Real-Time Rendering and 3D Games Programming

ASSIGNMENT 2 (v1.0)

Assessment:	50%
Deadline:	11:59pm Sunday 24th October 2021
Submission:	Source Code + Report + Demonstration (see below)

INTRODUCTION

Now that you are comfortable with the mechanics of "Modern" OpenGL, in the second half of this course we are looking at some more advanced techniques that build on this knowledge. For this assignment you will implement a simple game or "demo" that puts into practice these new techniques.

You may choose to implement either Option 1 or Option 2 described below. If you choose Option 2, your project must in some way implement all general assignment requirements and must be discussed with and approved by your lecturer by 26th September.

A ZIP file has been provided to you with a fully configured Visual Studio solution which has been set up as discussed in the lectures. It contains some configuration, utility and lighting code to help you get started, which you may choose to use as a base or ignore.

While the GLAD header file has been generated for a Compatibility Profile you may use the legacy OpenGL functions **only for the Debug parts of this assignment**.

ASSIGNMENT REQUIREMENTS

Graphics & FX (45%)

"Make It Pretty"

PA

Lighting and Materials – You may use your material and lighting model from Assignment 1 (if you implemented a full Blinn-Phong model), or the code provided as a starting point. All game objects must have materials applied to them which have Ambient, Diffuse, Specular and Shininess components. You must have one Directional light and multiple Point lights in the game. Demonstrate lights with animated positions, directions and colours.

Texture Mapping – Extend your materials and lighting model to add Diffuse Maps and Specular Maps. Demonstrate the use of both.

Sky Box – Your game world must be enclosed in a Sky Box implemented using OpenGL's Cube Map functionality.

Debug – Toggle overlay of debug text info, such as average frame rate and current game parameters; Toggle rendering of lights as objects showing position and colour.

CR

Environment Mapping – Extend your materials and lighting model to add support for a Reflection Map and/or Refraction Map, sampling the above Sky Box. Demonstrate its use on some objects.

Particle System – Implement and demonstrate a Particle system effect as taught in class (Explosion, Fire, Smoke, etc...).

Debug – Pause and fly camera (with mouse and keyboard) and reset view when done.

DI

Bloom Effect – Implement and demonstrate a Bloom effect as taught in class.

Directional Shadows – Implement and demonstrate Directional Shadow Maps as cast by the single animated directional light in your scene.

Debug – Render a line showing each object's velocity vector.

HD

Point/Omni-directional Shadows – Implement and demonstrate Point Shadow Maps as cast by at least one animated point light in your game.

Debug – Display hidden buffers; Slow Motion or Advance Frames (forward only).

Collision Detection / Spatial Data Structure (35%)

"Make It Faster"

PA

Spatial Data Structure – 2D Uniform Grid data structure for Stationary objects only. Moving objects can be compared using brute-force method.

Bounding Volumes – 2D Circles only.

Collision Detection – Circle/Line and Circle/Circle collision tests only. Must use Uniform Grid to determine which objects to test for collisions. Plus collisions and bouncing off axially aligned playfield walls.

Debug – Toggle drawing of spatial grid and bounding volumes.

CR

Spatial Data Structure – 2D Uniform Grid data structure for Stationary and Moving objects.

Bounding Volumes – 2D Circles and Axially-Aligned Boxes, as appropriate for different objects.

Collision Detection – 2D Circle/Circle and Circle/Box and Box/Box collision tests.

Debug – Toggle drawing of normals, tangents and reflection vectors at points of collision. These should be displayed for 1-2 seconds after the collision.

DI

Spatial Data Structure – Same as CR.

Bounding Volumes – 3D Spheres and Axially-Aligned Bounding Boxes, as appropriate for different objects.

Collision Detection – 3D Sphere/Sphere and Sphere/Box and Box/Box collision tests.

Debug – Extend PA and CR features to 3D.

HD

Spatial Data Structure – Implement a second type of spatial data structure in addition to the Uniform Grid. Use one for Static Objects and the other for Moving Objects. In you report discuss your implementation choices and changes in performance between the two approaches.

Collision Response – After a collision detected by DI level bounding volume, perform a second test against each edge of the underlying object and respond according to a collision with that edge.

**** If you would like to use Ray-Casting and Ray-Object intersection tests for Collision Detection / Avoidance, please speak with your lecturer about how we can change these requirements fairly and appropriately.***

Physics / Simulation (15%)

"Make It Real"

Your game model may be 2D and take place on a 2D plane, but it must be rendered in 3D.

Easy (PA)

You may use constant velocities and non-realistic acceleration and other forces for movement (whatever looks good and plays well).

Medium (CR/DI)

Realistic movement, acceleration, forces, inelastic collisions. (eg: gravity, friction, magnetic attraction/repulsion, black holes, ...). You will need to add additional attributes to your objects and/or materials to support this.

Hard (HD)

Realistic physics – elastic collisions, conservation of momentum, conservation of energy.

Report (5%)

PA – Measure and discuss performance characteristics related to the use of the Uniform Grid. Measure with and without grid. Increase number of objects in a scene to make the results clear and get a clear indication of the performance curve based on number of objects. Include text, tables and graphs as appropriate.

CR – Expand the discussion to include performance comparisons related to the new features. How does adding moving objects to the uniform grid impact performance. Does the effort involved in maintaining the structure justify the results you are seeing? Is there a point, in terms of number of objects managed, where the overhead starts to pay off by reducing the time taken for collision detection? Describe your data structure and how you kept it up to date as objects were moving between cells.

