-r4942/