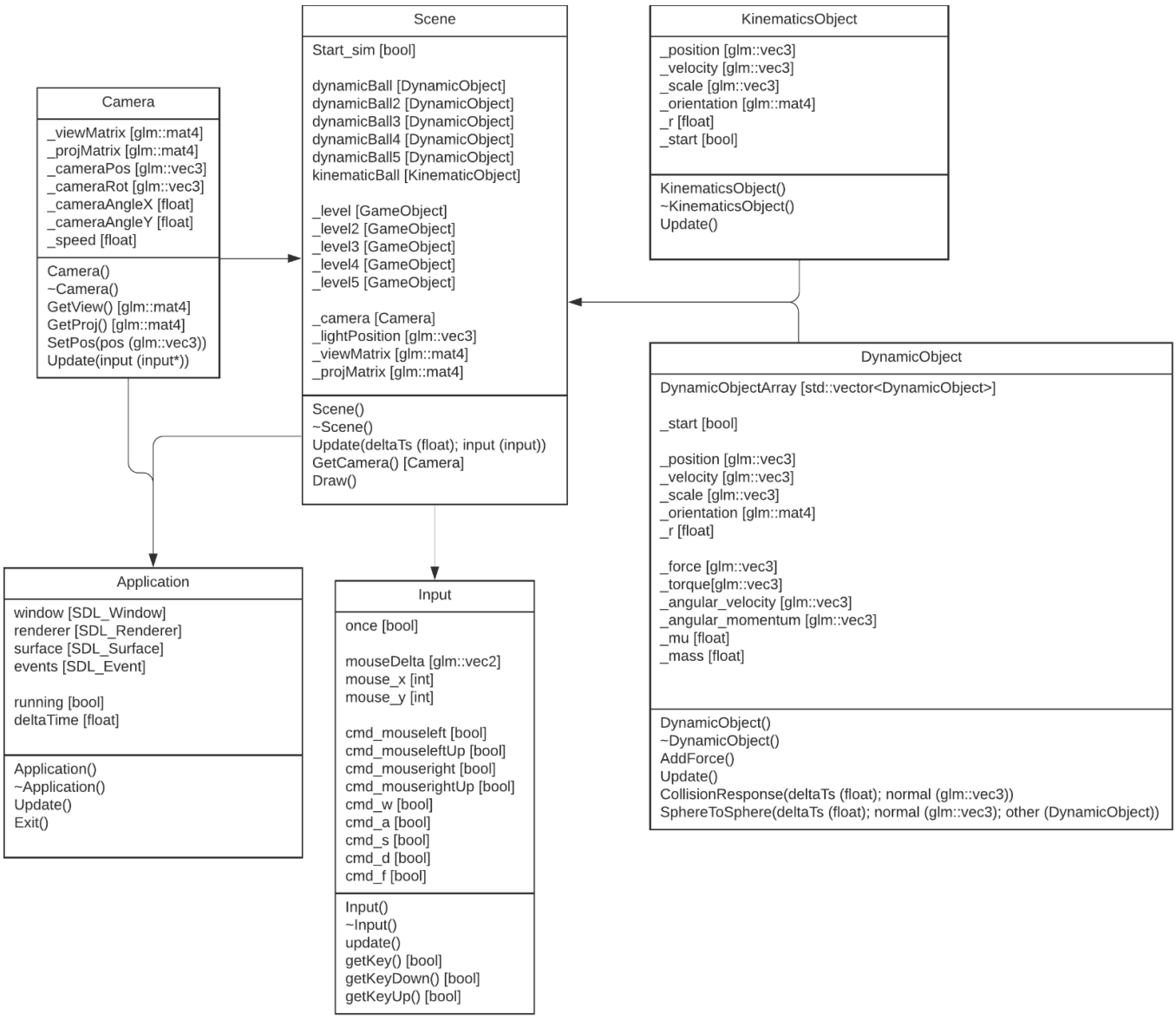
Physics for Games - Code and Report

Overview

The idea behind this project is to demonstrate the effects of dynamic motion between multiple balls in an enclosed space. I will create a box in which multiple balls will be released and interact. One ball will demonstrate kinematics, and the others will use dynamic motion. The balls will all be released at the same time with user input. These balls will be hard-coded into the application as each ball will have different masses, radii and positions. The camera's position will be controlled with the WASD keys and the camera orientation will be controlled with the mouse, allowing the user to view the simulation from any angle / distance they wish. Assets such as textures will be streamed through into the project at runtime alongside fragment shader snippets, which will be stored in a .txt file and accessed when in use. I will use Euler, Runge-Kutta 2nd Order and Runge-Kutta 4th Order integrations.



