PROJECT REPORT

OpenGL Program to draw an interactive fish aquarium
With keyboard events

SUBMITTED BY

MD. MAMUN

ID: 201-15-3523

Section: PC-H

Course Code: CSE421

SUBMITTED TO

MD. MUBTASIM FUAD

LECTURER, DEPARTMENT OF CSE

Project Name

OpenGL Program to draw an Interactive Aquarium

Project Description

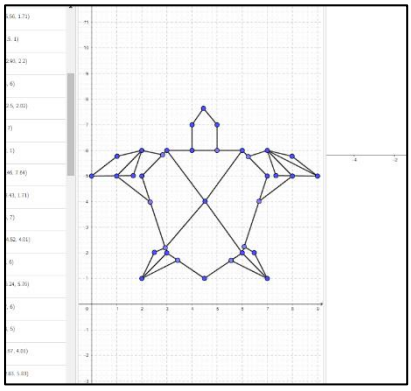
The Aquatic World Simulation is an OpenGL program that simulates an underwater environment with various aquatic creatures and objects. The program utilizes OpenGL for rendering and GLUT for window management. The simulation includes animated fish, turtles, and jellyfish, as well as bubbles and a dynamically changing background.

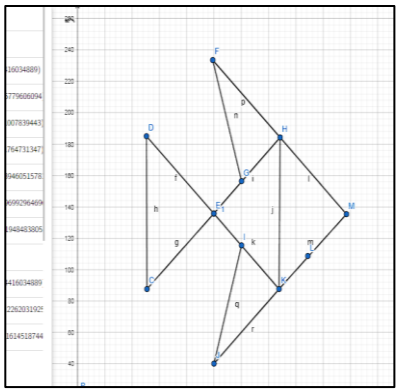
Contents

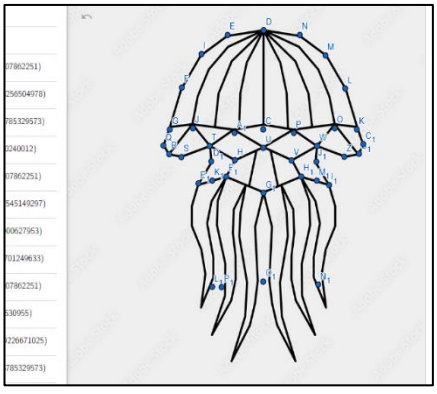
- Fish: Animated fish swimming across the screen.
- Turtles: Animated turtles with a detailed shell.
- Jellyfish: Animated jellyfish with realistic tentacles.
- Bubbles: Floating bubbles that rise to the surface.
- Background: A dynamically changing background to create an underwater atmosphere.

Initial Sketch

The initial sketch for the aquatic world simulation includes a rectangular window with dimensions 1920x1080 pixels. The window will serve as the canvas for rendering the underwater environment. The main components of the simulation, such as fish, turtles, jellyfish, bubbles, and the background, will be animated and rendered within this window.







Key Notes

Variables:

- windowWidth and windowHeight: Variables to store the dimensions of the window.
- numBubbles: Variable to store the number of bubbles in the simulation.
- bubbleRadius: Variable to store the radius of each bubble.
- bubbleSpeed: Variable to store the speed at which the bubbles rise.
- bubblePositionX and bubblePositionY: Arrays to store the positions of the bubbles.
- fishPosX and fishPosX2: Variables to store the initial X positions of the fish.
- jellyPosY: Variable to store the Y position of the jellyfish.
- x_cor and y_cor: Variables to store the X and Y positions for drawing objects.
- jellyCounter: Variable to keep track of the jellyfish animation.
- y_text: Variable to store the Y position for displaying text.
- newFishPosX2: Variable to store the updated X position of the second fish.
- motorStarted: Boolean variable to indicate whether the motor is started or not.

Functions:

- displayText(float x, float y, int r, int g, int b, const char* string): Function to display text on the window.
- handleKeyPress(unsigned char key, int x, int y): Function to handle keypress events, specifically the 't' key to toggle the motor state.
- turtle(float size, float xOffset): Function to draw a turtle shape with the specified size and X offset.
- jellyfish(): Function to draw a jellyfish shape with animation.
- fish1() : draws the fish shape
- fish2() : draws the fish shape
- fish3() : draws the fish shape
- fish3rev() : draws the fish shape in reverse order
- fish4(): draws the fish shape
- display() : function to initialize glut project display.
- drawBubbles(): draws the bubbles on the screen.

- updateBubbles(): Updates the bubble coordinates and shapes
- drawAquarium(): Draws the aquarium with water color and border for the aquarium in left bottom and right edge.
- drawGrass(): Draws the grass inside the aquarium.
- update(): function responsible for animating objects
- drawlight(): function responsible for motor indicator

Reason for Using Libraries

- OpenGL: OpenGL (Open Graphics Library) is a widely used graphics API that provides a set of functions for rendering 2D and 3D graphics. It is used in this project for rendering and animating the aquatic creatures and objects.
- GLUT (OpenGL Utility Toolkit): GLUT is a library that provides functions for creating and managing windows, handling input events, and other utility functions. It simplifies the process of creating a graphical user interface using OpenGL.

Drawing Shapes

- Circles: In this project, circles are drawn using the OpenGL primitive glBegin(GL_TRIANGLE_FAN) along with a loop that calculates the vertex positions of the circle. The glVertex2f function is used to specify the X and Y coordinates of each vertex, forming a complete circle shape.
- Polygons: Polygons are drawn using the glBegin(GL_POLYGON) primitive. Similar to circles, the glVertex2f function is used to specify the X and Y coordinates of each vertex of the polygon, creating the desired shape.
- Triangles: Triangles are drawn using the glBegin(GL_TRIANGLES) primitive. Three vertices are specified using glVertex2f, and OpenGL connects these vertices to form a filled triangle.
- Quads: Quads (four-sided polygons) are drawn using the glBegin(GL_QUADS) primitive. Four vertices are specified using glVertex2f, and OpenGL connects these vertices to form a filled quadrilateral shape.

How the animation works

Fish Animation: The fish animation involves repeatedly rendering fish shapes at different X positions to give the illusion of movement. The fish are drawn using a combination of polygons and triangles to create a streamlined fish shape. By incrementing the X position in each frame, the fish appear to swim across the screen. Additional transformations such as rotation and scaling can be applied to enhance the animation.

