DigiPen
INSTITUTE OF TECHNOLOGY
EUROPE - BILBAO

CS 230 | Game Implementation Techniques
Project 3 | SANDBOX (Physics & Collisions)

Topics

The assignment will cover the following topic

I. Physics simulation using various integration methods.
II. Static collisions with contact data calculation.
III. Basic implementation of collision engine.

Goal

Building on the previous assignments (Math + primitive overlap tests), we're going to extend our engine capabilities by adding rigid body physics and collision detection/resolution. After completing this assignment, students should have a naïve, yet robust physics engine handling collisions between circles, AABBs and OBBs.

Description

You are provided with one Visual Studio (2013) project called 03.Sandbox.vcxproj:

- The project properties and folder organization is setup for you, so you don't have to change anything.
- The project assumes that a folder AECore (containing the latest Alpha Engine API) is located in the same folder as the solution (similar to the previous assignment).
- The project also relies on the systems implemented in GAM 150 lab 1 through 6. These are provided to you in the form of a static library (GAM150Lab6.lib). **Note: You don't need to interact with (or even know about) those files to complete the assignment.**

This project depends on the Alpha Engine and on the implementations of the previous assignments. **You will need to copy your implementation from the previous assignments to this project (in the 'src\Engine' folder and subfolders).**

The header files for the files implemented in the previous assignments is provided for you. Simply paste the source file into the corresponding folder. The files you should copy from previous assignments are listed below. **Note: You will not be graded on the previous assignment files.**

The following sections describes in more details the implementation and follows a recommended order for you to follow in order to implement the assignment with the most efficiency.

Preparing the engine

- You are provided with an incomplete engine with the modules listed below, you must copy the files in **bold** from previous assignments in the following folders:
  - Engine\Utils\
    - GameStateManager.h (complete).
    - GameStateManager.cpp (missing – from assignment 1).
  - Engine\Math\
    - Vector2.h (complete).
    - Vector2.cpp (missing – from assignment 2).
    - Matrix33.h (complete).
    - Matrix33.cpp (missing – from assignment 2).
    - Transform2D.h (complete).
    - Transform2D.cpp (missing – from assignment 2).
    - Collisions.h (complete)
    - Collisions.cpp (missing – from assignment 2).
    - ContactCollisions.h (complete)
    - ContactCollisions.cpp (provided – incomplete – TODO)
    - Polygon2D.h/.cpp (provided – incomplete – TODO – extra credit)
  - Engine\Physics\
    - CollisionsSystem.h (complete)
    - CollisionSystem.cpp (provided – incomplete – TODO)
    - RigidBody.h (complete)
    - RigidBody.cpp (provided – incomplete – TODO)
  - Engine.h (complete)

- This file simply includes all header files listed above.

-

- The files labeled with **(provided – incomplete – TODO)** contain empty function definitions that you must complete yourself. Each function you must implement is labeled with a @TODO. Execute a 'Find All' (Ctrl + Shift + F) command in Visual Studio to locate them.

Suggested Order of Implementation

The assignment demo levels is organized such that it is easier to start implementing the functions in **ContactCollisions.cpp**. Level_1, Level_2, Level_3 serve as a testing for these functions, and follow the order of suggested implementation, that is, from easier to harder:

- `StaticCircleToStaticCircleEx`
- `StaticRectToStaticCircleEx`
- `StaticOBBToStaticCircleEx`
- `StaticRectToStaticRectEx`
- `OrientedRectToOrientedRectEx`

Once you have finished implementing these functions, then you can start working on the collision resolution function `ResolveContactPenetration` defined in **CollisionSystem.cpp**. The demo levels will call this function if you press the space key.

After this, you should implement the functions in RigidBody

```
void IntegrateEuler(float timeStep);
void IntegrateNewton(float timeStep);
void IntegrateVerlet(float timeStep);
void ComputeAABB(Vector2 * outPos, Vector2 * outSize);
```

Then finish by implementing `ResolveContactVelocity` and `CollideAllBodies` in **CollisionSystem.cpp** to test the rest of the code in the Sanbox level.

MORE DETAILS ARE PROVIDED IN THE FUNCTION HEADERS OF THE FUNCTIONS THAT YOU MUST IMPLEMENT, SO MAKE SURE TO READ THE COMMENTS THOROUGHLY.

DEMO DESCRIPTION AND CONTROLS PROVIDED IN THE README.TXT FILE IN THE DEMO FOLDER.

Extra Credit

What to Submit

Since the only code that is required for you to write is located in ContactCollisions.cpp, RigidBody.cpp, and CollisionSystem.cpp (EXTRA CREDIT: also submit Polygon2D.cpp), these are the ONLY files that you will submit, so make sure that your implementation doesn't rely on any other variable. (You should not create any extra variables to get the basic functionality working).

Code Quality

Each line of your code that is not *trivially obvious* must be commented. You can use C-style commenting or C++-style.

Each Function that you create apart from the one provided must have a header that explains what the function does and that includes:

- Function Name.
- Purpose.
- Parameters.
- Return value.

FOR THIS ASSIGNMENT, THE FUNCTION HEADERS ARE PROVIDED FOR YOU.

Additionally, you are **REQUIRED** to submit a README.txt file that is formatted as follows:

- Name/Student ID:
- Login:
- Project: For this project, just write "SANDBOX".
- Additional Notes: Describe your extra credit part and how to use it here.
- Known Bugs: List any bugs that your submitted implementation has. If you had no bugs, leave this field blank.
- Time of implementation: This is the time it took you to complete the implementation.
- Time of testing: This is the time that you spent testing your work.
- Comments: Leave any comments that you might have about the assignment here.

Place both README and submission files into a compressed .zip folder following the usual guidelines described in the first page.

Finally, each ".cpp" and ".h" file in your homework should include the following header:

```
// ---------------------------------------------------------------------------
// Copyright (C)DigiPen Institute of Technology.
// Reproduction or disclosure of this file or its contents without the prior
// written consent of DigiPen Institute of Technology is prohibited.
//
// File Name:    <put file name here>
// Purpose:   <explain the contents of this file>
// Project:   <specify student login, class, and assignment.
//            For example: if foo.boo is in class CS 230 and this file is a part
//            of assignment 3, then write: CS230_fooboo_3>
// Author:       <provide your name, student login, and student id>
// ---------------------------------------------------------------------------
```