DI – Expand the discussion to include performance comparisons related to the new features. How did you test that 3D collisions are OK at different approach vectors (if your game takes place on a 2D surface)?

HD – Expand the discussion to include performance comparisons related to the new features. Create test levels that will show off the features of the spatial structures, for example levels with clustered objects vs level with uniformly distributed objects.

PHYSICS – Describe all aspects of the physics/simulation features implemented in your assignment. Discuss how you implemented flipper sensitivity and behaviour.

Demonstration

It is Mandatory that you book a demonstration time with your lecturer. The demonstration will be held on Microsoft Teams, and you will need to have a webcam and show your student card. If you fail to organise a demonstration time or don't show up to a demonstration, your final mark for this assignment will be 0.

While the demonstration does not add marks to your final result, the lecturer may remove marks or decide to not award full marks for some functionality, based on your answers to questions. You must have a thorough understanding of the techniques taught and implemented in this assignment, and you must be able to confidently navigate and modify your code if asked to.

OPTION 1 – PINBALL GAME / SIMULATION

A dodgy, poorly constructed pinball cabinet sits as the main attraction in an empty arena enclosed in a SkyBox rendered environment, lit up by a single directional light source and some point lights. The camera is placed as if the player is standing above the table looking down at the playfield, which has a dizzying arrangement of pegs and bumpers all over it. A ball sits in the bottom right ready for the action to begin...

PA

Table – Rectangular table with an inlane along the right side where the ball is launched into play (the lane can be defined by placing a long rectangular block on the table leaving a gap just wide enough for the ball to travel through, with a corner block to bounce the ball into the playfield). The table is angled so that the balls naturally fall towards the bottom, but you do not need to model realistic physics yet – just some sort of constant tendency for the ball to move downwards.

Plunger – To launch balls just press a key. No plunger animation is needed. New balls are always available and can be launched at any time. Multiple balls can be "live" on the table at the same time.

Pegs – A number of stationery cylindrical pegs are positioned at various strategic positions around the table. When balls collide with pegs they bounce off, and the pegs have lights on top which light up for a short time (point lights).

Bumpers – Active objects that apply outward force when hit (doesn't have to be realistic for PA level). Support for Circular and Rectangular shapes of different sizes.

Flippers – There are two rectangular flippers at the bottom of the table. For this PA level, they behave like bumpers, but you must take their current angle into consideration for ball bounce and velocity.

CR

Table – PA + The outer edges of the table are made up of a metallic material which reflects the environment. The incline angle of the table (which you can increase or decrease with the keyboard) now matters because we are simulating the force of Gravity.

Plunger – PA + Press the launch key and hold for power, launch on release. Show some sort of power meter and/or animate the plunger appropriately.

Balls – PA + The balls are made of a highly reflective material which reflects the environment. When balls collide with each other they cause sparks to shoot out (particle system) and light up the environment (temporary point light).

Pegs – Some of the pegs will exert an attractive or repulsive magnetic force on the balls. Differentiate these visually with different materials.

Bumpers – Add support for Line (sling-shot) and Triangular shapes. Place sling-shots on either side of the table near the flippers. Some bumpers can now move around the table in predictable patterns. Some bumpers can spin about their centre, whether they are moving or not.

DI

Directional Shadows – All objects now cast shadows from the directional light.

Ramps & Levels – Add support for ramps that lead to playfields which are at different height levels.

Flippers – Implement much more nuanced and realistic flipper behaviour.

(Optional) Gobble Holes – Add support for Gobble Holes which exert a magnetic attraction on the balls. If a ball gets stuck in a gobble hole it explodes after five seconds.

HD

Point Shadows - All point lights, including temporary ones from pegs and ball sparks, will cast omni-directional shadows.

Physics – See above Assignment Requirements section for details.

OPTION 2 – CHOOSE YOUR OWN PROJECT

Produce a Design Specification similar to Option 1 above which accounts for each of the specified assignment requirements and is of comparable complexity.

It must be a cohesive design and not just consist of random or arbitrary application of the techniques.

Must be approved before 11:59pm, Sunday 26th September.

SUBMISSION INSTRUCTIONS – Please read very carefully...

This assignment must be done individually. While discussions and idea sharing with other students are encouraged, source code should never be shared. All code must be written by yourself.

This assignment must run on Windows 10 and be built with Visual Studio 2019. It must be done in C++, and you may NOT use any third-party libraries not already included in the Visual Studio Solution that has been provided to you as a starting point. You must use the solution provided to you as a starting point unless you have been given permission not to.

Each topic/technique has been taught a certain way in class. You must implement all features of this assignment using these techniques, even where alternative methods exist.

Source Code

You must submit a single ZIP file with all your source code, Visual Studio solution files, and all dependencies necessary to build and run your code. The easiest way to do this is to create a ZIP file containing everything inside your Solution directory.

The ZIP file must be named A2_SURNAME_FIRSTNAME.ZIP.

The submission should NOT include the .vs directory or anything in the Build directories (Debug/Release/x86/x64). Delete these directories before creating your ZIP file. Check the size of your ZIP file. If it's in the hundreds of megabytes, then you've included something you shouldn't have.

MARKING GUIDE

Things You Might Lose Marks For...

Not following the above submission instructions without speaking to your lecturer about it first.
Not being able to answer questions about your approach, report answers or code during the demonstration meeting.
Your solution not building or running without modification.
Late submissions will be penalised at 10% of your result per day. If more than 5 days late, your final mark will be 0%.
If you do not book and attend a Demonstration your final mark will be 0%.