DigiPen Institute of Technology
CSD1130
Game Implementation Techniques
Assignment 4 – Part2
Cage

Due date:
Section A: Thursday Mar 23rd, 2023, at 2:00pm
Section B: Thursday Mar 23rd, 2023, at 4:30pm

Topics
The assignment will cover the following topics:

1. Integrate the previously implemented math library.
2. Collision intersection between moving circles and static line body.

Goal
The goal of this assignment is to implement the intersection functions between the moving disk and a stationary wall. The output of the assignment would be a disk bouncing off walls.

Description

- ✓ A start-up application will be provided – Do open/use **MSVS 2019**.
- ✓ Language: C (C++ "environment / setup only").
- ✓ A library will be provided, which includes several hardware related functions like initializing, updating, and freeing the graphics and input engines.
  - Library name: "Alpha_Engine"
  - The header files and the lib files of the "Alpha_Engine" library are included in the solution folder.

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

```
/* Start Header ************************************************************/
/*!
\file        <put file name here> (e.g. main.cpp)
\author      <provide your name, student login, and student id>
             (e.g. DigiPen Singapore, Singapore, 60001906)
\par         <provide your email address> (e.g. email: digipen\@digipen.edu)
\date        <date on which you created this file> (e.g. Mar 01, 20xx)
\brief       <give a brief description>

Copyright (C) 20xx 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.
 */
/* End Header **************************************************************/
```

DigiPen Institute of Technology
CSD1130
Game Implementation Techniques
Assignment 4 – Part2
Cage

Assignment Submission **(check Grading Algorithm section)**

- To submit your programming assignment, organize a **folder** consisting of the following:
  - The solution **.sln** file
  - The project folder named "**CSD1130_Cage_Part2**".
  - Any other folder that your project depends on, to build and run.
- In other words, *your submission must be ready for compiling and linking by the grader*.
- Name this **folder** using the following convention:
  - **<class>_<section>_<student login name>_<assignment#>_<part#>**
  - For example, if your login is ***foo.boo*** and assignment 1 part 1 is being submitted, your **folder** would be named **csd1130_a_foo.boo_1_1**
- Your **folder** must not contain *Debug*, *Release*, *x64* and *x86* folders, *object files* or *executable* files.
- Your **folder** must not contain the **.tmp*, **.db*, **opendb*, **.sdf* or **.opensdf* files.
- Do delete the "**.vs**" read-only file. (*make sure windows explorer can show hidden files. ".vs" folders are usually set to hidden*). This will reduce the size of your submission file by a lot.
- The provided project template is already organized in a way that source files and header files have their own folders, as well as a "Bin" folder that will contain the generated output from "Release" or "Debug" versions. When submitting, make sure to restore the "Bin" folder to its **original** state.
- Do not change or update the following code files: "main.cpp", "GameStateMgr.h", "GameStateMgr.cpp", "GameStateList.h", "GameState_Cage.h".
- Zip this **folder** and name the resulting file using the following convention:
  <class>_<section>_<student login name>_<assignment#>_<part#>.zip
  For example, if your login is ***foo.boo*** and you are submitting assignment 1 part 1, your zipped file would be named as: **csd1130_a_foo.boo_1_1.zip**

- Next, upload your zip file after logging into the course web page using the link https://distance3.sg.digipen.edu.

- Finally, perform a **sanity** check to determine if your programming submission follows the guidelines by downloading the previously uploaded zip file, unzipping it, then compiling, linking, and executing your submission.

DigiPen Institute of Technology
CSD1130
Game Implementation Techniques
Assignment 4 – Part2
Cage

Using the right Compiler and MSVS setup

- The project must be tested in **RELEASE and DEBUG** modes with **warning level 4**, under **x64** platform. Under project settings the **SDK Version**: must be set to *10.0.…(latest installed version)* and the **Platform Toolset** set to *Visual Studio 2019 (v142)*. It must generate 0 warnings and 0 errors. This can be verified on a PC located at **Pascal lab**.
- Please validate the previous statement before your submission!

