MVD: Engine Programming

02 - Graphics System

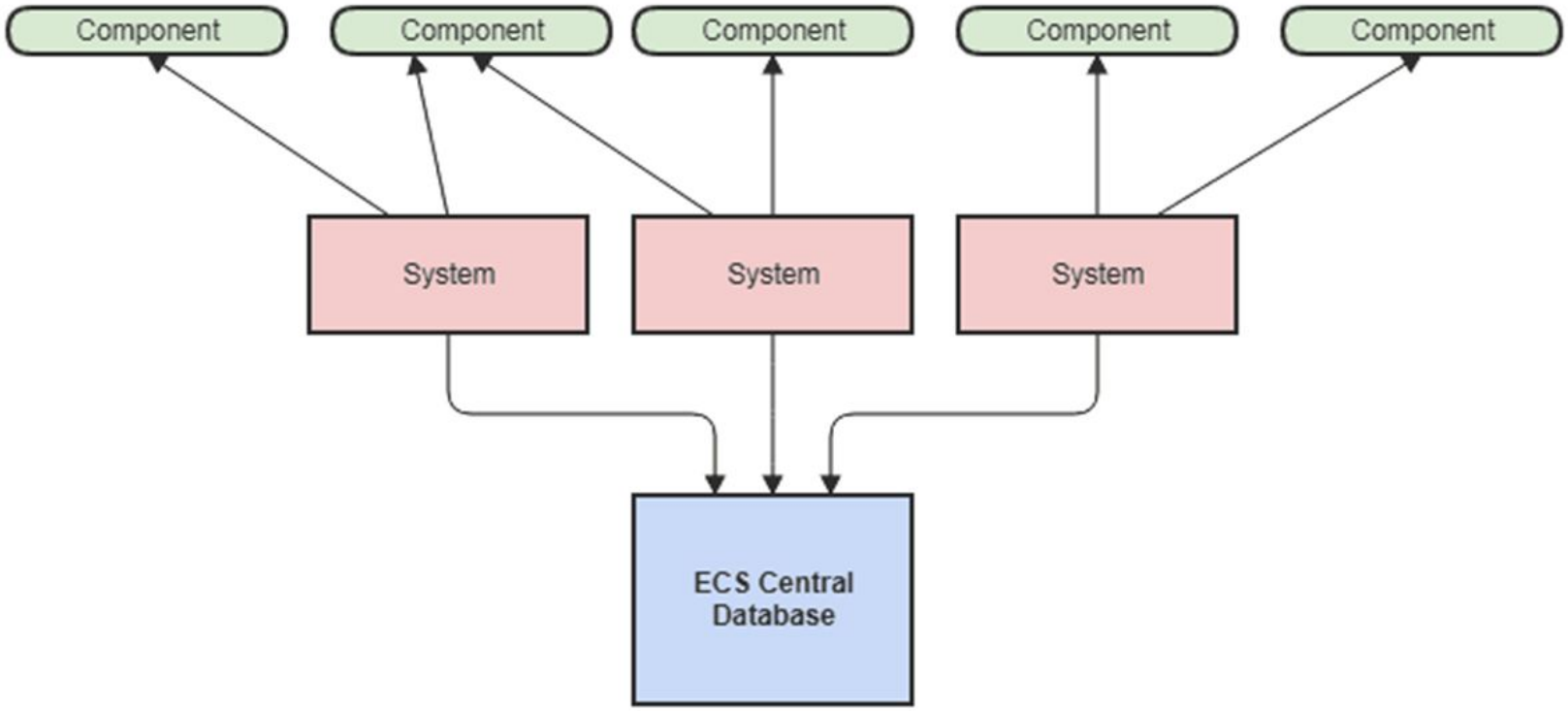
alunthomas.evans@salle.url.edu

laSalle ENG
Universitat Ramon Llull

The ‘Entity Component System’ model

Game is organised around ‘Systems’ of different types.