Turtle Animation:
The turtle animation utilizes the turtle function, which draws a turtle shape with a detailed shell. The shell is created using polygons and triangles to form the desired shape and is scaled based on the provided size parameter. The turtle function also uses an X offset parameter to position the turtle correctly within the simulation. By repeatedly calling this function with different parameters, the animation of multiple turtles can be achieved.

Jellyfish Animation: The jellyfish animation is implemented using the jellyfish function, which draws a jellyfish shape with animated tentacles. The function uses scaling factors to adjust the size of the jellyfish and a Y position variable to control its vertical movement. The tentacles are created using polygons and triangles, and they move up and down in a sinusoidal motion to mimic the jellyfish's floating movement. By repeatedly calling this function with appropriate parameters, the animation of multiple jellyfish can be achieved.

Bubble Animation: The bubble animation involves creating and rendering multiple bubbles that rise to the surface. The bubbles are represented as circles and are drawn using the glBegin(GL_TRIANGLE_FAN) primitive. The positions of the bubbles are updated in each frame, gradually increasing their Y coordinates to simulate their upward movement. By specifying the number of bubbles, their radius, and speed, a realistic bubble effect can be achieved.

Project Source Code

```
#include <GL/gl.h>
#include <GL/glut.h>
#include <cstdlib>
#include<string.h>
#include <ctime>
#include <cmath>
#include <iostream>
int windowWidth = 1920;
int windowHeight = 1080;
int numBubbles = 20;
float bubbleRadius = 20.0;
float bubbleSpeed = 1.0;
float bubblePositionX[100];
float bubblePositionY[100];
float fishPosX = 50.0;
float fishPosX2 = windowWidth-50;
float jellyPosY = 100;
float x_cor = 1080;
int jellyCounter = 0;
float y_cor = 50;
float y_text = 1050;
float newFishPosX2 = 50;
bool motorStarted = false;

void displayText(float x, float y, int r, int g, int b, const char* string) {
    int j = strlen(string);
    glColor3f(r, g, b);
    glRasterPos2f(x, y);
    for (int i = 0; i < j; i++) {
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, string[i]);
    }
}
void handleKeyPress(unsigned char key, int x, int y) {
    if (key == 't') {
        motorStarted = !motorStarted;

        if (motorStarted) {
            displayText(windowWidth / 2.5, windowHeight - 100, 1, 1, 1, "Motor Started!");
        } else {
            displayText(windowWidth / 2.5, windowHeight - 100, 1, 1, 1, "Oxygen Level is low
- T to start the motor");
        }

        glutPostRedisplay();
    }
```

```
}
void turtle(float size, float xOffset) {
    glColor3f(0.2f, 0.8f, 0.6f);
    glBegin(GL_POLYGON);
        glVertex3f(4.0f * size + xOffset, 7.0f * size + y_cor, 0.0f);
        glVertex3f(4.45833f * size + xOffset, 7.64208f * size + y_cor, 0.0f);
        glVertex3f(5.0f * size + xOffset, 7.0f * size + y_cor, 0.0f);
        glVertex3f(5.0f * size + xOffset, 6.0f * size + y_cor, 0.0f);
        glVertex3f(4.0f * size + xOffset, 6.0f * size + y_cor, 0.0f);
    glEnd();

    /* Triangles */
    glColor3f(0.8f, 0.2f, 0.2f);
    glBegin(GL_TRIANGLES);
        glVertex3f(3.0f * size + xOffset, 6.0f * size + y_cor, 0.0f);
        glVertex3f(6.0f * size + xOffset, 6.0f * size + y_cor, 0.0f);
        glVertex3f(4.52024f * size + xOffset, 4.01389f * size + y_cor, 0.0f);
    glEnd();

    /* Quads */
    glColor3f(0.4f, 0.4f, 0.4f);
    glBegin(GL_QUADS);
        glVertex3f(4.52024f * size + xOffset, 4.01389f * size + y_cor, 0.0f);
        glVertex3f(3.0f * size + xOffset, 2.0f * size + y_cor, 0.0f);
        glVertex3f(2.0f * size + xOffset, 5.0f * size + y_cor, 0.0f);
        glVertex3f(3.0f * size + xOffset, 6.0f * size + y_cor, 0.0f);
    glEnd();

    glColor3f(0.9f, 0.5f, 0.2f);
    glBegin(GL_QUADS);
        glVertex3f(4.52024f * size + xOffset, 4.01389f * size + y_cor, 0.0f);
        glVertex3f(6.0f * size + xOffset, 6.0f * size + y_cor, 0.0f);
        glVertex3f(7.0f * size + xOffset, 5.0f * size + y_cor, 0.0f);
        glVertex3f(6.0f * size + xOffset, 2.0f * size + y_cor, 0.0f);
    glEnd();

    glColor3f(0.5f, 0.2f, 0.9f);
    glBegin(GL_QUADS);
        glVertex3f(4.52024f * size + xOffset, 4.01389f * size + y_cor, 0.0f);
        glVertex3f(3.0f * size + xOffset, 2.0f * size + y_cor, 0.0f);