Implementation

- The following collision intersection functions should be implemented under "Collision.cpp":
  - ```
    void BuildLineSegment(
    LineSegment &lineSegment,
    const AEVec2 &p0,
    const AEVec2 &p1);
    ```
    - `LineSegment &lineSegment`: this parameter should return a line segment from two points pt0 and pt1. In addition, the normal of this line segment should be outward and normalized.
    - `const AEVec2 &p0`: this parameter should contain the first point edge "P0" of the line segment.
    - `const AEVec2 &p1`: this parameter should contain the second point edge "P1" of the line segment.

    To use this function, read the "NOTE to student" comment, under "GameStateCageInit" function, in "GameState_Cage.cpp" file.

  - ```
    int CollisionIntersection_CircleLineSegment(
    const circle &circle,
    const AEVec2 &ptEnd,
    const LineSegment &lineSeg,
    AEVec2 &interPt,
    AEVec2 &normalAtCollision,
    float &interTime
    bool & checkLineEdges);
    ```
    - `const circle &circle`: is an object that is passed by reference that contains the properties of the bounding circle of the disk which are the starting position and the radius.
    - `const AEVec2 &ptEnd`: is the disk's position at the end of the current frame.
    - `const LineSegment &lineSeg`: is an object that contains the properties of the line segment that bounds the wall which includes the starting and ending position of the wall with the normal that is **normalized** by calling the `BuildLineSegment` function.
    - `AEVec2 &interPt`: this parameter should contain the position of the disk at intersection with the wall.
    - `AEVec2 &normalAtCollision`: this parameter should contain the vector normal, at collision point (if any).

- `float &interTime` : this parameter should contain the value of the intersection time
- `bool & checkLineEdges:` when it is set to true, this will allow the check of collision with the edges of the line segment.
- The return type is an integer where value of 1 is for intersection and 0 is for no intersection.

- ```
  void CollisionResponse_CircleLineSegment(
  const AEVec2 &interPt,
  const AEVec2 &normal,
  AEVec2 &ptEnd,
  AEVec2 &reflected);
  ```
  - `const AEVec2 &interPt:` this parameter holds the coordinates of the position of the disk at intersection.
  - `const AEVec2 &normal:` this parameter holds the value of the normal of the wall on which the disk will reflect. It could be an outward normal or a normal formed at an edge of the wall.
  - `AEVec2 &ptEnd:` this parameter will contain the new position of the disk after response.
  - `AEVec2 &reflected:` this parameter will contain the new reflected normalized vector of the disk.

