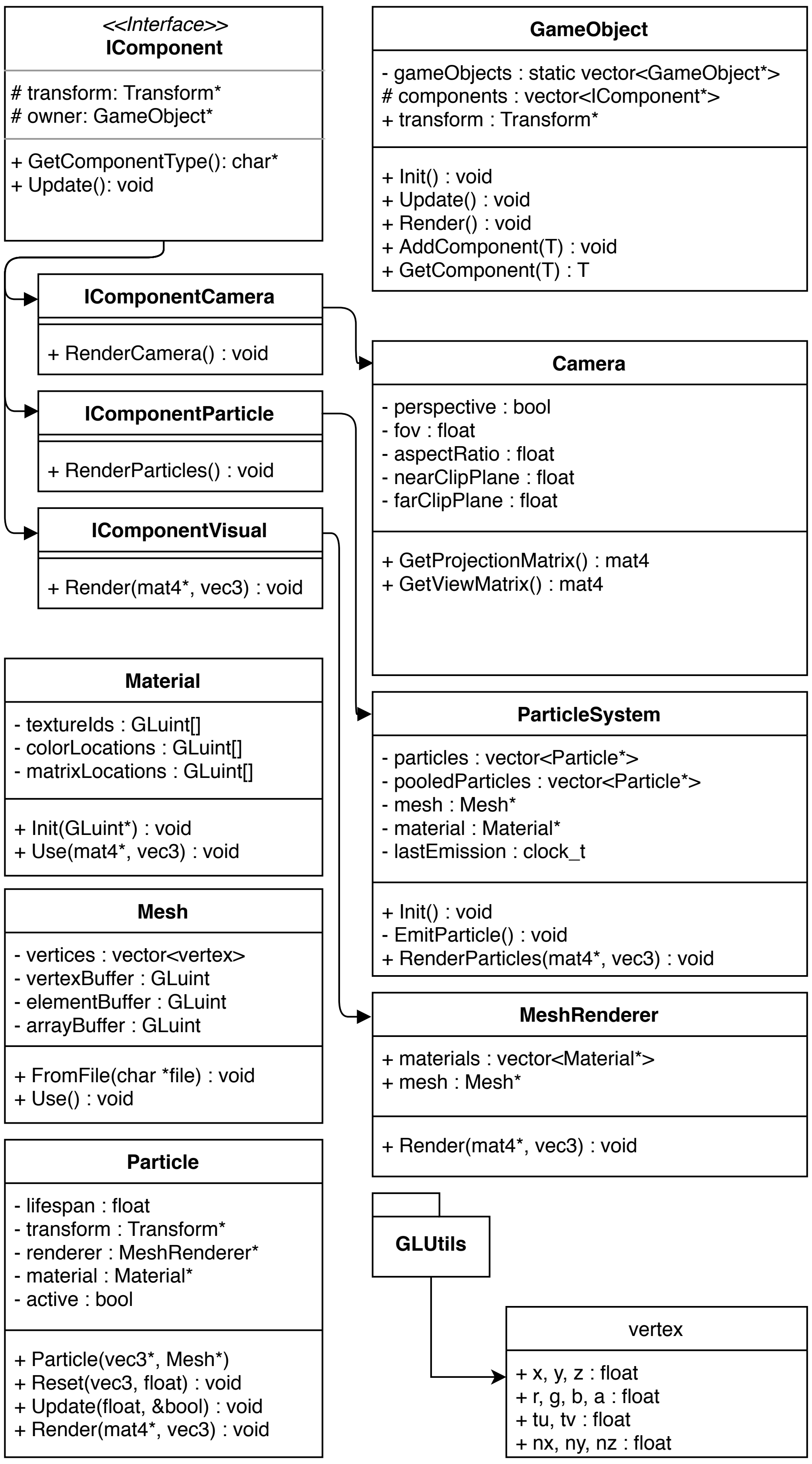
COMP220 - OpenGL Fluid simulation

Paul Rauca
BSc Computing for Games
Year 2

The purpose of this project is to create a lightweight and easy-to-use fluid physics library using using OpenGL. Given the lack of lightweight, reusable and open source implementations, a large aspect of this project will be its modularity.