        glVertex3f(4.49548f * size + xOffset, 1.00485f * size + y_cor, 0.0f);
        glVertex3f(6.0f * size + xOffset, 2.0f * size + y_cor, 0.0f);
    glEnd();

    glColor3f(0.1f, 0.6f, 0.8f);
    glBegin(GL_QUADS);
        glVertex3f(6.24f * size + xOffset, 5.76f * size + y_cor, 0.0f);
```

```
        glVertex3f(7.0f * size + xOffset, 6.0f * size + y_cor, 0.0f);
        glVertex3f(8.0f * size + xOffset, 5.0f * size + y_cor, 0.0f);
        glVertex3f(6.67f * size + xOffset, 4.01f * size + y_cor, 0.0f);
    glEnd();

    glColor3f(0.7f, 0.3f, 0.1f);
    glBegin(GL_QUADS);
        glVertex3f(7.0f * size + xOffset, 6.0f * size + y_cor, 0.0f);
        glVertex3f(7.99f * size + xOffset, 5.77f * size + y_cor, 0.0f);
        glVertex3f(9.0f * size + xOffset, 5.0f * size + y_cor, 0.0f);
        glVertex3f(7.34f * size + xOffset, 5.02f * size + y_cor, 0.0f);
    glEnd();

    glColor3f(0.8f, 0.4f, 0.7f);
    glBegin(GL_QUADS);
        glVertex3f(2.83f * size + xOffset, 5.83f * size + y_cor, 0.0f);
        glVertex3f(2.0f * size + xOffset, 6.0f * size + y_cor, 0.0f);
        glVertex3f(1.0f * size + xOffset, 5.0f * size + y_cor, 0.0f);
        glVertex3f(2.34f * size + xOffset, 3.98f * size + y_cor, 0.0f);
    glEnd();

    glColor3f(0.2f, 0.7f, 0.9f);
    glBegin(GL_QUADS);
        glVertex3f(2.0f * size + xOffset, 6.0f * size + y_cor, 0.0f);
        glVertex3f(1.02f * size + xOffset, 5.77f * size + y_cor, 0.0f);
        glVertex3f(0.0f * size + xOffset, 5.0f * size + y_cor, 0.0f);
        glVertex3f(1.66f * size + xOffset, 5.02f * size + y_cor, 0.0f);
    glEnd();

    glColor3f(0.6f, 0.4f, 0.1f);
    glBegin(GL_QUADS);
        glVertex3f(2.93f * size + xOffset, 2.2f * size + y_cor, 0.0f);
        glVertex3f(2.5f * size + xOffset, 2.02f * size + y_cor, 0.0f);
        glVertex3f(2.0f * size + xOffset, 1.0f * size + y_cor, 0.0f);
        glVertex3f(3.43f * size + xOffset, 1.71f * size + y_cor, 0.0f);
    glEnd();

    glColor3f(0.3f, 0.8f, 0.1f);
    glBegin(GL_QUADS);
        glVertex3f(6.08f * size + xOffset, 2.24f * size + y_cor, 0.0f);
        glVertex3f(6.48f * size + xOffset, 2.01f * size + y_cor, 0.0f);
        glVertex3f(7.0f * size + xOffset, 1.0f * size + y_cor, 0.0f);
        glVertex3f(5.56f * size + xOffset, 1.71f * size + y_cor, 0.0f);
    glEnd();

    glFlush();
}
```

```
void jellyfish()
{
    float scalingFactor = 15;

    // JellyFish Body
    glColor3f(1.0, 0.5, 0.8);
    glBegin(GL_POLYGON);
    glVertex2f(5.9348448602658 * scalingFactor + x_cor, 9.8338256504978 * scalingFactor +
jellyPosY);
    glVertex2f(5.0173194818357 * scalingFactor + x_cor, 9.7213785329573 * scalingFactor +
jellyPosY);
    glVertex2f(4.3528592418237 * scalingFactor + x_cor, 9.2715900627953 * scalingFactor +
jellyPosY);
    glVertex2f(3.851958445507 * scalingFactor + x_cor, 8.484460240012 * scalingFactor +
jellyPosY);
    glVertex2f(3.5452844885784 * scalingFactor + x_cor, 7.499007862251 * scalingFactor +
jellyPosY);
    glVertex2f(4.1279650067427 * scalingFactor + x_cor, 7.5285701249633 * scalingFactor +
jellyPosY);
    glVertex2f(5.17427742399 * scalingFactor + x_cor, 7.4189975691832 * scalingFactor +
jellyPosY);
    glVertex2f(5.9348448602658 * scalingFactor + x_cor, 7.4132305925632 * scalingFactor +
jellyPosY);
    glEnd();

    glColor3f(0.8, 0.5, 1.0);
    glBegin(GL_POLYGON);
    glVertex2f(5.9348448602658 * scalingFactor + x_cor, 9.8338256504978 * scalingFactor +
jellyPosY);
    glVertex2f(6.8573632234072 * scalingFactor + x_cor, 9.7213785329573 * scalingFactor +
jellyPosY);
    glVertex2f(7.5116009981883 * scalingFactor + x_cor, 9.2409226671025 * scalingFactor +
jellyPosY);
    glVertex2f(8.012501794505 * scalingFactor + x_cor, 8.46401530955 * scalingFactor +
jellyPosY);
    glVertex2f(8.3079247164197 * scalingFactor + x_cor, 7.499007862251 * scalingFactor +
jellyPosY);
    glVertex2f(7.7467176985002 * scalingFactor + x_cor, 7.5285701249633 * scalingFactor +
jellyPosY);
    glVertex2f(6.7040262449429 * scalingFactor + x_cor, 7.4189975691832 * scalingFactor +
jellyPosY);
    glVertex2f(5.9348448602658 * scalingFactor + x_cor, 7.499007862251 * scalingFactor +
jellyPosY);
    glEnd();

    glColor3f(0.7, 0.4, 0.9);
```

```
    glBegin(GL_POLYGON);
    glVertex2f(3.5452844885784 * scalingFactor + x_cor, 7.5644383692262 * scalingFactor +
jellyPosY);
    glVertex2f(3.3825360815292 * scalingFactor + x_cor, 7.1197290810414 * scalingFactor +
jellyPosY);
