COSC1187 – Interactive 3D Graphics & Animation

Semester 1 – 2021

Assignment 1 - ASTEROID ARENA

Due Date: End of Week 7 - 11:59pm, Sunday 18th April

INTRODUCTION

This assignment is worth 50% of your final mark for this subject. You must work on this assignment on your own and submit original code you have written yourself.

You will get a score between 0 and 100 based on how many of the features you implement and the quality of your implementation. The features start off easy but quickly get more difficult in the higher levels.

We will be building a top-down 2D space shooter game loosely based on the classic arcade game Asteroids. You can see the original in action here:

https://www.youtube.com/watch?v=WYSupJ5r2zo

STORY

You are on a mission to explore an uncharted region of space when you detect an impenetrable force-field appear all around your ship.

All of a sudden, Asteroids start hurtling towards your ship from somewhere beyond the force-field. You must dodge the asteroids and shoot them to survive for as long as you can.

LEVEL 1 FEATURES (40 points)

1.1 - Screen Mode: The game must run in full-screen mode. Use the appropriate GLUT and OpenGL commands to initialise the full screen mode and display mode (RGB mode with double buffering and a depth buffer).

1.2 - The Arena: The action takes place in a rectangular arena in deep space. The background colour should be set to black.

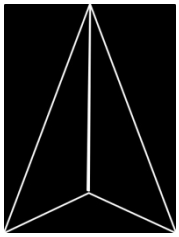
The size and dimensions of the arena, and the measurement units you use in your world coordinate system, are up to you - but you must set up your OpenGL viewing parameters so that the entire arena is visible in your viewport.

The arena and action will be on the x/y plane (x is horizontal left to right, y is vertical bottom to top). All z coordinates in this assignment will be 0. The camera is looking down the z-axis in the negative z direction.

If the viewport aspect ratio is different to your arena ratio, the arena must span the entire length of one dimension and be centred in the other dimension. Draw the arena as four separate lines. They may be any colour except red. Make the Arena Colour configurable.

Setup your viewing parameters according to the above guidelines. Make the Width and Height of the Arena configurable.

1.3 - Ready Player 1: Draw your Player Spaceship as a triangle strip of two filled triangles with an outline. Make the Player Fill Colour and Player Outline Colour configurable.



1.4 - Starting Location: The Player Starting Position should be set somewhere near the bottom-left corner of the arena, with the ship facing towards the top-right corner of the arena. The ship should be fully inside the arena.

1.5 - Player Movement: The ship movement will be controlled by pressing keys on the keyboard as follows: W - move forward, A - rotate or turn left, D - rotate/turn right. These keys should be configurable.

When the W key is pressed the ship must move forward at a constant speed until the key is released, at which point the ship must immediately stop moving. The speed should be configurable.

When the A or D keys are held down, the ship must turn either left or right. The Turning Speed should be configurable. The ship must rotate around its own centre point. Hint: Define the ship coordinates in its own local coordinate system with the origin at the centre of the ship.

It is up to you to decide what happens when the player holds down more than one of these keys at the same time. Experiment with different options and choose the one you like best.

HINT: You will need to create a Ship class or struct which tracks the current ship position, direction and other state variables. You should also create a class or struct for managing keyboard and mouse states. Your physics and rendering code should be able to tell what the current movement and firing states are at all times.

1.6 - Hit The Wall: If the Player ship gets close to any of the arena walls, warn the player by making that line segment turn Red. Make the Warning Distance configurable. If the Ship collides with a wall it is game over and you must reset the game back to the starting

conditions. If the ship moves back to a safe distance, change the arena wall colour back to its old value.

HINT: Use a Bounding Circle around the ship when checking for collisions with the wall. You should be keeping track of the ship position at all times. This point should coincide with the centre of the ship. Use the radius of your bounding circle to determine if the ship collides with a wall.

1.7 - Math Structs/Classes and Functions: Define and use your own Point and Vector classes or structs and use them throughout your code. Also create functions for operations on points and vectors. Do this throughout the assignment whenever something is related to math or physics.

1.8 - Code Quality: Your code must be well structured. Define appropriate classes or structs to represent game objects such as the player Ship, Asteroid, Wall, Configuration, etc... Use meaningful variable names. Be consistent with your conventions, such code indentation and letter case. Use multiple source files to separate related functions and classes. Don't create functions or methods which are overly long - break them up into smaller more focused units. Comment your code when something is not obvious or when you want to draw attention to something.

If using C++, do not go overboard with your design. Stick to the basics wherever possible. Don't implement any elaborate design patterns - this subject is about Interactive 3D Graphics, not OO Design.

LEVEL 2 FEATURES (20 points)

You must implement all LEVEL 1 features before moving on to this level.

2.1 - Launch Position: An asteroid is to be launched into the arena from outside (therefore off-screen too). Calculate a radius from the centre of the arena to define a circle that encircles the entire arena and is completely "off-screen". Using a random number generator, determine a position along this circle that will be used to launch an asteroid from.

