GDPHYSX Group 1
Phase 2 Documentation

OrthoCamera class

const glm::mat4& GetProjectionMatrix(float width, float height) const

- Returns the current projection matrix.

```
OrthographicCamera orthoCam(
    glm::vec3(0.0f, -10.0f, 0.0f), // Camera position
    glm::vec3(0.0f, 0.0f, 0.0f),      // Front direction
    glm::vec3(0.0f, 1.0f, 0.0f),      // Up direction
    0.0f,                             // Yaw
    0.0f,                             // Pitch
    50.0f,                            // Distance from target
    400.0f                             // Orthographic size
);

glm::mat4 projectionMatrix = orthoCam.GetProjectionMatrix(800.0f, 800.0f);
```

- Orthographic size is defines the vertical half-size of the orthographic view.
- Since the window is 800x800, this makes the camera view exactly:
- Width = 800 units (400 left, 400 right)
- Height = 800 units (400 down, 400 up)
- This would make it 1m:1pixel.

PerspectiveCamera class

const glm::mat4& GetProjection(float width, float height) const

- Returns the current projection matrix.

LineRenderer class

void drawLine(const glm::vec3& start, const glm::vec3& end)

```
line1.DrawLine(sphereObject.position, ToGlmVec3(aCable.anchorPoint));
line2.DrawLine(sphereObject2.position, ToGlmVec3(aCable2.anchorPoint));
line3.DrawLine(sphereObject3.position, ToGlmVec3(aCable3.anchorPoint));
line4.DrawLine(sphereObject4.position, ToGlmVec3(aCable4.anchorPoint));
line5.DrawLine(sphereObject5.position, ToGlmVec3(aCable5.anchorPoint));
```

- Lines are always drawn from the anchor point to the sphere's position.

sample.frag

```
if (isLine)
    FragColor = vec4(1.0, 1.0, 1.0, 1.0); // white line
else
    FragColor = vec4(modelColor, 1.0);
```

- Lines are rendered in white color, while the spheres/models are rendered based on the values set.

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.

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.

```
float cableLen = 0.0f;
float particleGap = 0.0f;
float particleRadius = 0.0f;
float gravityStrength = 0.0f;
float applyForceX = 0.0f;
float applyForceY = 0.0f;
float applyForceZ = 0.0f;

std::cout << "Enter Cable Length: ";
std::cin >> cableLen;
std::cout << "Enter Particle Gap: ";
std::cin >> particleGap;
std::cout << "Enter Particle Radius: ";
std::cin >> particleRadius;
std::cout << "Enter Gravity Strength (Y-axis, negative for downward): ";
std::cin >> gravityStrength;
std::cout << "Apply Force";
std::cout << std::endl << "X: "; std::cin >> applyForceX;
std::cout << "Y: "; std::cin >> applyForceY;
std::cout << "Z: "; std::cin >> applyForceZ;
```

- At the start of the program, the user would input the values for the cable length, particle gap, particle radius, gravity strength, and external force to be applied to the leftmost sphere/particle.

```
if (!forceApplied && spacePressed) {
    p1.AddForce(Physics::MyVector(Physics::MyVector(applyForceX, applyForceY, applyForceZ)));
    forceApplied = true;
    std::cout << "Force applied to cable!" << std::endl;
}
```

- Force is applied to the leftmost sphere after pressing the spacebar (spacebar can only be pressed once).