    glVertex2f(3.5294472025071 * scalingFactor + x_cor, 6.9186928102296 * scalingFactor +
jellyPosY);
    glVertex2f(3.8387337729868 * scalingFactor + x_cor, 6.8491033318716 * scalingFactor +
jellyPosY);
    glVertex2f(4.5578250493523 * scalingFactor + x_cor, 7.1119969167794 * scalingFactor +
jellyPosY);
    glVertex2f(5.2034674552142 * scalingFactor + x_cor, 6.7797545149297 * scalingFactor +
jellyPosY);
    glVertex2f(5.9264181237254 * scalingFactor + x_cor, 7.0656039312075 * scalingFactor +
jellyPosY);
    glVertex2f(6.7040262449429 * scalingFactor + x_cor, 7.4928811126095 * scalingFactor +
jellyPosY);
    glEnd();

    glColor3f(0.6, 0.3, 0.8);
    glBegin(GL_POLYGON);
    glVertex2f(5.1865928162797 * scalingFactor + x_cor, 7.4956199863057 * scalingFactor +
jellyPosY);
    glVertex2f(5.9264181237254 * scalingFactor + x_cor, 7.0656039312075 * scalingFactor +
jellyPosY);
    glVertex2f(6.6455094000909 * scalingFactor + x_cor, 6.7717816892517 * scalingFactor +
jellyPosY);
    glVertex2f(7.2983721582929 * scalingFactor + x_cor, 7.1130892370081 * scalingFactor +
jellyPosY);
    glVertex2f(7.9909059816779 * scalingFactor + x_cor, 6.8568354961336 * scalingFactor +
jellyPosY);
    glVertex2f(8.3673621974261 * scalingFactor + x_cor, 6.9296840832352 * scalingFactor +
jellyPosY);
    glVertex2f(8.4721651424391 * scalingFactor + x_cor, 7.1550104150133 * scalingFactor +
jellyPosY);
    glVertex2f(8.3079247164197 * scalingFactor + x_cor, 7.5644383692262 * scalingFactor +
jellyPosY);
    glEnd();

    glColor3f(0.5, 0.2, 0.7);
    glBegin(GL_POLYGON);
    glVertex2f(4.5578250493523 * scalingFactor - 20 + x_cor, 7.1119969167794 * scalingFactor
+ jellyPosY);
    glVertex2f(4.6013728757244 * scalingFactor - 20 + x_cor, 6.7493970157177 * scalingFactor
+ jellyPosY);
    glVertex2f(4.2747910056077 * scalingFactor + x_cor, 6.2316452704107 * scalingFactor +
jellyPosY);
```

```
    glVertex2f(4.983712626105 * scalingFactor + x_cor, 6.3909534997359 * scalingFactor +
jellyPosY);
    glVertex2f(5.93159659059 * scalingFactor + x_cor, 6.0086137493554 * scalingFactor +
jellyPosY);
    glVertex2f(5.9264181237254 * scalingFactor + x_cor, 7.0656039312075 * scalingFactor +
jellyPosY);
    glVertex2f(4.5578250493523 * scalingFactor + x_cor, 7.1119969167794 * scalingFactor +
jellyPosY);
    glEnd();

    glColor3f(0.4, 0.1, 0.6);
    glBegin(GL_POLYGON);
    glVertex2f(5.9264181237254 * scalingFactor + x_cor, 7.0656039312075 * scalingFactor +
jellyPosY);
    glVertex2f(5.93159659059 * scalingFactor + x_cor, 6.0086137493554 * scalingFactor +
jellyPosY);
    glVertex2f(6.8954113780076 * scalingFactor + x_cor, 5.8062015054959 * scalingFactor +
jellyPosY);
    glVertex2f(7.9840067143846 * scalingFactor + x_cor, 6.0803051942225 * scalingFactor +
jellyPosY);
    glVertex2f(8.9185521901107 * scalingFactor + x_cor, 6.3816781461181 * scalingFactor +
jellyPosY);
    glVertex2f(8.8165594743617 * scalingFactor + x_cor, 7.1550104150133 * scalingFactor +
jellyPosY);


    glEnd();
    glColor3f(0.2, 0.8, 0.4);
    for (float x = 4.6268621924165 * scalingFactor + x_cor; x <= 7.2904957867342 *
scalingFactor + x_cor; x += 0.2365727205 * scalingFactor)
    {
        glBegin(GL_LINES);
        glVertex2f(x, 6.2977581855807 * scalingFactor + jellyPosY);
        glVertex2f(x, 3.8006016909079 * scalingFactor + jellyPosY);
        glEnd();
    }

    glFlush();
}


void fish2(int pos)
{

    glColor3f(1.0, 0.0, 0.5);
    glBegin(GL_POLYGON);
    glVertex2f(3.3367370835797 * 11+fishPosX2, 7.0145439628335 * 11 + pos);
```

```
    glVertex2f(5.9275065298185 * 11+fishPosX2, 8.1342923482976 * 11 + pos);
    glVertex2f(10.6935727078222 * 11+fishPosX2, 8.13429234829 * 11 + pos);
    glVertex2f(13.2380532578832 * 11+fishPosX2, 7.0145439628335 * 11 + pos);
    glVertex2f(10.6935727078222 * 11+fishPosX2, 6.1324138284434 * 11 + pos);
    glVertex2f(5.9275065298185 * 11+fishPosX2, 6.1324138284434 * 11 + pos);
    glVertex2f(3.3367370835797 * 11+fishPosX2, 7.0145439628335 * 11 + pos);
    glEnd();

    // Fish tail
    glBegin(GL_POLYGON);
    glVertex2f(13.2380532578832 * 11+fishPosX2, 7.0145439628335 * 11+ pos);
    glVertex2f(14.6 * 11+fishPosX2, 8.4 * 11+ pos);
    glVertex2f(14.1682052819829 * 11+fishPosX2, 7.0145439628335 * 11+ pos);
    glVertex2f(14.6 * 11+fishPosX2, 5.8 * 11+ pos);
    glVertex2f(13.2380532578832 * 11+fishPosX2, 7.0145439628335 * 11+ pos);
    glEnd();

    // Eye
    const float eyeRadius = 0.3 * 11;
