Graphics 2 Report: 3D Submarine Simulation

Theofanis Chatzidimitriou - 100263854

December 2019

1 User Controls and Movement

The submarines is being controlled by the conventional keys, 'W' for moving forward and 'S' for backwards, 'A' for turning left and 'D' for right. It can tilt upwards with the 'SPACE' key and downwards with 'CTRL'. When turning left or right the submarine rotates around is Y-axis and when tilting up or down it rotates around it's Z-axis. While moves it can accelerate it's speed till a certain speed limit and it resets when stopping. In addition the user can use different camera views by using the keys '1', '2','3' and '4'.

2 Camera Views

The simulation has over four different camera views that show the submarine in different angles and distances. The first one is the main view, the submarine can be seen any angle by freely moving around, above or underneath it and with the mouse-wheel you can zoom in or out. This is being done by focusing the camera on the submarine and changing the pitch and yaw angles. The second one is a top view where the submarine can be viewed from above. The third view moves the camera right above the submarine viewing both the craft and the front of it, giving a first-person view sense. The last view is showing the submarine from below.

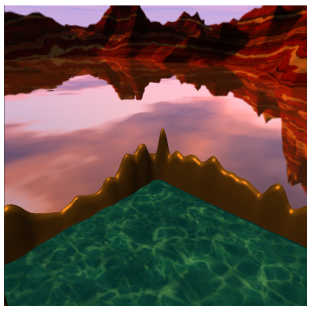
3 Graphics and Craft

3.1 Craft

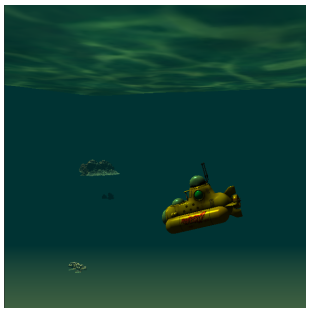
All the objects responsible for collision detected have been constructed and rendered in OpenGL, the rest were imported from 3DS Max. The ocean consists of two planes, one plane is the seabed which is cut in two in order for the fog to be applied at it's bottom . The other plane is textured with a wave image to represent the top of the ocean. The skybox has been also created with 3DS Max, it's essentially a cube with its normals facing inside, giving the ability to texture the inside of it.All the other objects as well as the submarine, have been found online as .obj files.

3.2 Graphics

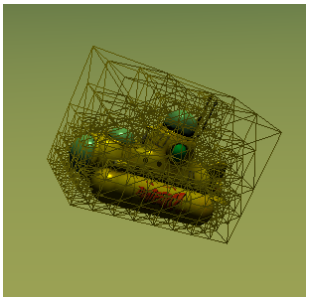
There was a main lightning source above the water that illuminates all the objects. No specular lightning has been applied on the objects under water, mainly diffuse ,thus they would not shine under water. The fog that was applied has a blue-green color for the water to be more realistic.



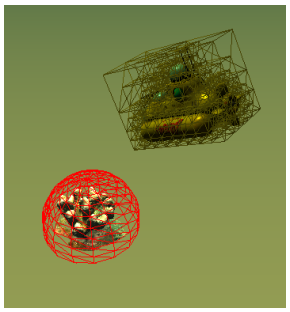
(a) Top-view scene



(b) Under water scene



(c) Bounding box around the submarine



(d) Sphere around an object

Figure 1: Graphical environment and collision detection objects

4 Collision Detection and Response

4.1 Collision Detection

The collision that was used for the simulation is Sphere Collision and Separate Axis Theorem with Bounding Boxes. The collision for the bottom and top of the ocean is using just an if-statement that checks the submarine's y-coordinates with theirs. The side walls of the ocean as well as the submarine use bounding boxes, where the other objects in the scene are using sphere collision, as with their shape it was much easier and efficient.
See algorithm 1 for the Separate Axis Collision Detection, see algorithm 2 for the Sphere collision detection with a bounding box.

4.2 Collision Response

The collision response when colliding with the walls is that it prevents the submarine moving any further or deeper by setting the submarine's position with the one just before the collision. When there is collision with the top or bottom of the ocean, the submarine's y-position is being adjusted thus it would not move below or above the ocean. When colliding with sphere objects, an up- vector is added to its current position and the submarine moves around the object's sphere.

5 Usability of the system

The use of the program is fairly easy, there are simple keys for the movements and for changing the camera views of the submarine. The graphical environment gives a pleasant feeling when using is and since the ocean objects are actually scans from "real-word" ocean, it lets the user see and learn what can be found in an actual ocean. The program has been tested and it is running smoothly with no crashes

Algorithm 1 isIntersectingOBBCD(aMinMax,bMinMax,Axis)

Require: Two arrays of vectors $aMinMax$ and $bMiMax$ containing the 8 combinations of the min and max values of the objects A and B and a vector $axis$ with the axis to be checked.

Ensure: It will return false if the two objects overlap or intersect each other. max() returns the maximum number, min() returns the minimum of the given numbers.

if $axis == (0, 0, 0)$ **then** ▷ checks if the axis is in parallel with the object
  return $false$
$aMin = bMin = \infty$
$aMax = bMax = -\infty$
for $i = 1$ to 8 **do**
  $aDist = aMinMax_i \times axis$ ▷ dot product the two vector
  $bDist = bMinMax_i \times axis$
  if $aDist < aMin$ **then**
    $aMin = aDist$
  if $aDist > aMax$ **then**
    $aMax = aDist$
  if $bDist < bMin$ **then**
    $bMin = aDist$
  if $bDist > bMax$ **then**
    $bMax = bDist$
$longSpan = max(aMax, bMax) - min(aMin, bMin)$
$sumSpan = aMax - aMin + bMax - bMiN$
if $longSpan \leq sumSpan$ **then**
  return $false$
else return $true$

Algorithm 2 isIntersectingSphereCD(box,sphere)

Require: A Bounding Box box and a Sphere $sphere$ containing their coordinates and values

Ensure: It will return true if the two objects are intersecting each other. max() returns the maximum number, min() returns the minimum of the given numbers. bx, by, bz are the boxes coordinates and $minX, minY, minZ, maxX, maxY, maxZ$ are the box's minimum and maximum values. $sx, sy, sz, radius$ are the spheres coordinates radius respectively.

$x = \max(minX + bx\ ,\ \min(sx,\ maxX + bx))$
$y = \max(minY + by\ ,\ \min(sy,\ maxY + by))$
$z = \max(minZ + bz\ ,\ \min(sz,\ maxZ + bz))$
$distance = \sqrt{2(x - sx) + 2(y - sy) + 2(z - sz)}$
if $distance \leq radius$ **then**
  return $true$
else
  return $false$