Assignment 1 3D Interface Technologies

# Student: Jahziel Angelo Belmonte

## Student number: 21335346

# Section 1: Scene Graph

A diagram of a space fighter

AI-generated content may be incorrect.

Figure 1 – Full Scene Graph

# Section 2: Introduction

This X3DOM Demo “Space Fighters”, aims to demonstrate the capabilities of X3DOM as well as its limitations. The concept of this demo is a ship in space, where the user can navigate using arrow keys, and mouse pointer. There are planets and star objects the user can interact with as well as a blaster that the user can operate when pressing ‘f’. The user has a dedicated score, health and ammo system which is seen over the x3d scene. Camera view is modified from ‘third person’ to ‘first person’ by pressing ‘v’ .

## Initialization

The scene is first initialized with its objects, “ship”, “planets” and “ammo Boxes” . These objects attached to the root scene and are updated within the game loop. All three objects are added to the scene graph within the DOM heading.

### Ship object

A diagram of a ship object

AI-generated content may be incorrect.

Figure 2 – Ship object Graph

The ship object is comprised of custom geometry, material attributes, transforms, group nodes and lighting. The first three transforms, “Roll, Pitch, and Yaw” controls the ship object. The translation transform maintains the ships origin position located at 0,0,0. The ship composes of 3 components its body, accelerators, and barrel.

The main body is composed of triangle vertices. The colour, glow and points are stored in an array ‘shapes’ and passed into createShape() which creates shape nodes of geometry 2DTriangleSet. The triangles are determined from the 3 vertices points in space. It was important to ensure that the triangles were front facing (Anti-clockwise) as back face culling was enabled. The function createShape() is called for each element in the shapes array.



Figure 3 – Custom geometry for the ship body

The accelerators are created by X3D’s SpatialGeometryNodes ‘cone’ and the ships barrel with ‘cylinder’. The barrel has a child node PointLight which provides lighting to the scene from the specified point spreading outwards with a radius attribute. Combining all elements together to the transform object created the ship object. 3 Transform nodes are parents of the ship object providing movement in 3D Space.

A screenshot of a video game

AI-generated content may be incorrect.

Figure 4- Ship object created from 2D Triangles and GeometryNodes. Shining pointlight on object.

### Planet objects

A diagram of different types of objects

AI-generated content may be incorrect.

Figure 5 – Scene graph for planet objects. Simplified to Sun and Saturn as rest of objects share semantics.

The planet objects are created to the Geometry Nodes ‘sphere’ Nodes with two notable exceptions, “Sun” and “Saturn.” All nodes are collectively children of the “planetGroup” Group node for intractability. Each planet’s appearance contains material and ImageTexture nodes to simulate realistic textures as well as lighting collisions.

The sun object contains boilerplate node creation with a transform node for rotation. Using the Point Light node element, I was able to simulate realistic sunlight emission in my space environment.

A red and orange sun with white text

AI-generated content may be incorrect.

Figure 6 – Sun object with texture mapping. Point light can be seen on the surface of mars object.



Figure 7 – Planet boilerplate code and lighting for sun

The ring object is appended to the scene but is a property of planetGroup. Variables to create the ring objects were needed such as number of points, inner and outer radii and anglestep which is the angular spacing for each point. The math would provide triangles forming a circle as it would draw on the circumference of the rings. The function would calculate the inner and outer coordinates of the points and push them into an array to be passed into createShape function. Finally, a pointlight element was appended to the ring transform. I added an animation to rotate the rings via the rings transform element accessing the rotation property, rotating around the y axis by angle \* pi / 180.



Figure 8 – Ring transform initialization and rotation animation

Finally, when the document was finished loading, function call to animatePlanets() would play which would rotate the planet transforms via requestAnimationFrame.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9 – Saturn and ring transforms. Light collisions and shadows are shown on the planet’s surface as well as the ring geometry around the planet.

### AmmoBoxes

The ammoBoxes or Star models are used for the user to collect, refill ammo, and gain points. I used an external model from free3d. [1] which provides open sourced models to use. I selected a star object and added it to my scene creating an Inline element. The attributes URL, scale and rotation were set and was appended to the transform which was appended to ammoGroup element. To initialize multiple star objects, I used a for loop. To generate 350 star objects, each with a randomized coordinate contained within the boundaries [ [-2000,2000], [-2000, 2000], [-4000,1000] ]. Animation functionality is added by indexing for each ammoTransform element and using an incrementing angle on the rotation attribute.

A screenshot of a video game

AI-generated content may be incorrect.

Figure 10 – ammoGroup objects rotating via requestAnimationFrame()

### Main

This script contains majority of the game logic including, camera and ship movement, audio events, keydown events, projectile interactions and collision.

#### Movement

Movement consists of mouse pointer position on overlay and keyevents ‘w’ and ‘s’. To calculate the movement of the camera and ship, directional vectors are used by getting x,y,z positions determinant on yaw and pitch values. This is then normalized by getting the magnitude. To track the key events I used an object initializer to store pressed keys. The canvas is used in calculating the mouse position on the x3d canvas.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 11 – MouseTracking Overlay defined by the white borders. Allows to listen for mouse events within the overlay.

On a mousemove event, the pointer position is calculated by calculating the offset of the mousepointer relative to the centre of the element. By normalizing the offset the values are clipped to a range [-1,1]. To prevent flipping the ship by exceeding values of pitch, a limit was implemented by setting a maximum angle and getting the maximum angle relative to -maxpitch, and min(pitch,maxpitch). These were displayed via html element’s id “yaw, pitch and roll” that could be modified using element.textContent.

When the mouse is captured within the overlay border, the ship\_orientation() is called which modifies the ships orientation attribute based on yaw \* sensitivity, pitch \* sensitivity. The roll is affected by the change in yaw. The transforms, ‘yawTransform’,

‘pitchTransform’, ‘rollTransform’ are changed .



Figure 12 – Logic for Mouselistener event and rotation of ship.

The camera logic involves adjusting the viewpoint element with the shipPos and cameraOffset. There are two views available for the user to use, ‘Third person’ enabled by default and ‘First person’ toggled by pressing ‘v.’

##### First person

The camera calculation involves using z-translation by using a forwardVector [ 0, 0 ,1 ] \* Firstperson camera distance + the ships original co-ordinates to find the camera position. The ship is then updated with yaw, pitch and roll by updateShipPosition()

##### Third person

Since the camera position is offset from the ship by y and z co-ordinate, a translation vector is needed to calculate where:

Equations for Z-direction vector

These vectors were normalized and used for updating shipPos values. I implemented an acceleration and deceleration system using speed, acceleration and max values. If user held down ‘w’ or ‘s’ the corresponding movement logic would be handled. The ship position would be handled by setting the translation of the shipTransform element to shipPos values.



Figure 13 – Camera and Ship position logic