COSC1187 – Interactive 3D Graphics & Animation

Semester 1 – 2022

Assignment 1 - ASTEROID ARENA

Due Date: End of Week 7 - 11:59pm, Sunday 17th 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

This assignment gives you creative control for deciding the look/feel of the game such as rendering spaceships/game elements and color picking. Keep the 2D geometry for your game elements simple, preferably less than 8 triangles (except the asteroids). Colors must be different for the game elements. For example, don't use the same color for the spaceship as you would for the arena or asteroids etc.. Different instances of the same element can have the same color, for example the asteroids, bullets etc...

HINT: (optional) If having trouble deciding on colors, use online tools to generate color palettes to use for your various game elements such as arena color, spaceship color, bullets, missiles etc..
HINT: You will NOT be penalised if your game elements contain geometry > 8 triangles, but keep things as simple as possible!

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 (5pt): 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 (5pt): The action takes place in a rectangular arena in deep space. Pick an appropriate background color of your liking.

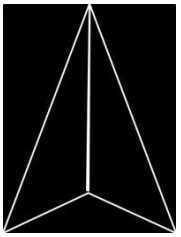
The size and dimensions of the arena, and the measurement units you use in your world coordinate system, are up to you - but you must set up your OpenGL viewing parameters so that the entire arena is visible in your viewport.

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.

Setup your viewing parameters according to the above guidelines.

1.3 - Ready Player 1 (5pt): Draw your Player Spaceship with an outline using a drawing primitive of your choosing (GL_TRIANGLES, GL_GEOMETRY) etc... You have creative flexibility to choose the spaceship color and shape. When rendering your spaceship, the triangles must NOT overlap. The spaceship must have a fill and an outline color of different values. See example of a spaceship made from 2 triangles, your design can be different.



1.4 - Starting Location (5pt): The Player Starting Position should be set at the center of the arena whilst facing upwards. The ship should be fully inside the arena.

1.5 - Player Movement (5pt): 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.

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 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 (5pt): If the Player ship gets close to any of the arena walls, warn the player by making that line segment turn a preferred shade of red color. 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 - Shooting (5pt): Holding down the left mouse button will make the player ship shoot a stream of bullets, one at a time at a Firing Rate. Decide on a rate so as to make the game enjoyable and fair. Bullets start at the front of the ship and are launched in the current ship movement direction. They travel at a constant speed in this direction. Draw the bullets as GL_POINTS of an appropriate size and colour.

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.

1.8 - Math Structs/Classes and Functions (2pt): 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.

1.9 - Code Quality (3pt): Your code must be well structured. Define appropriate classes or structs to represent game objects such as the player Ship, Asteroid, Wall, 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 (2pt): 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 (3pt): 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 min/max speed range of your choosing. Draw the asteroid as a Circle using an appropriate OpenGL primitive and a colour of your choosing. 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. The asteroid must have an outline color of your choosing. Fill color is optional.

2.3 - Gun Rendering & Shooting (5pt): Draw your Gun on your player Spaceship with an outline using a drawing primitive of your choosing (GL_TRIANGLES, GL_GEOMETRY) etc... You have creative flexibility to choose the gun color and shape. When rendering your gun, the triangles must NOT overlap. The gun must have a fill and an outline color of different values and be smaller than your spaceship. The gun must be rendered on your spaceship and some overlap is acceptable. Change the shooting mechanic so bullets are shot from the tip of the gun.

2.4 - Asteroid / Ship / Bullet Collision (5pt): 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. Bullet collisions are tested against each asteroid's bounding circle and the arena walls. If any bullet hits an asteroid or wall, the bullet should be destroyed along with the asteroid.

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.

2.5 - Multiple Asteroids (5pt): 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 min/max range. Asteroids should start with some random size between some min/max values (you decide).

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.

LEVEL 3 FEATURES (20 points)

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

3.1 - Procedurally Generated Asteroids (5pt): 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 (1pt): 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 - Gun Switching (7pt): Add a missile launcher as a second gun toggleable by pressing a key on the keyboard as follows: Z - toggle between original gun and a missile launcher.

Create a new missile launcher with a different design of the original gun. When guns are switched, the graphics should switch to show the difference. The rendering requirements are the same as that of the gun.

The missile launcher shoots a seeking missile that constantly tracks and hits the nearest asteroid. If the asteroid is destroyed, the missile should seek the next closest target from its current position.

The missile should always face the target asteroid. Decide on an appropriate speed and rate of fire.

3.4 - Hit Points (2pt): 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. Missiles/Guns should deal different damage. Change the asteroid color to reflect the current state of the asteroid health (outline or fill or both).

3.5 - Time and Score (2pt): 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 the 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 (3pt): 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 (5pt): 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 be calculated accurately based on the angle of approach and the normal vector of the wall.

4.2 - Bouncies 2 (5pt): 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 (2pt): 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 (8pt): When an asteroid is destroyed, instead of just having it disappear, replace it with a short explosion particle effect. Decide on the number of particles to maintain a good performance and visual quality ratio. They should then 'launch' into different random directions at different speeds and decay in size and/or colour until they disappear.

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 - Spawn Drone (10 points): Add a mode using the keyboard X button that spawns a stationary drone at current spaceship position that automatically shoots the standard gun (same rate of fire) at nearby asteroids.

The drone should only last a limited amount of time and only 2 drones can be active at any time. Spawning the drone should cost some points (deducted from score) which are collected by destroying the asteroids.

You have full creative control on how to render the drones. Assume that they ignore collisions between player bullets, other drone bullets and the player itself.

If an asteroid collides with the drone, it should be destroyed.

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.

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.

In your README.TXT make note of the features that were attempted but not successfully implemented. Unsuccessfully implemented features/code can be submitted as commented out code, make a note of the file/code line number/function name. Based on the effort, partial marks up to 50% can still be earned.

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.