Computer Graphics (UCS505)

Project on

Open World 3D view of Undersea Realty: Conch Street 3D

Submitted By:

Nitesh Yadav 102217260
Ashutosh Swarnakar 102217263
Ravinder Singh 102217039

B.E. Third Year – COPC

Submitted To:

Dr. Rajkumar Tekchandani

Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
Patiala – 147001

Table Of Contents

1) Introduction to Project:

1.1 Intro

Step into the whimsical underwater realm of **Conch Street**, where our interactive 3D experience reimagines the beloved neighborhood from the iconic animated series. Designed to immerse players in the charming world of **SpongeBob SquarePants**, this project invites users to explore a vibrant digital recreation of Bikini Bottom's most famous street. As users navigate through the environment using keyboard controls, they'll encounter familiar homes, quirky characters, and a unique twist— each resident is looking to sell their underwater abode.

1.2 Project Overview

Built using **OpenGL and C++**, this project delivers a rich, animated 3D world that captures the personality and atmosphere of Conch Street. Users can freely explore the neighborhood, interact with its elements, and even make virtual property purchases by collecting scattered bundles of cash.

Core Features:

- **Free Exploration:** Navigate through the 3D environment using keyboard inputs and uncover all corners of Conch Street, from SpongeBob's pineapple house to Squidward's Easter Island head.
- **Detailed Graphics:** Leveraging OpenGL's powerful rendering capabilities, the project features visually appealing models, iconic structures, and vibrant underwater aesthetics.
- **Interactive Real Estate:** Encounter "For Sale" signs outside each character's home. By collecting in-game currency, users can purchase properties from well-known characters like SpongeBob, Patrick, and Squidward.
- **Engaging Atmosphere:** From animated sea life to familiar soundscapes, the experience is designed to fully immerse users in the underwater world they know and love.

1.3 Technical Details:

The project leverages OpenGL's capabilities to render 3D objects, manage user inputs for movement and interaction, and handle collision detection for purchasing homes.

2) Computer Graphics Concepts Used:

1. Translation:

Translation involves moving objects from one position to another in the 3D space. In the project, translation might be used to animate objects or position them at specific locations within the scene. This concept is achieved by modifying the coordinates of each vertex of the object along the x, y, and z axes.

2. Scaling:

Scaling changes the size of objects in the scene. It involves multiplying the coordinates of each vertex by a scaling factor along each axis. Scaling can be uniform (equal scaling along all axes) or non-uniform (different scaling factors along each axis). It's commonly used for zooming effects or adjusting the size of objects relative to each other.

3. Rotation:

Rotation changes the orientation of objects in the 3D space. It involves rotating objects around a specified axis by a certain angle. Rotation can be performed using various methods such as Euler angles, matrices, or quaternions. It's essential for animating objects and creating dynamic visual effects in the scene.

4. Transformation Matrices:

Transformation matrices are mathematical representations used to apply translation, scaling, and rotation transformations to objects. These matrices store the transformation operations and are multiplied with the object's vertex coordinates to achieve the desired transformation. Matrices enable efficient manipulation of objects in 3D space and are fundamental to computer graphics programming.

5. Projection:

Projection is the process of converting 3D coordinates into 2D coordinates for rendering on a 2D screen. In the project, perspective projection might be used to simulate realistic depth perception, where objects farther away from the viewer appear smaller. This creates a sense of depth and immersion in the rendered scene.

6. Lighting:

Lighting is crucial for rendering realistic scenes. Different lighting models such as ambient, diffuse, and specular lighting might be used to simulate how light interacts with objects in the scene. Shadows, reflections, and refractions can also be achieved through advanced lighting techniques like ray tracing or global illumination.

7. Texturing:

Texturing involves applying images (textures) to surfaces of 3D objects to add detail and realism. Texture mapping techniques such as UV mapping or procedural texturing might be used to wrap textures onto objects accurately. This enhances the visual quality of the rendered scene by adding surface details like color, patterns, and roughness.

8. Shading:

Shading determines how light interacts with the surface of objects and affects their appearance. Different shading models such as Phong shading or Blinn-Phong shading might be used to calculate the color and intensity of each pixel in the rendered image. This contributes to the overall visual quality and realism of the scene.

3) User Defined Functions:

1) changeSize(int w, int h):

This function is responsible for adjusting the viewport and the projection matrix when the window size changes. It calculates the aspect ratio of the window and sets the perspective accordingly.

2) mainDisplayCallback(void): This function is the main display callback for GLUT. It clears the color and depth buffers, calls cameraHandler() to update the camera position, drawStuff() to render the scene, and renderText() to display text on the screen. Finally, it swaps the buffers to display the rendered scene.