    const int numSegments = 50;
    const float angleIncrement = 2.0 * M_PI / numSegments;
    const float centerX = 5.432516609874 * 11+fishPosX2;
    const float centerY = 7.2299742010855 * 11+ pos;

    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
    for (int i = 0; i < numSegments; ++i)
    {
        float angle = i * angleIncrement;
        float x = centerX + eyeRadius * cos(angle);
        float y = centerY + eyeRadius * sin(angle);
        glVertex2f(x, y);
    }
    glEnd();

    // Fins
    glColor3f(0.0, 1.0, 1.0);
    glBegin(GL_TRIANGLES);
    glVertex2f(6.4015607510735 * 11+fishPosX2, 8.1342923482976 * 11+ pos);
    glVertex2f(7.0741489011767 * 11+fishPosX2, 9.0252912175314 * 11+ pos);
    glVertex2f(7.0863057373307 * 11+fishPosX2, 8.1342923482976 * 11+ pos);
    glEnd();

    glBegin(GL_TRIANGLES);
    glVertex2f(6.4191183148237 * 11+fishPosX2, 6.1324138284434 * 11+ pos);
    glVertex2f(7.0687481735805 * 11+fishPosX2, 5.693790987022 * 11+ pos);
    glVertex2f(7.0863057373307 * 11+fishPosX2, 6.1324138284434 * 11+ pos);
```

```
    glEnd();

    glBegin(GL_TRIANGLES);
    glVertex2f(9.1405406961022 * 11+fishPosX2, 8.1342923482976 * 11+ pos);
    glVertex2f(9.8055456071312 * 11+fishPosX2, 8.5782083807511 * 11+ pos);
    glVertex2f(10.2805711212103 * 11+fishPosX2, 8.1342923482976 * 11+ pos);
    glEnd();

    // Mouth
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2f(6.0092761705584 * 11+fishPosX2, 7.6589715264465 * 11+ pos);
    glVertex2f(6.2568421167704 * 11+fishPosX2, 7.4353635750291 * 11+ pos);
    glVertex2f(6.2568421167704 * 11+fishPosX2, 7.4353635750291 * 11+ pos);
    glVertex2f(6.2648281150353 * 11+fishPosX2, 7.0280776635189 * 11+ pos);
    glEnd();

    // Line on the mouth
    glColor3f(1.0, 0.5, 1.0);
    glBegin(GL_LINES);
    glVertex2f(6.2568421167704 * 11+fishPosX2, 7.4353635750291 * 11+ pos);
    glVertex2f(6.2648281150353 * 11+fishPosX2, 7.0280776635189 * 11+ pos);
    glEnd();

    glFlush();
}
void fish4(int pos)
{

    glColor3f(0.0, 0.0, 1.0);
    glPushMatrix();

    glTranslatef(newFishPosX2, pos, 0.0);
    glRotatef(180.0, 0.0, 0.0, 1.0);
    glTranslatef(-newFishPosX2, -pos, 0.0);


    glBegin(GL_POLYGON);
    glVertex2f(3.3367370835797 * 11 + newFishPosX2, 7.0145439628335 * 11 + pos);
    glVertex2f(5.9275065298185 * 11 + newFishPosX2, 8.1342923482976 * 11 + pos);
    glVertex2f(10.6935727078222 * 11 + newFishPosX2, 8.13429234829 * 11 + pos);
    glVertex2f(13.2380532578832 * 11 + newFishPosX2, 7.0145439628335 * 11 + pos);
    glVertex2f(10.6935727078222 * 11 + newFishPosX2, 6.1324138284434 * 11 + pos);
    glVertex2f(5.9275065298185 * 11 + newFishPosX2, 6.1324138284434 * 11 + pos);
    glVertex2f(3.3367370835797 * 11 + newFishPosX2, 7.0145439628335 * 11 + pos);
    glEnd();
```

```
    glBegin(GL_POLYGON);
    glVertex2f(13.2380532578832 * 11 + newFishPosX2, 7.0145439628335 * 11 + pos);
    glVertex2f(14.6 * 11 + newFishPosX2, 8.4 * 11 + pos);
    glVertex2f(14.1682052819829 * 11 + newFishPosX2, 7.0145439628335 * 11 + pos);
    glVertex2f(14.6 * 11 + newFishPosX2, 5.8 * 11 + pos);
    glVertex2f(13.2380532578832 * 11 + newFishPosX2, 7.0145439628335 * 11 + pos);
    glEnd();

    // Draw the eye
    const float eyeRadius = 0.3 * 11;
    const int numSegments = 50;
    const float angleIncrement = 2.0 * M_PI / numSegments;
    const float centerX = 5.432516609874 * 11 + newFishPosX2;
    const float centerY = 7.2299742010855 * 11 + pos;

    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
    for (int i = 0; i < numSegments; ++i)
    {
        float angle = i * angleIncrement;
        float x = centerX + eyeRadius * cos(angle);
        float y = centerY + eyeRadius * sin(angle);
        glVertex2f(x, y);
    }
    glEnd();

    glColor3f(0.0, 1.0, 1.0);
    glBegin(GL_TRIANGLES);
    glVertex2f(6.4015607510735 * 11 + newFishPosX2, 8.1342923482976 * 11 + pos);
    glVertex2f(7.0741489011767 * 11 + newFishPosX2, 9.0252912175314 * 11 + pos);
    glVertex2f(7.0863057373307 * 11 + newFishPosX2, 8.1342923482976 * 11 + pos);
    glEnd();

    glBegin(GL_TRIANGLES);
    glVertex2f(6.4191183148237 * 11 + newFishPosX2, 6.1324138284434 * 11 + pos);
    glVertex2f(7.0687481735805 * 11 + newFishPosX2, 5.693790987022 * 11 + pos);
    glVertex2f(7.0863057373307 * 11 + newFishPosX2, 6.1324138284434 * 11 + pos);
    glEnd();

    glBegin(GL_TRIANGLES);
    glVertex2f(9.1405406961022 * 11 + newFishPosX2, 8.1342923482976 * 11 + pos);