Research

The most important aspect of the simulation was sphere to sphere collision, as it is the focal point of the project. Online tutorials [(Haubold, 2021), (Mozilla, 2021)], have provided a good explanation and demonstration. The collision works by detecting intersections between two spheres through comparing their positions in space, and then applying impulse to them. This check for collisions happens every tick, (or main loop iteration).

```
bool SphereToSphereCollision(const glm::vec3& c0, const glm::vec3 c1, float r1, float r2, glm::vec3& cp)
{
    float d = glm::length(c0 - c1);
    glm::vec3 n;

    if (d <= (r1 + r2))
    {
        n = glm::normalize(c0 - c1);
        cp = r1 * n;
        return true;
    }
    return false;
}
```

The application of impulse was also a key area of research, since we need to act on the objects involved in the collision(s) and accurately display the outcome. “The impulse method allows us to directly affect the velocities of the simulated objects which have intersected. This is achieved through the application of an impulse, which can be thought of as an immediate transfer of momentum between the two bodies” - [(2017 Tutorial 6 - Collision Response, 2021)].

```
void DynamicObject::SphereToSphere(float deltaTs, glm::vec3 normal, DynamicObject* other)
{
    float bounce = 0.85f;
    glm::vec3 normal_this = glm::normalize(_position - other->GetPosition());
    glm::vec3 normal_other = glm::normalize(other->GetPosition() - _position);
    glm::vec3 ContactForce_this = glm::vec3(0.0f, 9.81f * _mass, 0.0f) * normal_this;
    glm::vec3 ContactForce_other = glm::vec3(0.0f, 9.81f * _mass, 0.0f) * normal_other;

    // "_this" and "_other" are representative of both balls included in the collision
    glm::vec3 r1 = _r * normal_this;
    glm::vec3 r2 = other->GetRadius() * normal_other;

    glm::vec3 Vel_this = _velocity;
    glm::vec3 Vel_other = other->GetVelocity();

    glm::vec3 Vel_this_Relative = Vel_this - Vel_other;
    glm::vec3 Vel_other_Relative = Vel_other - Vel_this;

    float invColliderMass = 1 / other->GetMass();
    float eCof_this = -(1.0f + bounce) * glm::dot(Vel_this_Relative, normal_this);
    float eCof_other = -(1.0f + bounce) * glm::dot(Vel_other_Relative, normal_other);

    float invMass = 1 / _mass;
    float jLin_this = eCof_this / (invMass + invColliderMass);
    float jLin_other = eCof_other / (invMass + invColliderMass);

    glm::vec3 impulseF_this = jLin_this * normal_this / deltaTs;
    glm::vec3 impulseF_other = jLin_other * normal_other / deltaTs;

    AddForce(impulseF_this + ContactForce_this);
    other->AddForce(impulseF_other + ContactForce_other);

    glm::vec3 impulse_angular_this = glm::cross(r1, jLin_this * normal_this);
    glm::vec3 impulse_angular_other = glm::cross(r2, jLin_other * normal_other);
    AddTorque(impulse_angular_this);
```

This simulation sets the balls' velocities to when travelling at a speed close to 0. This is because the balls will continue to move slowly after the simulation has stopped.

```
if (glm::length(_velocity) < 0.05)
{
    _velocity = glm::vec3(0.0f, 0.0f, 0.0f);
}
```

To create the container I added 5 normals and matched their positions with 5 floor models. Each wall is loaded separately. By hard coding each wall in its entirety instead of loading all of the shaders in one block, then all of the textures in one block, etc; I am able to quickly and easily amend a wall model without mistaking one for another.

```
// Repeat for the 4 walls
//right wall (x = 10)
_level2 = new GameObject();
Material* modelMaterial2 = new Material();
modelMaterial2->LoadShaders("assets/shaders/VertShader.txt", "assets/shaders/FragShader.txt");
modelMaterial2->SetDiffuseColour(glm::vec3(0.8, 0.8, 0.8));
modelMaterial2->SetTexture("assets/textures/diffuse.bmp");
modelMaterial2->SetLightPosition(_lightPosition);
_level2->SetMaterial(modelMaterial2);
Mesh* groundMesh2 = new Mesh();
groundMesh2->LoadOBJ("assets/models/woodfloor.obj");
_level2->SetMesh(groundMesh2);
_level2->SetPosition(5.0f, 5.0f, 0.0f);
_level2->SetRotation(0.0f, 0.0f, 1.5708f);

//left wall (x = -10)
_level3 = new GameObject();
Material* modelMaterial3 = new Material();
modelMaterial3->LoadShaders("assets/shaders/VertShader.txt", "assets/shaders/FragShader.txt");
modelMaterial3->SetDiffuseColour(glm::vec3(0.8, 0.8, 0.8));
modelMaterial3->SetTexture("assets/textures/diffuse.bmp");
modelMaterial3->SetLightPosition(_lightPosition);
_level3->SetMaterial(modelMaterial3);
Mesh* groundMesh3  = new Mesh();
groundMesh3->LoadOBJ("assets/models/woodfloor.obj");
_level3->SetMesh(groundMesh2);
_level3->SetPosition(-5.0f, 5.0f, 0.0f);
_level3->SetRotation(0.0f, 0.0f, 1.5708f);

//front wall (z = 10)
_level4 = new GameObject();
Material* modelMaterial4 = new Material();
modelMaterial4->LoadShaders("assets/shaders/VertShader.txt", "assets/shaders/FragShader.txt");
modelMaterial4->SetDiffuseColour(glm::vec3(0.8, 0.8, 0.8));
modelMaterial4->SetTexture("assets/textures/diffuse.bmp");
modelMaterial4->SetLightPosition(_lightPosition);
_level4->SetMaterial(modelMaterial4);
Mesh* groundMesh4 = new Mesh();
groundMesh4->LoadOBJ("assets/models/woodfloor.obj");
_level4->SetMesh(groundMesh2);
_level4->SetPosition(0.0f, 5.0f, 5.0f);
_level4->SetRotation(1.5708f, 0.0f, 0.0f);
```

Analysis

The container holding all of the balls for the simulation was created through the formation of 4 normals oriented around the centre of the scene along the y axis (rotations if each made a square interior) Alongside this, another is placed below on the to act as the base of the box. The wooden floor texture was used to be placed in the same position and rotation as the normals to give off the impression of a solid box in 3D space that interacted with the balls.

Pressing the F key begins the simulation by changing a boolean value from 0 to 1, which, due to being checked in a loop, triggers the other physics routines to begin. The balls are given hard-coded positions on runtime as well as velocities on the start of the simulation, so they have different movements through each axis upon pressing F.

Evaluation

By confining the balls to such a small space, I have successfully demonstrated the collision and impulse of multiple objects. Interacting with both other balls and the walls that create the bounds of the box, the physical reactions are lifelike. To improve upon the simulation, I would take a closer look at the balls reaching / having reached rest. Currently when they reach a velocity of 0, they vibrate and begin to clip through the surface normal. By resolving this issue, the application would reach a genuine finish as the current vibrations have the possibility of starting an impulse reaction out of nothing, breaking Newton's first and third laws [(Newton's Laws of Motion | Glenn Research Center | NASA, 2021)].

References

Haubold, L., 2021. *NeHe Productions: Collision Detection*. [online] Nehe.gamedev.net. Available at: <http://nehe.gamedev.net/tutorial/collision_detection/17005/> [Accessed 5 May 2021].

Developer.mozilla.org. 2021. *3D collision detection - Game development | MDN*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection> [Accessed 5 May 2021].

2021. *2017 Tutorial 6 - Collision Response*. 1st ed. [ebook] ncl, p.2. Available at: <https://research.ncl.ac.uk/game/mastersdegree/gametechnologies/previousinformation/physics6collisionresponse/2017%20Tutorial%206%20-%20Collision%20Response.pdf> [Accessed 21 May 2021].

Glenn Research Center | NASA. 2021. *Newton's Laws of Motion | Glenn Research Center | NASA*. [online] Available at: <https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/newtons-laws-of-motion/> [Accessed 21 May 2021].