Each system updates all the components of a certain type *regardless of which entity they belong to.*

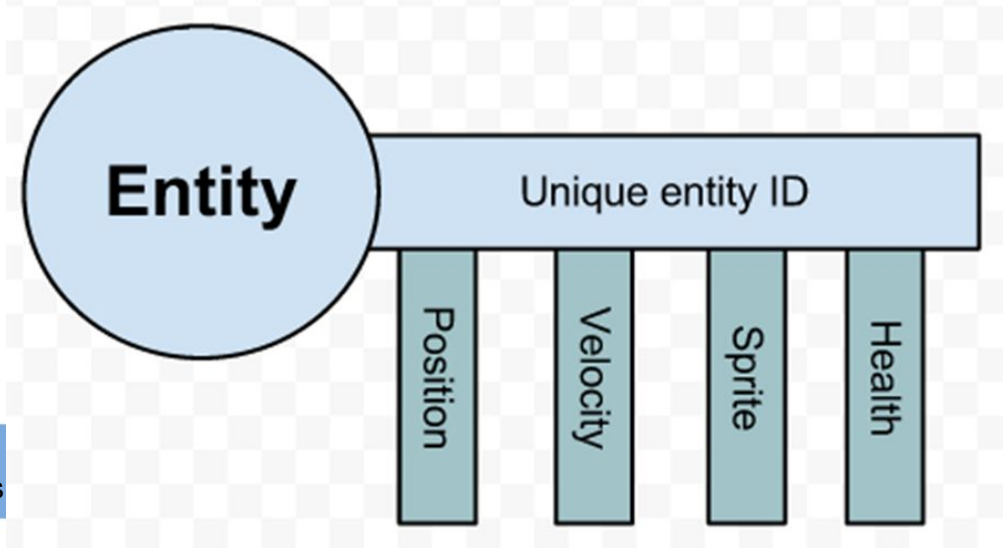


Entities are just IDs

An Entity is now essentially nothing more than a number.

Components, when they are created, are told which ID is their owner.

The Entity maintains some sort of reference to the components, but nothing else



Systems do all the work

We **never update entities**.

We create *Systems*.

Systems access and update **only the components they need to.**

Entities are just bags of components

Components are not shared (*my rule!)

The Mesh Component so far....

```
struct Mesh : public Component {
    GLuint vao;
    GLuint num_tris;
};
```

currently our component only stores information about the geometry of our mesh.

Introducing Materials

A Material is the description of the appearance of an object.

Various objects may share the same material

Should we make a a new 'Material' component?