    glVertex2f(9.8055456071312 * 11 + newFishPosX2, 8.5782083807511 * 11 + pos);
    glVertex2f(10.2805711212103 * 11 + newFishPosX2, 8.1342923482976 * 11 + pos);
    glEnd();

    // Draw the mouth
    glColor3f(1.0, 1.0, 0.0);
```

```
    glBegin(GL_LINES);
    glVertex2f(6.0092761705584 * 11 + newFishPosX2, 7.6589715264465 * 11 + pos);
    glVertex2f(6.2568421167704 * 11 + newFishPosX2, 7.4353635750291 * 11 + pos);
    glVertex2f(6.2568421167704 * 11 + newFishPosX2, 7.4353635750291 * 11 + pos);
    glVertex2f(6.2648281150353 * 11 + newFishPosX2, 7.0280776635189 * 11 + pos);
    glEnd();

    // Draw the line on the mouth
    glColor3f(0.5, 0.0, 0.5);
    glBegin(GL_LINES);
    glVertex2f(6.2568421167704 * 11 + newFishPosX2, 7.4353635750291 * 11 + pos);
    glVertex2f(6.2648281150353 * 11 + newFishPosX2, 7.0280776635189 * 11 + pos);
    glEnd();

    glPopMatrix();
    glFlush();
}
void drawlight(float r, float g, float b) {
   // Draw the sun
   glColor3f(r, g, b);
   float radius = 30.0;
   int centerX = windowWidth/2;
   int centerY = windowHeight-50;
   int numSegments = 100;
   glBegin(GL_TRIANGLE_FAN);
   glVertex2f(centerX, centerY);
   for (int i = 0; i <= numSegments; ++i) {
      float theta = 2.0f * 3.1415926f * float(i) / float(numSegments);
      float x = radius * cos(theta);
      float y = radius * sin(theta);
      glVertex2f(centerX + x, centerY + y);
   }
   glEnd();


   glColor3f(1.0, 1.0, 0.0);
   glLineWidth(2.0);
   glBegin(GL_LINES);
   for (int i = 0; i < numSegments; i += 10) {
      float theta = 2.0f * 3.1415926f * float(i) / float(numSegments);
      float x = radius * cos(theta);
      float y = radius * sin(theta);
      glVertex2f(centerX, centerY);
      glVertex2f(centerX + x, centerY + y);
   }
   glEnd();
}
```

```
void fish3(int s) {
   // Triangle 1
   glColor3f(0.95, 0.79, 0.44);
   glBegin(GL_TRIANGLES);
   glVertex3f(20.375566 + fishPosX, 35.024176 + s, 0.0);
   glVertex3f(20.21639 + fishPosX, 74.022232 + s, 0.0);
   glVertex3f(39.95418 + fishPosX, 54.28444 + s, 0.0);
   glEnd();

   // Triangle 2
   glColor3f(0.29, 0.47, 0.72);
   glBegin(GL_TRIANGLES);
   glVertex3f(39.95418 + fishPosX, 54.28444 + s, 0.0);
   glVertex3f(59.37362 + fishPosX, 73.70389 + s, 0.0);
   glVertex3f(59.055268 + fishPosX, 35.024176 + s, 0.0);
   glEnd();

   // Triangle 3
   glColor3f(0.78, 0.22, 0.36);
   glBegin(GL_TRIANGLES);
   glVertex3f(59.37362 + fishPosX, 73.70389 + s, 0.0);
   glVertex3f(78.792 + fishPosX, 54.125264 + s, 0.0);
   glVertex3f(59.055268 + fishPosX, 35.024176 + s, 0.0);
   glEnd();

   // Triangle 4
   glColor3f(0.61, 0.75, 0.34);
   glBegin(GL_TRIANGLES);
   glVertex3f(48.072143 + fishPosX, 62.561576 + s, 0.0);
   glVertex3f(39.635829 + fishPosX, 93.44167 + s, 0.0);
   glVertex3f(59.37362 + fishPosX, 73.70389 + s, 0.0);
   glEnd();

   // Triangle 5
   glColor3f(0.11, 0.45, 0.57);
   glBegin(GL_TRIANGLES);
   glVertex3f(48.072143 + fishPosX, 46.166477 + s, 0.0);
   glVertex3f(40.0 + fishPosX, 16.0 + s, 0.0);
   glVertex3f(59.055268 + fishPosX, 35.024176 + s, 0.0);
   glEnd();

   glFlush();
```

```
}


void fish1(int s) {
   // Triangle 1
   glColor3f(0.39, 0.58, 0.93);
   glBegin(GL_TRIANGLES);
   glVertex3f(20.375566 + fishPosX, 35.024176+s, 0.0);
   glVertex3f(20.21639 + fishPosX, 74.022232+s, 0.0);
   glVertex3f(39.95418 + fishPosX, 54.28444+s, 0.0);
   glEnd();

   // Triangle 2
   glColor3f(1.0, 0.56, 0.0);
   glBegin(GL_TRIANGLES);
   glVertex3f(39.95418 + fishPosX, 54.28444+s, 0.0);
   glVertex3f(59.37362 + fishPosX, 73.70389+s, 0.0);
   glVertex3f(59.055268 + fishPosX, 35.024176+s, 0.0);
   glEnd();

   // Triangle 3
   glColor3f(0.87, 0.43, 0.63);
   glBegin(GL_TRIANGLES);
   glVertex3f(59.37362 + fishPosX, 73.70389+s, 0.0);
   glVertex3f(78.792 + fishPosX, 54.125264+s, 0.0);
   glVertex3f(59.055268 + fishPosX, 35.024176+s, 0.0);
   glEnd();

   // Triangle 4
   glColor3f(0.4, 0.72, 0.41);
   glBegin(GL_TRIANGLES);
   glVertex3f(48.072143 + fishPosX, 62.561576+s, 0.0);
   glVertex3f(39.635829 + fishPosX, 93.44167+s, 0.0);
   glVertex3f(59.37362 + fishPosX, 73.70389+s, 0.0);
   glEnd();

   // Triangle 5
   glColor3f(0.93, 0.78, 0.31);
   glBegin(GL_TRIANGLES);
   glVertex3f(48.072143 + fishPosX, 46.166477+s, 0.0);
   glVertex3f(40.0 + fishPosX, 16.0+s, 0.0);
   glVertex3f(59.055268 + fishPosX, 35.024176+s, 0.0);
   glEnd();

   glFlush();
}
void fish1rev(int s) {
