# Complex Game System

# C++ particle system

# Purpose

The purpose of this particle system to use an advanced algorithm of compute shader to render the particle on the GPU. This will involve using OpenGL (GLM) as a library to run the system.

# Mathematical Operation

Mathematic operation that will be used is OpenGL Mathematics (GLM). This allows the use of glm:: syntax to create vectors and matrices. vectors will be mainly vec4 in a struct of other vec4 that will initialise the values inside the vec4 to be stored into a vector array. Then to have its values change in the compute shader which can’t take in a vec3 without padding so it’s a vec4. Matrices will be used to create a Model View Project matrix (mvpMartix) mat4 that will be times by the position of the particles in the vertex shader to set the gl\_position.

# Advanced Algorithms

Compute shaders is the advanced algorithm to be implemented in the particle system. Doing so means the processing of the particles is done on the GPU and not the CPU but the data of the particles will be pass back and forth the two processers. This will use three different shader the vertex and fragment shader that will link together and the compute shader linked on its own. The buffer VBO of the particles will be stored in the first buffer of an SSBO (GL\_SHADER\_STORAGE\_BUFFER). Then glDispatchCompute will launch the compute shader with the buffer in the SSBO into the compute shader that will update the particles data on the GPU to then be pass back to the CPU to be later used in the vertex and fragment shader.

# Example Test Program

From the main cpp file initialize the particle system class which will create 1000 particles each with a position and a velocity and then in the update function call the update and draw function of the particle system which will store then into a vector array. Storing the vector into the first slot in an SSBO. Then call the update function to dispatch the compute shader of the type of particle to be updated in the compute shader on the GPU. Then call the draw function to render the particles and vertex and fragment shader. This should render in the window particles moving around the screen that has the texture of fire or smoke.

# Integrate System

Integrating the particles system into a new or existing project is to set the project up with OpenGL, then download the file that containing the files needed to run. Particles system cpp and header, particle header, texture cpp and header, ShaderProgram cpp and header files and the compute, vertex and fragment shader and the textures. Put the shaders in the working file and the textures in a file in the working filed called Texture. In visual studio in the source file add the cpp and shader headers. In the main cpp add the include for ShaderProgram and ParticleSystem header file. In the main code glEnable(GL\_BLEND) and glBendFunc(GL\_one, GL\_ONE\_MINUS\_SRC\_ALPHA) for the alpha blending to run. After initialise the default constructor of the particle system and initialise two ShaderProgram with the first passing in the vertex and the fragment shader and the second pass the compute shader. In the game loop call the update of the ParticleSystem passing the smoke or fire and call the draw function after passing the shader of the vertex and fragment.