- Extra Credits (in case you would like to work on it)
  1. Collision Circle with Line Segment edges:
     a. The following collision intersection function should be implemented:
        `int CheckMovingCircleToLineEdge(…);`
        - Check the parameters description in the “Collision.h” file
  ```

DigiPen Institute of Technology
CSD1130
Game Implementation Techniques
Assignment 4 – Part2
Cage

- You are given a project that contains the debug and release version of the library. The library "Alpha_Engine(D).lib" contains the implementation of the following:
    - Collision intersection and response algorithms

- When you first run the project, you will see **five** disks moving and bouncing on **eleven** reddish walls. The creation of the disks and the walls is read from a file called "*LevelData - Original.txt*". The format of the text file is simple. The first line contains a number that describes the number of disks we have in the level. Next comes the description of the disks. The same goes for the walls.

- First step in your project you should replace the entire alpha engine math with your own math library (Matrix3x3.h/.cpp and Vector2D.h/.cpp) that you implemented in assignment 4 (part 1).
  In the **Collision.h**, **Collision.cpp**, and **GameState_Cage.cpp** files.
    - Replace **AEVec2** with **CSD1130::Vec2**
    - Replace **AEMtx33** with **CSD1130::Mtx33**

- The implementation of the collision intersection and response is already done in the Alpha Engine library under the function names "**AE**`CollisionIntersection_CircleLineSegment`" and "**AE**`CollisionResponse_CircleLineSegment`".
  In order to use your own implementation, you must remove the '**AE'** letters from the beginning of the function's names. As for the parameters, replace **AELineSegment** with **LineSegment** and **AECircle** with **Circle**. This applies for all the instances you can find in the project.

- Replace "**AE**`BuildLineSegment`" function call with your "`BuildLineSegment`" function. Remove the '**AE'** letters. This applies for all the instances you can find in the project.
  Follow "NOTE to student" comment, under "GameStateCageInit" function, in "GameState_Cage.cpp" file.

- In case you didn't use all the intersection function's parameters, just ignore those parameters, if the collision behavior is as requested. Validate that you do not get any compile warnings!

DigiPen Institute of Technology
CSD1130
Game Implementation Techniques
Assignment 4 – Part2
Cage

Planning to work without Extra Credits:

- You must follow the "**Moving Circle vs Static LNS – part1**" file method, provided onto the Moodle course site and simply ignore "*CheckMovingCircleToLineEdge()*" section.
    - At "*CheckMovingCircleToLineEdge()*", just "return 0".

- The main mechanics of working without extra credits is to collide and reflect the circles from the bodies of the line segments **only**, **without** the edge's collision detection.

- Set "int EXTRA_CREDITS = 0;" in "GameState_Cage.cpp" file.
  While we grade this assignment, we will follow the "EXTRA_CREDITS" variable value, you have submitted. We will grade based on its value. So, if it is set to 0, it means you do not want us to grade the extra credit section.

Planning to work with Extra Credits:

- You must follow the "**Moving Circle vs Static LNS – FULL**" file method, provided onto the Moodle course site.

- The main mechanics of working with extra credits is to collide and reflect the circles from the bodies of the line segments **and from their** edge's P0 and P1.

- Set "int EXTRA_CREDITS = 1;" in "GameState_Cage.cpp" file.
  While we grade this assignment, we will follow the "EXTRA_CREDITS" variable value, you have submitted. We will grade based on its value. So, if it is set to 1, it means you want us to grade the extra credit section.

- The creation of the disks and the walls is read from a file called "*LevelData - Extra Credits*".

DigiPen Institute of Technology
CSD1130
Game Implementation Techniques
Assignment 4 – Part2
Cage

Testing Your Application (Do not include this in the final submitted project)

Before submitting any code, always perform test cases. Few test cases that you can check:

- Change the radius of the circles and check for intersections.
- Add more circles of different sizes.
- Add more lines of different lengths and slopes.
- Draw debugging data (e.g. line normal, disk's velocity and reflected vector etc…)

Evaluation

Here are the most common reasons assignments are marked down:

- Project does not compile/build.
- Project does not build without warnings.
- One or more items in the "Requirements" section was not satisfied.
- A fundamental concept was not understood.
- Code is sloppy and hard to read (e.g. indentation is not consistent, no comments, etc…).
- Your solution is difficult (or impossible) for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, etc…
- Project assignment was turned in late.

Grading Algorithm

This project will be graded according to the following rules. Any rule that is missing, letter grades will be deducted from the project's grade:

- Not following the submission guidelines (from above section).
- Missing header comment, at the top of the ".h" and ".cpp" files:
  - For any amount of files
  - Penalty applies, only, on the files that the students update or add.
- Header comment does not match the given one or does not contain the necessary information:
  - For any amount of files
  - Penalty applies, only, on the files that the students update or add.
- Compile errors or does not compile for some reasons.
- The demo is crashing at runtime:
  - Demo crashes right from the beginning.
  - Demo crashes while playing, or at random places.
- Compile Warnings (Unique Instances) for any number of warnings:
    - IntelliSense warnings are not penalized.
- Implemented functions:
  - Not using your custom math library (from project 4 part 1):
      - Partial usage.
      - Not using it at all.
  - Not replacing all the Alpha Engine collision functions and parameters with your custom functions and parameters.
  - Incorrect collision intersection behavior:
      - A buggy output but works most of the time -> partial penalty.
      - Very wrong collision, or collision behavior is not correct. For example, the circle does not stop exactly at the border of the body of the line segment -> full penalty.
  - Incorrect collision response behavior. The reflection is not correct.
  - Incorrect build of line segment function. Example, Normal is not outward or normalized.
- Checking the source code:
  - No comments.
  - No consistent indentation.
  - No appropriate identifiers.

- Have memory leaks!
  - Code to check memory leaks was given in assignment 2 (it is added in main.cpp).
- Extra Credits will be awarded accordingly!

Notes

- Depending on different implementation you might need to add new headers or .cpp files. You can add new code files by respecting the project folder organization.
- Do **NOT** change the "C" structure of the assignment to "C++".
- Your collision and response behavior must mimic the original AE behavior, with minor variations. For example, the positioning and movement direction of the dynamic circles may differ while running the demo for a longer time. That is due to the delta time, that is not a fixed value throughout the application lifetime.
- Your demo must run correctly for at least 5 minutes time!