```

```
// Triangle 1
glColor3f(0.39, 0.58, 0.93);
glPushMatrix();
glTranslatef(fishPosX, 0.0, 0.0);
glRotatef(-180.0, 0.0, 0.0, 1.0);
glBegin(GL_TRIANGLES);
glVertex3f(-20.375566, 35.024176+s, 0.0);
glVertex3f(-20.21639, 74.022232+s, 0.0);
glVertex3f(-39.95418, 54.28444+s, 0.0);
glEnd();
glPopMatrix();

// Triangle 2
glColor3f(1.0, 0.56, 0.0);
glPushMatrix();
glTranslatef(fishPosX, 0.0, 0.0);
glRotatef(-180.0, 0.0, 0.0, 1.0);
glBegin(GL_TRIANGLES);
glVertex3f(-39.95418, 54.28444+s, 0.0);
glVertex3f(-59.37362, 73.70389+s, 0.0);
glVertex3f(-59.055268, 35.024176+s, 0.0);
glEnd();
glPopMatrix();

// Triangle 3
glColor3f(0.87, 0.43, 0.63);
glPushMatrix();
glTranslatef(fishPosX, 0.0, 0.0);
glRotatef(-180.0, 0.0, 0.0, 1.0);
glBegin(GL_TRIANGLES);
glVertex3f(-59.37362, 73.70389+s, 0.0);
glVertex3f(-78.792, 54.125264+s, 0.0);
glVertex3f(-59.055268, 35.024176+s, 0.0);
glEnd();
glPopMatrix();

// Triangle 4
glColor3f(0.4, 0.72, 0.41);
glPushMatrix();
glTranslatef(fishPosX, 0.0, 0.0);
glRotatef(-180.0, 0.0, 0.0, 1.0);
glBegin(GL_TRIANGLES);
glVertex3f(-48.072143, 62.561576+s, 0.0);
glVertex3f(-39.635829, 93.44167+s, 0.0);
glVertex3f(-59.37362, 73.70389+s, 0.0);
glEnd();
glPopMatrix();
```

```
   // Triangle 5
   glColor3f(0.93, 0.78, 0.31);
   glPushMatrix();
   glTranslatef(fishPosX, 0.0, 0.0);
   glRotatef(-180.0, 0.0, 0.0, 1.0);
   glBegin(GL_TRIANGLES);
   glVertex3f(-48.072143, 46.166477+s, 0.0);
   glVertex3f(-40.0, 16.0+s, 0.0);
   glVertex3f(-59.055268, 35.024176+s, 0.0);
   glEnd();
   glPopMatrix();

   glFlush();
}


void update(int value) {

    newFishPosX2 += 5;
    if (newFishPosX2>=1800){
    newFishPosX2 = 50;
    }
    y_cor += 2;
    if(y_cor > 1050){
    y_cor = 50;
    }
    jellyPosY += 5.0;
    if (jellyPosY > 1050){
    jellyPosY = 0;
    }
    fishPosX2 -= 5.0;
    if (fishPosX2 < 0){
    fishPosX2 = windowWidth-200;
    }
   fishPosX += 5.0;
   if (fishPosX > 1800.0)
      fishPosX = 50.0;

   glutPostRedisplay();
   glutTimerFunc(30, update, 0);
}
void drawGrass()
{

    GLfloat darkGreen[] = { 0.0f, 0.392f, 0.0f };
    GLfloat lightGreen[] = { 0.0f, 0.749f, 0.0f };
```

```
for (int j = 0; j < 2; j++) {
    glColor3fv(j == 0 ? darkGreen : lightGreen);
    glLineWidth(2.0f - j);

    for (int i = 50; i < 1870; i += 10) {
        glBegin(GL_LINES);
        glVertex2f(i, 0.0f);
        glVertex2f(i + 5, 30.0f - j * 10);
        glEnd();

        glBegin(GL_LINES);
        glVertex2f(i + 5, 30.0f - j * 10);
        glVertex2f(i + 10, 0.0f);
        glEnd();

        glBegin(GL_LINES);
        glVertex2f(i + 5, 30.0f - j * 10);
        glVertex2f(i + 5, 50.0f - j * 10);
        glEnd();

        glBegin(GL_LINES);
        glVertex2f(i + 2, 0.0f);
        glVertex2f(i + 3, 15.0f - j * 5);
        glEnd();

        glBegin(GL_LINES);
        glVertex2f(i + 8, 0.0f);
        glVertex2f(i + 7, 15.0f - j * 5);
        glEnd();

        glBegin(GL_LINES);
        glVertex2f(i + 3, 15.0f - j * 5);
        glVertex2f(i + 5, 20.0f - j * 5);
        glEnd();

        glBegin(GL_LINES);
        glVertex2f(i + 7, 15.0f - j * 5);
        glVertex2f(i + 5, 20.0f - j * 5);
        glEnd();

        glBegin(GL_LINES);
        glVertex2f(i, 0.0f);
        glVertex2f(i + 3, 20.0f - j * 8);
        glEnd();
```

```
            glBegin(GL_LINES);
            glVertex2f(i + 3, 20.0f - j * 8);
            glVertex2f(i + 6, 0.0f);
            glEnd();

            glBegin(GL_LINES);
            glVertex2f(i + 2, 0.0f);
            glVertex2f(i + 4, 20.0f - j * 8);
            glEnd();

            glBegin(GL_LINES);
            glVertex2f(i + 4, 20.0f - j * 8);
            glVertex2f(i + 7, 0.0f);
            glEnd();

            glBegin(GL_LINES);
            glVertex2f(i + 1, 0.0f);
            glVertex2f(i + 5, 18.0f - j * 7);
            glEnd();

            glBegin(GL_LINES);
            glVertex2f(i + 5, 18.0f - j * 7);
            glVertex2f(i + 9, 0.0f);
            glEnd();
        }
    }
}
void drawAquarium()
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0.678f, 0.847f, 0.902f);
    glBegin(GL_POLYGON);
    glVertex2f(0.0f, 0.0f);
    glVertex2f(0.0f, 1080.0f);
    glVertex2f(1920.0f, 1080.0f);
    glVertex2f(1920.0f, 0.0f);
    glEnd();

    glColor3f(0.282f, 0.239f, 0.545f);
    glBegin(GL_POLYGON);
    glVertex2f(0.0f, 0.0f);
    glVertex2f(50.0f, 0.0f);
    glVertex2f(50.0f, 1080.0f);
    glVertex2f(0.0f, 1080.0f);
    glEnd();
```