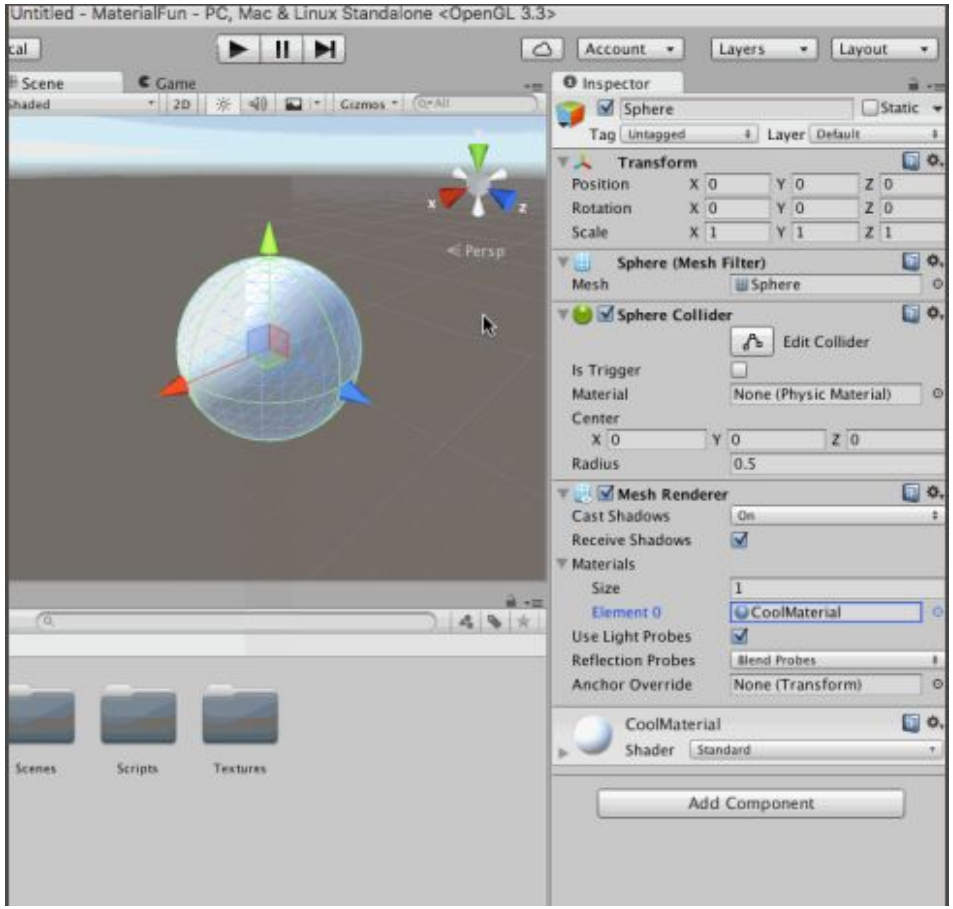
Materials don’t need to be components

Materials are not related to the rest of the game engine.

The only system that will ever access them is the graphics system

They need to be easily shareable.

So let’s NOT make them components



the new Mesh Component

Three.js has a very simple and abstract Mesh implementation:

```
Mesh( geometry : Geometry, material : Material )

geometry — (optional) an instance of Geometry or BufferGeometry. Default is a new BufferGeometry.
material — (optional) a single or an array of Material. Default is a new MeshBasicMaterial
```

we can model our MeshComponent on that

Material and Geometry structs

Geometry

GLuint vao
GLuint num_tris

Material

GLuint diffuse_map
GLuint shader_id
vec3 diffuse
vec3 specular
float shininess
...etc

Materials and Geometries can be *shared* between objects

So let's store them in the graphics system, along with the shaders

```
//resources
std::unordered_map<std::string, Shader*> shaders_;
std::vector<Geometry> geometries_;
std::vector<Material> materials_;
```

Components can reference a Material or a Geometry by its index in the storage

new Mesh Component

```
struct Mesh : public Component {
    int material_id;
    int geometry_id;
};
```

Graphics System update pseudocode

```
For all mesh components in ECS:
    get material of component
    set shader properties based on material

    get geometry of component (vao and num_tris)
    render geometry
```

C++: references (&) vs pointers (*)

Both references and pointers store the address of some data

Differences:

- a reference is like an alias:

```
int i = 5;
int &j = i; // j = 5
```

- references
  - must be assigned at initialisation (can't be null)
  - can't be reassigned
  - can't be used to delete/free memory

Returning references from getter functions

It's very useful to return references from functions that access class member variables

```
Material& getMaterial(int index) {
   return materials_.at(i);
}
```

note: std::vector accessors always returns references!!

More fun C++: array iteration and auto keyword

instead of

```
for (size_t i = 0; i < materials_.size(); i++) {
   //’materials_[i]’ to access content
}
```

we can write:

```
for (auto &mat : materials_) {
   //’mat’ to access content
}
```

.OBJ parsing

1) Create temporary vectors to store each **attribute**
   - vertices
   - normals
   - textures

2) Create dictionary (std::unordered_map) to store indices of final vertices of faces
   - e.g. “0/4/2” [0]; “1/4/2” = [1] etc.
   - use indices in dictionary to create final arrays for **attributes**

3) Send final arrays to OpenGL

Code

https://github.com/AlunAlun/MVD_02_GraphicsSystem

Today

Create Geometry and Material structs

- (see render code to find out what you need to store in each)

In graphics system, create storage arrays for Geometries and Materials.

- create functions that return **references** to then

Change mesh component to store ids of geometry and material (in graphics system storage)

Change graphics system component render to use geometry and material of component

Add a function to GraphicsSystem to parse an OBJ file and store it's geometry