Implementation of a component system



<<Interface>>
IComponent

# transform: Transform*
# owner: GameObject*

+ GetComponentType(): char*
+ Update(): void

IComponentCamera

+ RenderCamera() : void

IComponentParticle

+ RenderParticles() : void

IComponentVisual

+ Render(mat4*, vec3) : void

Material

- textureIds : GLuint[]
- colorLocations : GLuint[]
- matrixLocations : GLuint[]

+ Init(GLuint*) : void
+ Use(mat4*, vec3) : void

Mesh

- vertices : vector<vertex>
- vertexBuffer : GLuint
- elementBuffer : GLuint
- arrayBuffer : GLuint

+ FromFile(char *file) : void
+ Use() : void

Particle

- lifespan : float
- transform : Transform*
- renderer : MeshRenderer*
- material : Material*
- active : bool

+ Particle(vec3*, Mesh*)
+ Reset(vec3, float) : void
+ Update(float, &bool) : void
+ Render(mat4*, vec3) : void

GameObject

- gameObjects : static vector<GameObject*>
# components : vector<IComponent*>
+ transform : Transform*

+ Init() : void
+ Update() : void
+ Render() : void
+ AddComponent(T) : void
+ GetComponent(T) : T

Camera

- perspective : bool
- fov : float
- aspectRatio : float
- nearClipPlane : float
- farClipPlane : float

+ GetProjectionMatrix() : mat4
+ GetViewMatrix() : mat4

ParticleSystem

- particles : vector<Particle*>
- pooledParticles : vector<Particle*>
- mesh : Mesh*
- material : Material*
- lastEmission : clock_t

+ Init() : void
- EmitParticle() : void
+ RenderParticles(mat4*, vec3) : void

MeshRenderer

+ materials : vector<Material*>
+ mesh : Mesh*

+ Render(mat4*, vec3) : void

GLUtils

vertex

+ x, y, z : float
+ r, g, b, a : float
+ tu, tv : float
+ nx, ny, nz : float

Base principle

The component system works using the design pattern of "Observers" - entities that keep track of all of a single type of component and will update them accordingly.

Particle Functionality

The particles have several parameters, such as lifespan, emission direction, mesh, material etc. and will also feature colliders - these will be used when detecting particle collisions.

If used as fluids, however, the planned approach is to mimic the system used by fluid simulators such as the one in Blender: the simulation will require a simulation domain and obstructors (obstacles).

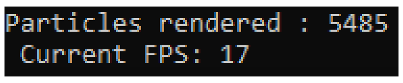
Particle Performace

In order to maintain performance, the particle system pools all inactive particles and attempts to re-use them when a new particle needs to emitted. All active particles are updated by their corresponding Particle Systems and are using the same mesh and material, and therefore only 1 instance of each will be in existence at any given time.

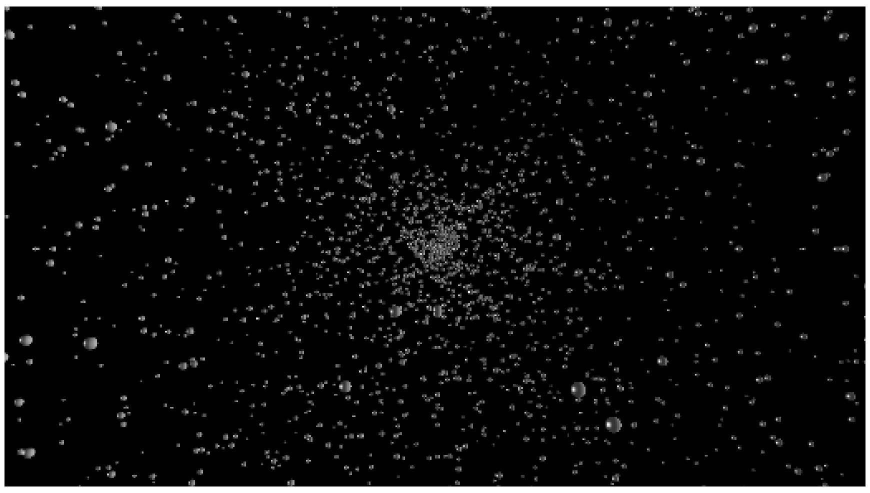
Fluid simulation

The fluid simulation approach to be implemented is known as SPH (Smoothed Particle Hydrodynamics) - and with several open-source implementations at hand, a lightweight implementation is well within scope to be developed.

See: http://www.fluids3.com/

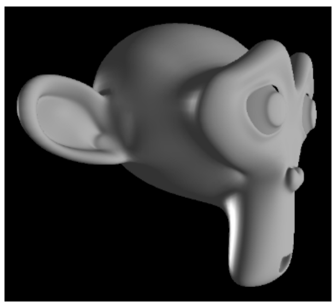


```
Particles rendered : 5485
 Current FPS: 17
```



What went well

The performance is satisfactory: with 4000 particles at 47k vertices each and Blinn-Phong shading the CPU was struggling but maintained over 10 FPS.



The component system can be re-purposed and further developed to suit any needs.

Future improvements

Applying colliders and an SPH algorithm is the next step in finishing the system

Offloading the simulation of particles to the GPU will greatly improve performace.

Adding a Ray Marching shader will create a much more realistic looking fluid simulation.