Technical Documentation for 2D Physics Engine

Contents

Intro

About

The app is a Physics Library for 2D games and simulations. It allows for collision definition, detection and resolution. The main points of the library are definition of the world, its structures and physical bodies.

Requirements

The library has been tested on WSL:Ubuntu and Windows 10 with at least 4GB RAM compiled with both MSVC and G++ (the demo). The code itself is written in C++. There is no need for any external downloads unless using the SFML Demo, in which case the user needs to download necessary libraries included in the build instructions.

Installation

After downloading/cloning the source code, the user has to follow the build instructions for the specific platform, which are included in the build_instructions.md file. To include the library in another project, the user can follow the CMakeLists.txt file for the demo.

Code Structure

The code in its full shape is made in a non-strict OOP fashion, so while there are some similarities to design patterns, there is no explicit use of any. The whole library uses mostly raw pointers when passing the data around. The reason for that is that the library itself is not responsible for the body storage, this is mainly due to the fact that the users might have different needs for the object management.

World

Body

Class representing a single physical object in the simulated world.

It stores data essential for collision detection.

Body Shape

The abstract class for a shape of a physical object in the simulation world. Acts as a prototype for the shapes, since they are normally used only for collision. (Could have been done using the prototype design pattern)

Circle

Represents a circular shaped physical object. Inherits from the parent abstract class body shape, overrides the methods from the base class and defines its own getter method for radius.

Rectangle

Represents a non-angled rectangle shaped physical object. Inherits from the parent abstract class body shape, overrides the methods from the base class and defines its own methods for getting the half extents of the rectangle.

Polygon

Represents a convex (for now only regular) polygon (angled multi-vertex shape) as a body shape for a physical object. Inherits from the parent abstract class body shape, overrides the methods from the base class and defines its own methods for getting the half extents of the rectangle.

Ellipse

TODO:

Manifold

Struct representing the points of collision between the two objects, including the normal and penetration of the collision. Used mainly for utility purposes since it would not be nice to move around unnamed tuples (could be assigned an alias though).

Collision Manager

Class defining an abstract collision resolver, which applies a response to the participating collision bodies. Could be either one of the predefined ones (Impulse resolution or position correction) or the user can also define their own collision manager, just by passing a function to the custom collision manager. The object’s solve collision method will then automatically call this custom function while updating the world.

Collision Grid

A class providing a spatial optimization structure, which is used during the broad phase of the collision detection. The grid is basically a matrix of cells (to be exact, it is a vector<vector<vector<body*>>>), so it ensures collision only with the bodies inside this “cell“. It mainly contains methods used only by the library itself, since there is no need for the user to manipulate with the contents of the grid outside of setting its initial dimensions.

2DMath

Provide basic mathematical structures and functions necessary for manipulation with 2D physical objects like 2D vector and 2x2 matrix. Also a definition for a transform is added, which is basically just the position and angle compressed into a single structure. For swifter manipulation with vector math, there are also helper methods allowing for simpler float math on the whole object.

What more could be done?

The collision detection is not sharp enough, can be done much better.

The collision resolution is very hard to visualize properly and slopes are a huge problem.

More shapes.

Better memory management (shared pointers)