2.2 - Asteroid Launch: Create an instance of your Asteroid class or struct, and "launch" the asteroid, from the launching point calculated above, in a straight line towards the current Player Position. Calculate a random speed for the asteroid which is between some configurable min/max speed range. Draw the asteroid as a Circle using an appropriate OpenGL primitive and a configurable Asteroid Colour. Make sure to use double-buffering so there is no flickering of the display. The asteroid does not collide with the arena boundaries but passes right through.

2.3 - Asteroid / Ship Collision: The player must use the ship keyboard controls to move the ship to avoid colliding with the asteroid. Using the current ship and asteroid positions, and their respective bounding circles, check for collisions. If the ship collides with the asteroid the game is over and resets to the starting condition.

For the next section - Asteroid attacks will come in waves. At the start of the level only one asteroid is launched at you. After some time, two asteroids will be launched, then three, then four, etc ... each from a random location on the launch circle, and towards the current player position. The time between waves should be configurable. Use your judgement to determine what a good set of parameters is for best gameplay.

2.4 - Multiple Asteroids: Implement waves of asteroids getting launched as described above. All of them start from a random location and are launched towards the current player position at random speeds between some configurable min/max range. It is up to you to decide if all asteroids start off the same size (which is configurable) or some random radius between some configurable range.

HINT: You must have a data structure (array or list) to keep track of the current position of all asteroids which are currently active, and an Asteroid class or struct to keep track of things like size, position, direction, speed for each asteroid.

In this level of the assignment, asteroids do not collide with each other and they can leave the arena. Once they are off-screen it is not necessary to keep tracking their position, so you may stop tracking their location.

2.5 - Particle Puff: Add a very simple particle system to represent some sort of exhaust coming from the back of the Player ship, when it is moving. There should be no exhaust when the ship is not moving (except the last few particles immediately after movement stops). The exhaust should be made up of a set of points which change size over time. They start off at the largest size and get smaller based on some function of time, until they disappear. They get "dropped" from the current position of the Player ship (offset a little to the rear from the centre, along the ship direction vector, to make it look like it's coming from the back of the ship). If the ship is turning, the stream of exhaust particles will follow a curved path. The number of particles, time between "dropping" each particle, the starting size, the time to "decay" should all be configurable parameters. Experiment with different values to get the exhaust particle system that looks best to you. You might even want to experiment with having the colour change over time for some effect.

LEVEL 3 FEATURES (20 points)

You must implement all LEVEL 2 features before moving on to this level.

3.1 - Procedurally Generated Asteroids: Instead of drawing every asteroid as a circle, add some variety to the shapes of each asteroid. Use some randomness to make them look more interesting and unique. Vary the number of points and the angles between each line segment. Still keep the shape roughly circular, as you will continue to use a bounding circle for collision detection.

3.2 - Rotating Asteroids: Asteroids need to rotate about their centre as they are moving. The rotation direction needs to be randomly selected to be clockwise or counter-clockwise when the asteroid is launched, and maintained for its entire lifetime. The rotation speed should be some random amount between a configurable min/max range.

3.3 - Shooting: Holding down the left mouse button will make the player ship shoot a stream of bullets, one at a time at a certain configurable Firing Rate. Bullets start at the front tip of the ship and are launched in the current ship movement direction. They travel at a constant (configurable) speed in this direction until they either hit an Asteroid or one of the arena walls. Draw the bullets as GL_POINTS of an appropriate size and colour (both configurable). You must keep track of all active bullets in some sort of list or array data structure. Collisions are tested against each asteroid's bounding circle and the arena walls. If any bullet hits an asteroid or wall, both the bullet and asteroid are "dead" and should no longer be drawn.

HINT: Don't set your firing rate to be too high (other than for a laugh) as it will lead to lots of bullets being active at the same time. This might make the gameplay too easy. Also look out for overflowing your bullet array and causing a game crash. If using a fixed size array for bullets, you will need to put in place something to stop you from having too many bullets on-screen at the same time.

3.4 - Hit Points: Each Asteroid should require more than a single hit before being killed. This should be a function of its radius - larger asteroids should require more bullet hits than smaller asteroids to be destroyed. Use your judgement for what a "good" number of hit points is.

3.5 - Time and Score: Keep track of the elapsed time in minutes and seconds and the number of Asteroids destroyed. Print one of these "scores" in the top-left and the other in the top-right of screen. These should be set to zero every time the game is re-started.

HINT: You will need to implement a function to calculate text width so that you can correctly position the score on the right. After drawing your game frame, you will need to change your orthographic projection settings to match the display pixel resolution, render the text using a GLUT bitmap or stroke font, and then reset your projection matrix back to its previous settings. You may also want to disable and then re-enable the OpenGL depth test during this phase.

3.6 - Game Over, Man: When the player ship is destroyed, instead of immediately restarting the game, display a message saying "Game Over. Press any key to play again..." and wait for the user to press a key before restarting the game. Keep all asteroids moving during this time. Likewise, the first time the game is started (only), have a message saying, "Press any key to start...".

LEVEL 4 FEATURES (20 points)

You must implement all LEVEL 3 features before moving on to this level.

4.1 - Bouncies: Make the asteroids bounce off the arena walls instead of passing through. They will maintain the same velocity and rotation direction and speed throughout. The angle that they bounce off at should calculated accurately based on the angle of approach and the normal vector of the wall.

4.2 - Bouncies 2: Make the asteroids bounce off each other instead of passing through. They will maintain the same velocity and rotation direction and speed throughout.

4.3 - Do The Splits: When an Asteroid's Hit Points are exhausted, instead of it being destroyed it will split into two new asteroids. Implement this by replacing the original asteroid with two new asteroids which have half the radius of the original. The starting positions of the new asteroids must be calculated by translating them left or right from the original asteroid's last position, along a vector perpendicular to the original asteroid's direction vector, and by an amount which ensures that they do not overlap. The new direction of the asteroids will be at 45 degrees to the right or left of the original asteroid's direction vector. Hit Points of the new asteroids should be calculated using the same formula as all other asteroids, and they should start off with full hits points. Asteroids can only be split once.

HINT: Make sure the two new asteroids are not overlapping when they're created, as that might confuse your collision detection code.

4.4 - Earth Shattering Kaboom: When an asteroid is destroyed, instead of just having it disappear, replace it with a short explosion particle effect. The particle system should start with a configurable number of particles at the asteroid's last position. They should then 'launch' into different random directions at different speeds and decay in size and/or colour until they disappear. This should all take some configurable amount of time.

BONUS FEATURES

You can implement these features at any stage for extra marks, but it is not possible to score more than 100 for this assignment.

B.1 - Better Ship Movement (5 points): Add two extra states to the player ship's movement to make it a little more realistic. Instead of instantly moving at full speed from rest when the W key is pressed, have it enter a "speeding up" state during which the velocity is increased linearly or based on some curve. Likewise, instead of instantly stopping when the key is released, have the ship enter a "slowing down" state during which the velocity is slowed down. Make the rates and time taken configurable and experiment with different values until you find a configuration that feels best to you.

B.2 - Black Hole (10 points): Add a "black hole" at some random position of the arena each time the game is re-started.

The black hole should have the effect of "pulling" all game entities (player ship, asteroids, bullets) in the direction of the centre of the black hole. Use a simple formula based on the square root of the distance from the black hole to the object you're testing against to determine how much that object will change its trajectory due to the black hole.

If bullets hit the black hole they should disappear. If an asteroids crashes into the black hole, it should disappear or explode (depending on which level you have implemented up to). If the player ship collides into the black hole it is instantly the end of the game. Do not worry about physics or event horizons - implement what looks and plays nicely to you.

Draw the black hole as a circle which "pulses inward" on some frequency, ie: It start off at its full size, and reduce its radius each frame based on elapsed time, resetting to its full size after every one or two seconds. Make colour and pulse rate configurable.

GENERAL COMMENTS

Scale and Units

Track all distances and rates in world coordinates/units rather than pixels. You should have complete separation between your model data structures/physics and your viewport. It is up to you to decide on the scale and units of your universe.

Physics

You are not required to use accurate physics calculations in this assignment for anything except for calculating bounce angles (on asteroid/wall and asteroid/asteroid collisions). You do not need to model mass or acceleration or conservation of energy for anything.

Configurable Parameters

Where you are asked to make something configurable, it means that the values for that feature should be stored as variables in some sort of configuration structure or class in your code, instead of hard coded everywhere these values are used. It must be easy to change these values in one place in your code and have the new values used where needed.

For example, the game arena must be of some size. Store the arena width and height as configuration variables. If we want to change the arena size in the future, we should only have to change the values in one place.

Experiment with different sized arenas. The larger the arena (in world coordinates) the smaller everything will be drawn on the screen, giving you more room for manoeuvrability. Find the configuration that makes for the most enjoyable game for you.

Another example might be Asteroid Colour. Store the RGB value in a configuration or template structure somewhere. If we want to change the colour of all Asteroids, we should only have to change it in one place.

A third example might be the Turning Speed of your ship. Put it in a configuration structure or ship template or default class variable somewhere and use that value where needed.

You should easily be able to change these values, recompile your code, and see the effect the change will have on the gameplay. This will be very useful for balancing your gameplay. You do not need to have the ability to change these values at run-time based on user inputs or some sort of console or configuration file - constant values defined in appropriate structs or classes are enough.

Submission

Submissions will be via Canvas.

You will need to submit a single ZIP file containing all of your code and a README.TXT file telling me which features you have implemented, how to build your code, and any other things I should be aware of.

If submitting on Windows, please include your Visual Studio solution and all dependencies I would need to run your code. Use the Visual Studio solution I have already provided you as a starting point. If you have changed any compiler options, please highlight them in the README.TXT file and let me know why you made those changes.

If submitting on macOS or Linux, please include a shell script file named build.sh which will execute the appropriate command(s) to build your code for me.

Late submissions will receive a 10% per day penalty.