**GDPHYSX Group 1**

**Phase 1 Documentation**

**OrthoCamera class**

*void SetProjection(float width, float height, float nearPlane, float farPlane)*

* Recalculates the orthographic projection matrix using the given dimensions and clipping planes.

*const glm::mat4& GetProjection() const*

* Returns the current projection matrix.

*const glm::mat4& GetView() const*

* Returns the current view matrix.

*void SetView(const glm::mat4& v)*

* Sets the view matrix (e.g., to apply camera transforms or world offset).

**PerspectiveCamera class**

*void SetProjection(float fovDegrees, float aspectRatio, float nearPlane, float farPlane)*

* Recalculates the perspective projection matrix.

*const glm::mat4& GetProjection() const*

* Returns the current projection matrix.

*const glm::mat4& GetView() const*

* Returns the current view matrix.

*void SetView(const glm::mat4& v)*

* Sets the view matrix (e.g., for camera movement/rotation).

**ModelLoader class**

Responsible for loading 3D OBJ models using TinyOBJLoader and setting up OpenGL buffers (VAO, VBO, EBO).

*static bool LoadObjAndSetupBuffers(const std::string& path, GLuint& vao, GLuint& vbo, GLuint& ebo, size\_t& indexCount)*

* Loads a .obj file from the given path, initializes OpenGL buffers, and outputs them.
* Returns true on success, false on failure (e.g., if the OBJ is missing or malformed).

*void Draw(GLuint shaderProgram, float scale, float x\_rot)*

* Renders the loaded model using the provided shader program.

**PhysicsWorld class**

*void addParticle(Particle\* toAdd)*

* Adds a new particle to the simulation and registers it with the gravity force generator.

*void Update(float time)*

* Removes any destroyed (lifespan-expired) particles using UpdateParticleList().
* Updates all registered forces via forceRegistry.UpdateForces(time).
* Updates each particle’s physics state (position, velocity, lifespan, etc.) by calling Particle::Update(time).

*void UpdateParticleList()*

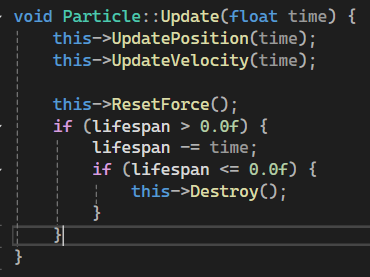
* Iterates through the list of particles and removes any particle for which IsDestroyed() returns true.

**RenderParticle class**

*void Draw(GLuint shaderProgram, float scale, float x\_rot)*

* Renders a single particle only if it's still alive (i.e., not destroyed).
* This function delegates the actual drawing to the associated ModelLoader object, passing in updated transformation data.

**Particle class**



* Each particle has a lifespan in seconds.
* Reduces the lifespan by the time step (time).
* If lifespan hits zero or less, the particle is marked for destruction via this->Destroy();

**Particle spawn logic**





* A new particle is spawned only if the number of existing particles is below a target limit (particleSpawnCount)
* A new particle is spawned only if a timer (spawnTimer) has accumulated enough time to pass a threshold (spawnInterval)
* Ensures that particles spawn one at a time

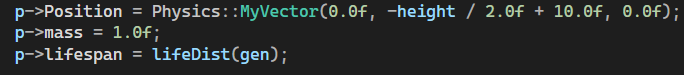


* dt is the time elapsed since the last frame



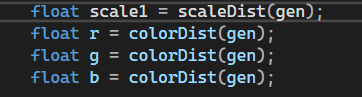
* spawnTimer accumulates that time
* Once spawnTimer exceeds spawnInterval, a new particle is allowed to spawn, then spawnTimer is reduced





* Particles spawn at a fixed Y position
* Particles have a random lifespan (ranging from 1 to 10 seconds)





* Each particle has a random scale and color