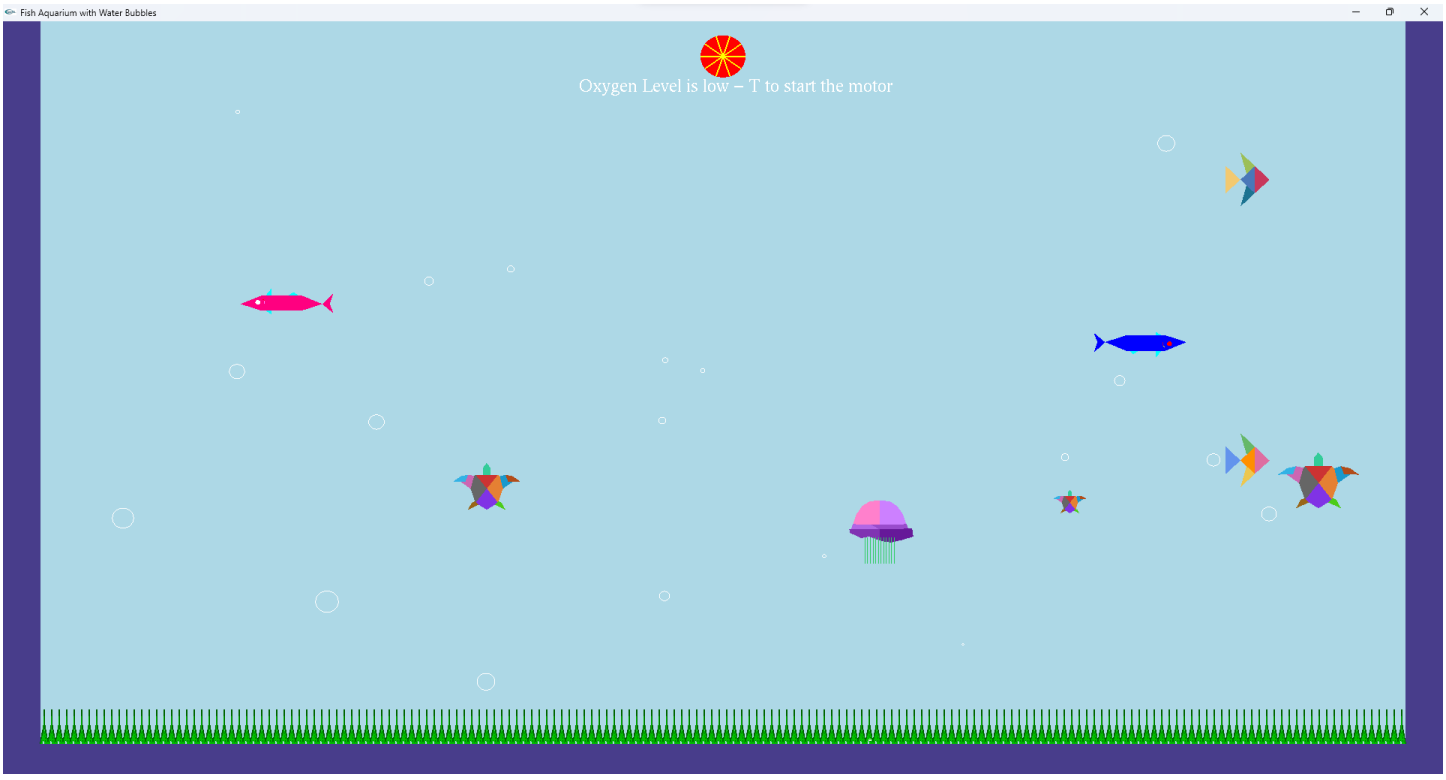
```
    glBegin(GL_POLYGON);
    glVertex2f(1870.0f, 0.0f);
    glVertex2f(1920.0f, 0.0f);
    glVertex2f(1920.0f, 1080.0f);
    glVertex2f(1870.0f, 1080.0f);
    glEnd();

    glBegin(GL_POLYGON);
    glVertex2f(0.0f, 0.0f);
    glVertex2f(1920.0f, 0.0f);
    glVertex2f(1920.0f, 50.0f);
    glVertex2f(0.0f, 50.0f);
    glEnd();
    glPushMatrix();
    glTranslatef(0.0f, 50.0f, 0.0f);
    drawGrass();
    glPopMatrix();
    glFlush();
}
void drawBubbles() {

   glColor3f(1.0, 1.0, 1.0);
   glPointSize(1.0);
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

   int streamX = 250;
   int streamY = 100;

   for (int i = 0; i < numBubbles; ++i) {
      glPushMatrix();
      glTranslatef(bubblePositionX[i], bubblePositionY[i], 0.0);

      float dx = bubblePositionX[i] - streamX;
      float dy = bubblePositionY[i] - streamY;
      float distance = sqrt(dx * dx + dy * dy);

      float bubbleSize = bubbleRadius * (1.0 - distance / (windowHeight - streamY));

      glBegin(GL_POLYGON);
      for (int j = 0; j <= 360; j++) {
         float angle = j * 3.14159 / 180.0;
         float x = bubbleSize * cos(angle);
         float y = bubbleSize * sin(angle);
         glVertex2f(x, y);
      }
      glEnd();
      glPopMatrix();
```

```
    }

    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glFlush();
}


void updateBubbles() {
    for (int i = 0; i < numBubbles; ++i) {
        bubblePositionY[i] += bubbleSpeed;

        if (bubblePositionY[i] > windowHeight - 50) {
            bubblePositionX[i] = rand() % (windowWidth - 100) + 50;
            bubblePositionY[i] = 50;
        }

        for (int j = 0; j < numBubbles; ++j) {
            if (i == j) continue;

            float dx = bubblePositionX[i] - bubblePositionX[j];
            float dy = bubblePositionY[i] - bubblePositionY[j];
            float distance = sqrt(dx * dx + dy * dy);

            if (distance < 2 * bubbleRadius) {
                bubblePositionX[i] += (2 * bubbleRadius - distance) * dx / distance;
                bubblePositionY[i] += (2 * bubbleRadius - distance) * dy / distance;
            }
        }
    }
}
void init() {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, windowWidth, 0.0, windowHeight);

    for (int i = 0; i < numBubbles; ++i) {
        bubblePositionX[i] = rand() % (windowWidth - 100) + 50;
        bubblePositionY[i] = rand() % (windowHeight - 100) + 50;
    }
}


void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    drawAquarium();
```

```
    drawBubbles();

     fish1(400);
     fish2(600);
     fish3(800);
     fish4(700);

     fish1rev(400);
     jellyfish();
     turtle(10, 600);
     turtle(12, 1700);
     turtle(5, 1400);
     if (motorStarted) {
         displayText(windowWidth / 2.2, windowHeight - 100, 1, 1, 1, "Motor Started!");
         drawlight(0.0, 1.0, 0.0);
     } else {
         displayText(windowWidth / 2.5, windowHeight - 100, 1, 1, 1, "Oxygen Level is low - T
to start the motor");
         drawlight(1.0, 0.0, 0.0);
     }
    glutSwapBuffers();
}

void timer(int) {
    updateBubbles();
    glutPostRedisplay();
    glutTimerFunc(10, timer, 0);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Fish Aquarium with Water Bubbles");
    init();
    glutDisplayFunc(display);
    glutTimerFunc(10, timer, 0);
    glutTimerFunc(0, update, 0);
    glutKeyboardFunc(handleKeyPress);
    glutMainLoop();
    return 0;
}
```

Output

Low Oxygen Indicator Before:



Good Level of Oxygen indicator After: