2023

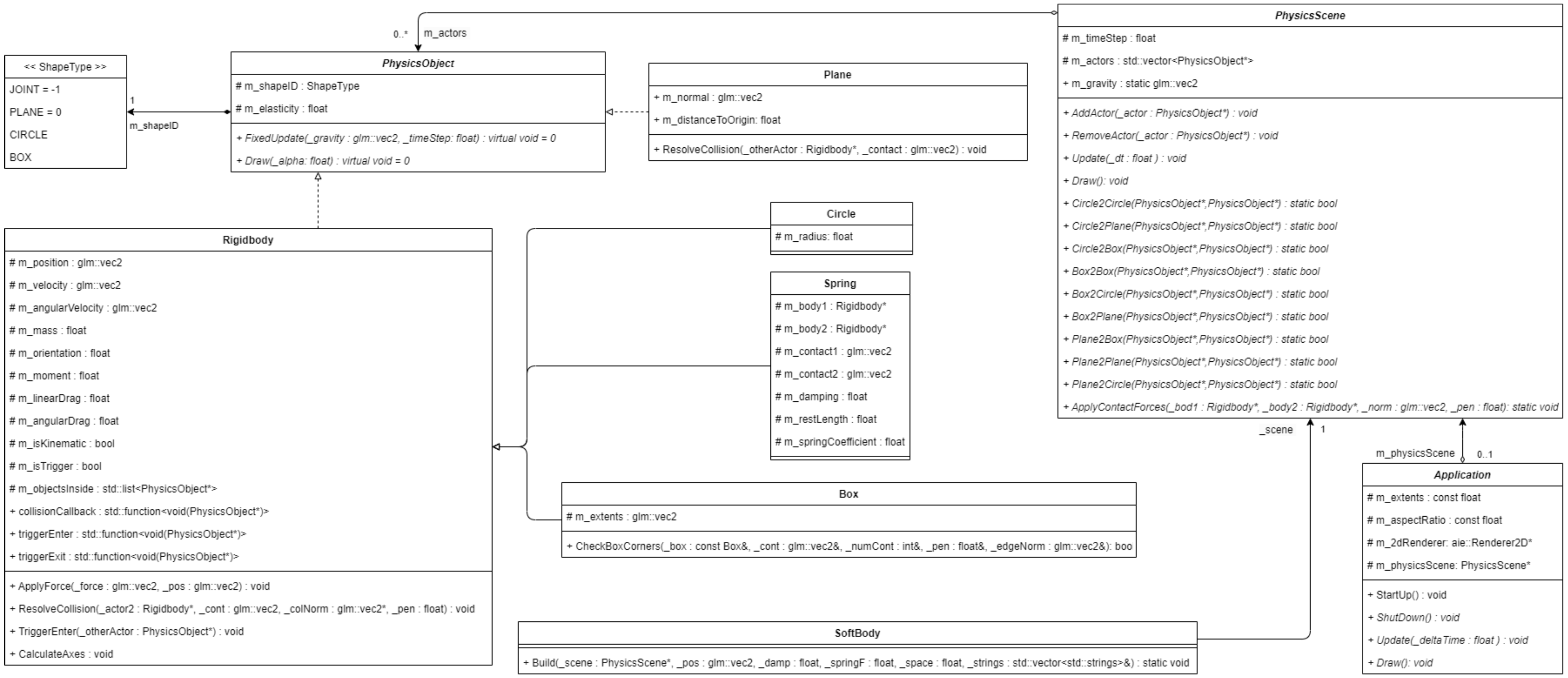
Custom Physics Documentation

NETIX LIBRARY

JUSTIN GREEN

Contents

1.0 - Custom Physics Simulation Class Diagram


0..*
m_actors
<< ShapeType >>
JOINT = -1
PLANE = 0
CIRCLE
BOX
1
m_shapeID
PhysicsObject
# m_shapeID : ShapeType
# m_elasticity : float
+ FixedUpdate(_gravity : glm::vec2, _timeStep: float) : virtual void = 0
+ Draw(_alpha: float) : virtual void = 0
Plane
+ m_normal : glm::vec2
+ m_distanceToOrigin: float
+ ResolveCollision(_otherActor : Rigidbody*, _contact : glm::vec2) : void
PhysicsScene
# m_timeStep : float
# m_actors : std::vector<PhysicsObject*>
+ m_gravity : static glm::vec2
+ AddActor(_actor : PhysicsObject*) : void
+ RemoveActor(_actor : PhysicsObject*) : void
+ Update(_dt : float) : void
+ Draw(): void
+ Circle2Circle(PhysicsObject*,PhysicsObject*) : static bool
+ Circle2Plane(PhysicsObject*,PhysicsObject*) : static bool
+ Circle2Box(PhysicsObject*,PhysicsObject*) : static bool
+ Box2Box(PhysicsObject*,PhysicsObject*) : static bool
+ Box2Circle(PhysicsObject*,PhysicsObject*) : static bool
+ Box2Plane(PhysicsObject*,PhysicsObject*) : static bool
+ Plane2Box(PhysicsObject*,PhysicsObject*) : static bool
+ Plane2Plane(PhysicsObject*,PhysicsObject*) : static bool
+ Plane2Circle(PhysicsObject*,PhysicsObject*) : static bool
+ ApplyContactForces(_bod1 : Rigidbody*, _body2 : Rigidbody*, _norm : glm::vec2, _pen : float): static void
Circle
# m_radius: float
Spring
# m_body1 : Rigidbody*
# m_body2 : Rigidbody*
# m_contact1 : glm::vec2
# m_contact2 : glm::vec2
# m_damping : float
# m_restLength : float
# m_springCoefficient : float
Rigidbody
# m_position : glm::vec2
# m_velocity : glm::vec2
# m_angularVelocity : glm::vec2
# m_mass : float
# m_orientation : float
# m_moment : float
# m_linearDrag : float
# m_angularDrag : float
# m_isKinematic : bool
# m_isTrigger : bool
# m_objectsInside : std::list<PhysicsObject*>
+ collisionCallback : std::function<void(PhysicsObject*)>
+ triggerEnter : std::function<void(PhysicsObject*)>
+ triggerExit : std::function<void(PhysicsObject*)>
+ ApplyForce(_force : glm::vec2, _pos : glm::vec2) : void
+ ResolveCollision(_actor2 : Rigidbody*, _cont : glm::vec2, _colNorm : glm::vec2*, _pen : float) : void
+ TriggerEnter(_otherActor : PhysicsObject*) : void
+ CalculateAxes : void
Box
# m_extents : glm::vec2
+ CheckBoxCorners(_box : const Box&, _cont : glm::vec2&, _numCont : int&, _pen : float&, _edgeNorm : glm::vec2&): boo
_scene
1
m_physicsScene
0..1
Application
# m_extents : const float
# m_aspectRatio : const float
# m_2dRenderer: aie::Renderer2D*
# m_physicsScene: PhysicsScene*
+ StartUp() : void
+ ShutDown() : void
+ Update(_deltaTime : float) : void
+ Draw(): void
SoftBody
+ Build(_scene : PhysicsScene*, _pos : glm::vec2, _damp : float, _springF : float, _space : float, _strings : std::vector<std::strings>&) : static void

2.0 - Custom Physics Simulation Interactions

My custom physics simulation demonstrates most aspects of real-world physics that can be applied to a 2d environment. What this means is that real world interaction of physics bodies can be simulated together within a real-time application and interact both as dynamic and static objects.

The Netix library uses no third party physics library and allows for simulations of static and dynamic rigid bodies, applied forces to dynamic rigid bodies and the expected interaction of static and dynamic rigid bodies.

This library implements rigid bodies to simulate common objects. These objects have basic attributes such as position, mass and velocity to more advance attributes like orientation, moment, and drag both linear and angular. Static objects can also be simulated as kinematic bodies and are treated with infinite mass.

There are only 3 primitive types within this library, dynamic circles, dynamic boxes and static planes each with their own collision detection. All objects detect collision with planes by using velocities and the plane’s normal. Sphere collisions use simple radius and distance detection. The boxes are oriented bounding boxes which use extents and contact points to detect collision.

Netix uses the concepts of Newton’s three laws of motion to provide a realistic model of physics implementation. These laws are applied by treating rigid bodies as a single point mass and finding the acceleration of the point for given forces by dividing the force by the mass which provides acceleration that can be used to calculate the objects movement.

Newton’s laws are also applied to the collision resolution between two physics bodies. When it comes to collision resolution the conservation of momentum is used as a core concept and a key indicator for the accuracy of applied impulsive forces and the collision normal for objects. This ensures expected interaction between both rigid and static bodies.

The Netix library demonstrates a physics simulation for a 2d environment through the implementation of dynamic and static rigid bodies, accurate application of force and by displaying the expected interaction of two physics objects.

3.0 - Custom Physics Simulation Potential Improvements

Although the Netix library can provide for accurate enough physics simulations, there are a couple of improvements that can be made to improve accuracy and efficiency of the simulations being made, such as the implementation or toque and quadtrees.

3.1 - Improvement #1: Torque

In Netix there are many rotational equivalents to translational physics equations, this can also occur in force. Force alters translational motion of objects, there is a rotational counterpart to this called torque which alters the rotational motion of an object around an axis.

Implementing force means the implementation of three main concepts; the further force is applied from an objects axis of rotation the more angular acceleration there is, effectiveness of force depends on the angle of which the force is applied and force magnitude must be involve. Another thing to note is that torque is either clockwise or counterclockwise relative of the object’s pivot point. Torque can be calculated by multiplying applied force by the perpendicular distance from the rotational axis to the line of action of the force.

3.2 - Improvement #2: Quadtrees

Performing collision checks against every object in a scene every frame can get quite costly, this is where quadtrees can become useful. Quadtrees are partition based data structures (sometimes referred to as bounding volume hierarchies), which recursively subdivide equal amounts of space into smaller sections using trees, nodes and leaves.

Quadtrees specifically are subdivided into four smaller squares at each recursive step and can be implemented as a collision detection by storing objects within leaves of the quadtree. These objects are inserted into these leaves by checking if the object intersects with a node, then recurses until leaf level is reached, and are then added to the collection. Collision detection occurs when exacting the steps to insertion however the object is tested for collision against all objects the collection

4.0 - Visualised Game Using Your Custom Physics Simulation

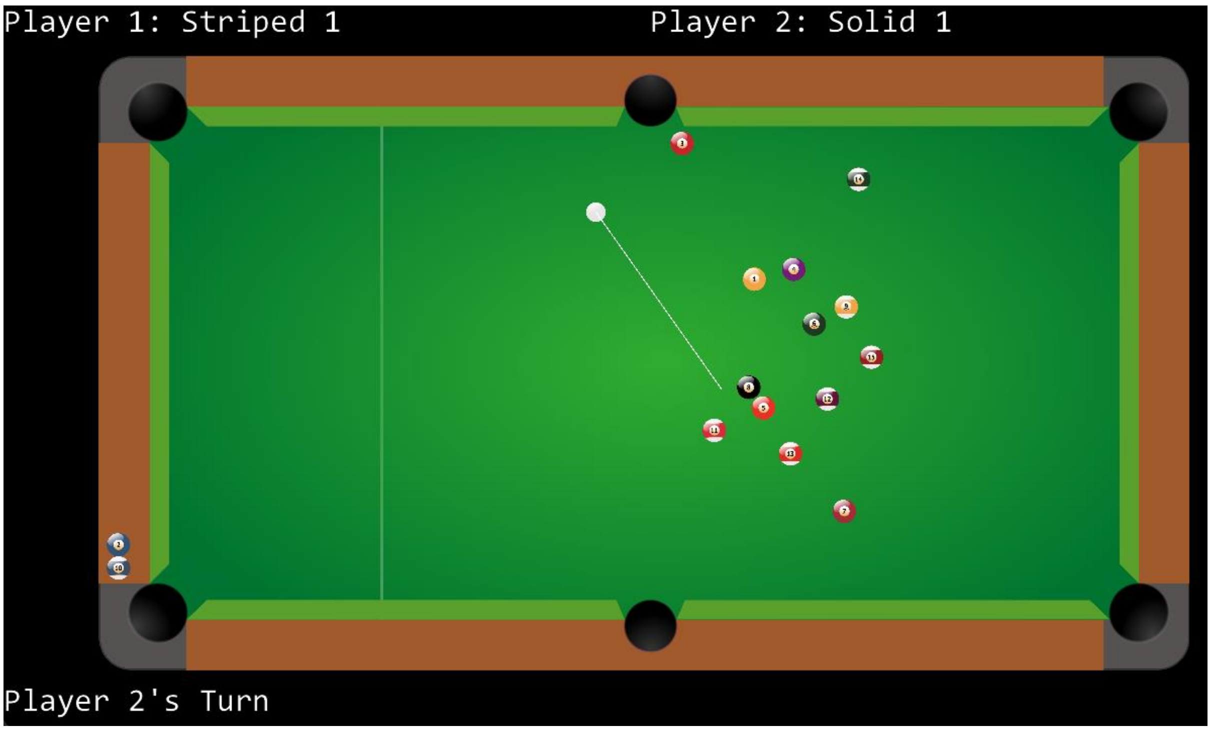
Make a statement as to the meaning or interpretation of something, giving sufficient detail as to allow it to be distinguished from similar things.

The visualised physics simulated game I created using my custom physics library was Eight Ball pool, as this game is an ideal framework that showcases basic laws of physics while also providing an enjoyable two player experience.

The game was made by using Netix primitives such as dynamic circles, static boxes and invisible planes as the boundaries. The pool balls where all made a child of the circle primitive, with the mass of real pool balls, and the pool table didn't have gravity however friction of a table was simulated with linear and angular drag.

The Eightball pool fundamentals such as sinking the pall and checking for fouls were created using the collision call-backs and collision triggers also apart of the Netix library.

Overall, the Eightball pool visualisation is a perfect example of the Netix libraries capabilities.



5.0 - Third Party Libraries

The only third party library which was used was in the making of Netix was the AIE Bootstrap. The AIE Bootstrap is an OpenGL wrapper which was used in Netix to mainly display Gizmos.

6.0 - References

Moebs, W., Sanny, J. and Ling, S.J. (2016) 10.6 torque - university physics volume 1, OpenStax. OpenStax. Available at: https://openstax.org/books/university-physics-volume-1/pages/10-6-torque (Accessed: February 21, 2023).

Ambler, S.W. (2004) UML 2 class diagrams: An agile introduction, AgileModeling.com. Available at: http://www.agilemodeling.com/artifacts/classDiagram.htm (Accessed: February 21, 2023).

Bell, D. (2004) The UML 2 class diagram, IBM developer. Available at: https://developer.ibm.com/articles/the-class-diagram/ (Accessed: February 21, 2023).

ERICSON, C. (2005) Real-time collision detection. Amsterdam ; Boston ; Heidelbergetc.: Elsevier.

mikolalysenko (2015) Collision detection (part 2): Box intersection, 0 FPS. Available at: https://0fps.net/2015/01/18/collision-detection-part-2/ (Accessed: February 21, 2023).

O., D. (2011) Quadtree for 2D collision detection, Stack Overflow. Available at: https://stackoverflow.com/questions/4981866/quadtree-for-2d-collision-detection (Accessed: February 21, 2023).