3) drawStuff(): This function is responsible for rendering the entire scene. It first resets the transformations, sets up the camera using gluLookAt(), and then renders the ground, roads, houses, signs, plants, rocks, and money stacks using various OpenGL primitives and custom shapes.

4) drawPatrickHouse(): This function draws Patrick's house using OpenGL primitives. It creates a spherical structure representing the house.

5) drawSquidwardHouse(): This function draws Squidward's house using a combination of OpenGL primitives and custom shapes. It creates the base, ears, nose, eyes, and other features of Squidward's house.

6) drawSpongebobHouse(): This function draws Spongebob's house using a combination of OpenGL primitives and custom shapes. It creates the base, door, windows, and leaves around Spongebob's house.

7) drawSign(string text): This function draws a sign with the specified text using OpenGL primitives. It creates a cube for the sign's post and another cube for the signboard, displaying the provided text.

8) keyDown(unsigned char key, int x, int y): This function handles key press events for regular keys. It sets various flags based on the keys pressed, such as movement controls and house purchase actions.

9) keyUp(unsigned char key, int x, int y): This function handles key release events for regular keys. It resets the corresponding flags when keys are released.

10) specialDown(int key, int x, int y): This function handles key press events for special keys (e.g., arrow keys). It sets flags for camera movement based on the keys pressed.

11) specialUp(int key, int x, int y): This function handles key release events for special keys. It resets flags for camera movement when keys are released.

12) jump(): This function handles the jumping behavior of the player. It updates the player's vertical position based on the current jump energy.

13) cameraHandler(): This function updates the camera position and orientation based on user input and game state. It handles camera movement, including looking around and moving forward/backward.

14) renderText(): This function displays text on the screen using OpenGL primitives. It sets up the orthographic projection and modelview matrices to render 2D text on the screen.

4) Code:

```
//
//  main.cpp
//  CG
//
//  Created by Nitesh Yadav on 03/05/2025.
//
#include <iostream> #include
<vector> #include <string>
#include <GLUT/glut.h>
#include <math.h>

using namespace std;

// Actual vector representing the camera's direction
float lx = 0.0f, lz = -1.0f;
// XZ position of the camera
float x = 0.0f, y = 1.0f, z = 5.0f;
// Angle of camera
float angle = 0.0f;
// Camera x sensitivity
float x_sens = 0.03f;
// Camera y sensitivity
float y_sens = 0.02f;
// Camera move speed
```

```
float speed = 0.55f;

// Movement variables
```

```
bool is_left, is_right, is_forward, is_backward, is_look_left, is_look_right,
is_look_down, is_look_up, is_light = false; bool is_jumping = false;
bool on_ground = true;
bool byPatrickHouse = false; bool
bySquidwardHouse = false; bool
bySpongebobHouse = false; int jump_energy,
energy_max = 25; double jump_force =
0.053; double jump_speed = 0.15;

// Plant variables vector<vector<vector<float>>>
plantMatrix; unsigned int number_of_plants = 500;

// Text and font variables std::string draw_text;
void* helvetica = GLUT_BITMAP_HELVETICA_18; void*
roman = GLUT_BITMAP_TIMES_ROMAN_24; void* font =
GLUT_STROKE_ROMAN;

// Color and alpha key variables
int blackColor[3] = { 0, 0, 0 };
int whiteColor[3] = { 255, 255, 255 };

// Window variables int
windowWidth = 1000; int
windowHeight = 600;

// Currency
int funds = 6;
bool stack1_collision = false; bool
stack2_collision = false; bool stack3_collision
= false; bool stack4_collision = false; bool
secretstack_collision = false;

// House sold variables bool
spongebob_sold = false; bool
patrick_sold = false; bool
squidward_sold = false;

// Prototypes
void changeSize(int w, int h); void
mainDisplayCallback(void); void drawStuff();
void drawPatrickHouse(); void
drawSquidwardHouse(); void
drawSpongebobHouse();
void timerCallback(int value);
void keyDown(unsigned char key, int x, int y);
void keyUp(unsigned char key, int x, int y);
void specialDown(int key, int x, int y); void specialUp(int
key, int x, int y); void cameraHandler();
void renderText();
void drawText(std::string text, int x, int y, int rgb[3], void* font);
void drawLeaf(float size, float rotx, float roty, float rotz, int
seed);
void drawPlantLeaf(float size, float rotx, float roty, float rotz,
int parent_index, int index);
void drawPlant(float size, int index);
void drawSign(string text);
void createPlant(int seed);
void drawRock(GLfloat angle, double size, GLint smoothness);
void checkByHouses();
void drawSaleInfoSign();
void checkMoneyCollision();


int main(int argc, char** argv) {
        // Initialize GLUT and create window glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowPosition(100, 100); glutInitWindowSize(windowWidth,
        windowHeight); glutCreateWindow("Project 3 - Team 1");

        // Register callbacks
        glutDisplayFunc(mainDisplayCallback);
        glutReshapeFunc(changeSize);
        glutKeyboardFunc(keyDown);
        glutKeyboardUpFunc(keyUp);
        glutSpecialFunc(specialDown);
        glutSpecialUpFunc(specialUp); glutTimerFunc(8,
        timerCallback, 123);

        // OpenGL initilization glEnable(GL_DEPTH_TEST);
        glClearColor(0.047f, 0.64f, 1.0f, 0); // Background/sky color
        // Plant initilization
        for (size_t i = 0; i < number_of_plants; i++) createPlant(i *
                rand());

        glutMainLoop();
        return 1;
}


void timerCallback(int value) { mainDisplayCallback();
        glutTimerFunc(8, timerCallback, 123);
}
```

```
void mainDisplayCallback(void) { glClear(GL_COLOR_BUFFER_BIT |
        GL_DEPTH_BUFFER_BIT);
        cameraHandler();
        drawStuff(); renderText();
        glutSwapBuffers();
}


void drawStuff() {
        // Reset transformations glLoadIdentity();

        // Set the camera
        if (is_jumping)
                gluLookAt(x, y, z, x + lx, y, z + lz, 0.0f, 1.0f, 0.0f);
        else
                gluLookAt(x, 1.0f, z, x + lx, y, z + lz, 0.0f, 1.0f,
0.0f);

        checkByHouses();
        checkMoneyCollision();

        // Draw ground glColor3ub(249, 213,
        187); glBegin(GL_QUADS);
        glVertex3f(-1000.0f, 0.0f, -1000.0f); glVertex3f(-
        1000.0f, 0.0f, 1000.0f); glVertex3f(1000.0f, 0.0f,
        1000.0f); glVertex3f(1000.0f, 0.0f, -1000.0f); glEnd();

        // Draw street road glColor3ub(160, 160,
        160); glBegin(GL_QUADS);
        glVertex3f(-1000.0f, 0.01f, -4.0f); glVertex3f(-
        1000.0f, 0.01f, 4.0f); glVertex3f(1000.0f, 0.01f,
        4.0f); glVertex3f(1000.0f, 0.01f, -4.0f); glEnd();

        // Road to Squidward's House glColor3ub(50,
        50, 50); glBegin(GL_QUADS); glVertex3f(2.0f,
        0.02f, -35.0f); glVertex3f(8.0f, 0.02f, -35.0f);
        glVertex3f(8.0f, 0.02f, -31.0f); glVertex3f(2.0f,
        0.02f, -31.0f); glEnd();
```

```
glBegin(GL_QUADS); glVertex3f(2.0f, 0.02f, -
27.0f); glVertex3f(8.0f, 0.02f, -27.0f);
glVertex3f(8.0f, 0.02f, -23.0f); glVertex3f(2.0f,
0.02f, -23.0f); glEnd();

glBegin(GL_QUADS); glVertex3f(2.0f, 0.02f, -
19.0f); glVertex3f(8.0f, 0.02f, -19.0f);
glVertex3f(8.0f, 0.02f, -15.0f); glVertex3f(2.0f,
0.02f, -15.0f); glEnd();

glBegin(GL_QUADS); glVertex3f(2.0f, 0.02f, -
11.0f); glVertex3f(8.0f, 0.02f, -11.0f);
glVertex3f(8.0f, 0.02f, -7.0f);
glVertex3f(2.0f, 0.02f, -7.0f); glEnd();

// Road to Patrick's House
glColor3ub(50, 50, 50);
glBegin(GL_QUADS);
glVertex3f(-17.5f, 0.02f, -36.0f);
glVertex3f(-12.5f, 0.02f, -36.0f);
glVertex3f(-12.5f, 0.02f, -3.9f);
glVertex3f(-17.5f, 0.02f, -3.9f); glEnd();

// Road to Spongebob's House glColor3ub(50,
50, 50); glBegin(GL_QUADS); glVertex3f(23.0f,
0.02f, -35.0f); glVertex3f(27.0f, 0.02f, -35.0f);
glVertex3f(27.0f, 0.02f, -3.9f); glVertex3f(23.0f,
0.02f, -3.9f); glEnd();

// Draw Patrick's house
glPushMatrix(); glTranslatef(-15, 0, -
40); drawPatrickHouse();
glPopMatrix();

// Draw Squidward's house
glPushMatrix(); glTranslatef(5, 0, -40);
drawSquidwardHouse(); glPopMatrix();

// Draw Spongebob's house
```

```
        glPushMatrix(); glTranslatef(25, 0, -40);
        drawSpongebobHouse(); glPopMatrix();

        // "For Sale" signs
        if (!patrick_sold) { glPushMatrix();
                glTranslatef(-12, 0, -10);
                drawSign("For Sale"); glPopMatrix();
        }
        if (!squidward_sold) { glPushMatrix();
                glTranslatef(9, 0, -10);
                drawSign("For Sale");
                glPopMatrix();
        }
        if (!spongebob_sold) { glPushMatrix();
                glTranslatef(28, 0, -10);
                drawSign("For Sale");
                glPopMatrix();
        }

        glPushMatrix(); glTranslatef(1100, 0, 0);
        drawSign("EasterEgg"); glPopMatrix();

        // Plants
        for (size_t i = 0; i < number_of_plants; i++) { glPushMatrix();
                glTranslatef(plantMatrix[i][1][0], 0, plantMatrix[i][1]
    [1]);

                drawPlant(0.25, i); glPopMatrix();
        }

        //  Rocks  glPushMatrix();
        glTranslatef(35, 0, 10);
        drawRock(0, 0.33, 5); glPopMatrix();
        glPushMatrix(); glTranslatef(-25, 0,
        5.25);
        drawRock(20.0f, 0.5, 8);
        glPopMatrix(); glPushMatrix();
        glTranslatef(-1, 0, -40);
        drawRock(40.0f, 0.33, 5); glPopMatrix();
```

```
glPushMatrix(); glTranslatef(18.5, 0, -20);
drawRock(60.0f, 0.25, 5);
glPopMatrix(); glPushMatrix();
glTranslatef(18, 0, -18);
drawRock(80.0f, 0.33, 5);
glPopMatrix(); glPushMatrix();
glTranslatef(11.5, 0, -50);
drawRock(100.0f, 0.25, 8);
glPopMatrix(); glPushMatrix();
glTranslatef(-8, 0, -5);
drawRock(120.0f, 0.25, 5); glPopMatrix();

// Money
// Stack 1
if (!stack1_collision) { glPushMatrix();
        glColor3ub(15, 78, 12);
        glTranslatef(-6, -1, -5);
        glRotatef(30.0f, 0, 1.0f, 0);
        glScalef(1.2f, 5.0f, 1.5f);
        glutSolidCube(0.47); glPopMatrix();
}
// Stack 2
if (!stack2_collision) { glPushMatrix();
        glColor3ub(15, 78, 12);
        glTranslatef(-16, -1, -15);
        glRotatef(20.0f, 0, 1.0f, 0);
        glScalef(1.2f, 5.0f, 1.5f);
        glutSolidCube(0.47); glPopMatrix();
}
// Stack 3
if (!stack3_collision) { glPushMatrix();
        glColor3ub(15, 78, 12);
        glTranslatef(-22, -1, -40);
        glRotatef(10.0f, 0, 1.0f, 0);
        glScalef(1.2f, 5.0f, 1.5f);
        glutSolidCube(0.47); glPopMatrix();
}
// Stack 4
if (!stack4_collision) { glPushMatrix();
        glColor3ub(15, 78, 12);
```

```
                glTranslatef(-42, -1, -36);
                glRotatef(15.0f, 0, 1.0f, 0);
                glScalef(1.2f, 5.0f, 1.5f);
                glutSolidCube(0.47); glPopMatrix();
        }
        // Secret Stash
        if (!secretstack_collision) {
                glPushMatrix(); glColor3ub(15, 78,
                12);
                glTranslatef(3, -1, -40);
                glRotatef(15.0f, 0, 1.0f, 0);
                glScalef(1.2f, 5.0f, 1.5f);
                glutSolidCube(0.47); glPopMatrix();
        }
}


void drawPatrickHouse() { glColor3ub(102, 18,
        40);
        glutSolidSphere(5, 20, 20);
}


void drawSquidwardHouse() {
        GLUquadricObj* baseObj = gluNewQuadric();
        gluQuadricDrawStyle(baseObj, GLU_FILL);

        // Base of house glPushMatrix();
        glColor3ub(8, 33, 93);
        glRotatef(-90.0f, 1.0f, 0, 0);
        gluCylinder(baseObj, 5, 4.0f, 15, 15, 15); glPopMatrix();

        // Left ear glPushMatrix();
        glColor3ub(37, 53, 103);
        glTranslatef(4.8f, 7.5f, 0.0f); glRotatef(4.0f, 0, 0, 1.0f);
        glScalef(1.2f, 5.0f, 1.5f);
        glutSolidCube(1); glPopMatrix();

        // Right ear glPushMatrix();
        glColor3ub(37, 53, 103);
        glTranslatef(-4.8f, 7.5f, 0.0f); glRotatef(-4.0f,
        0, 0, 1.0f);
        glScalef(1.2f, 5.0f, 1.5f);
        glutSolidCube(1);
```

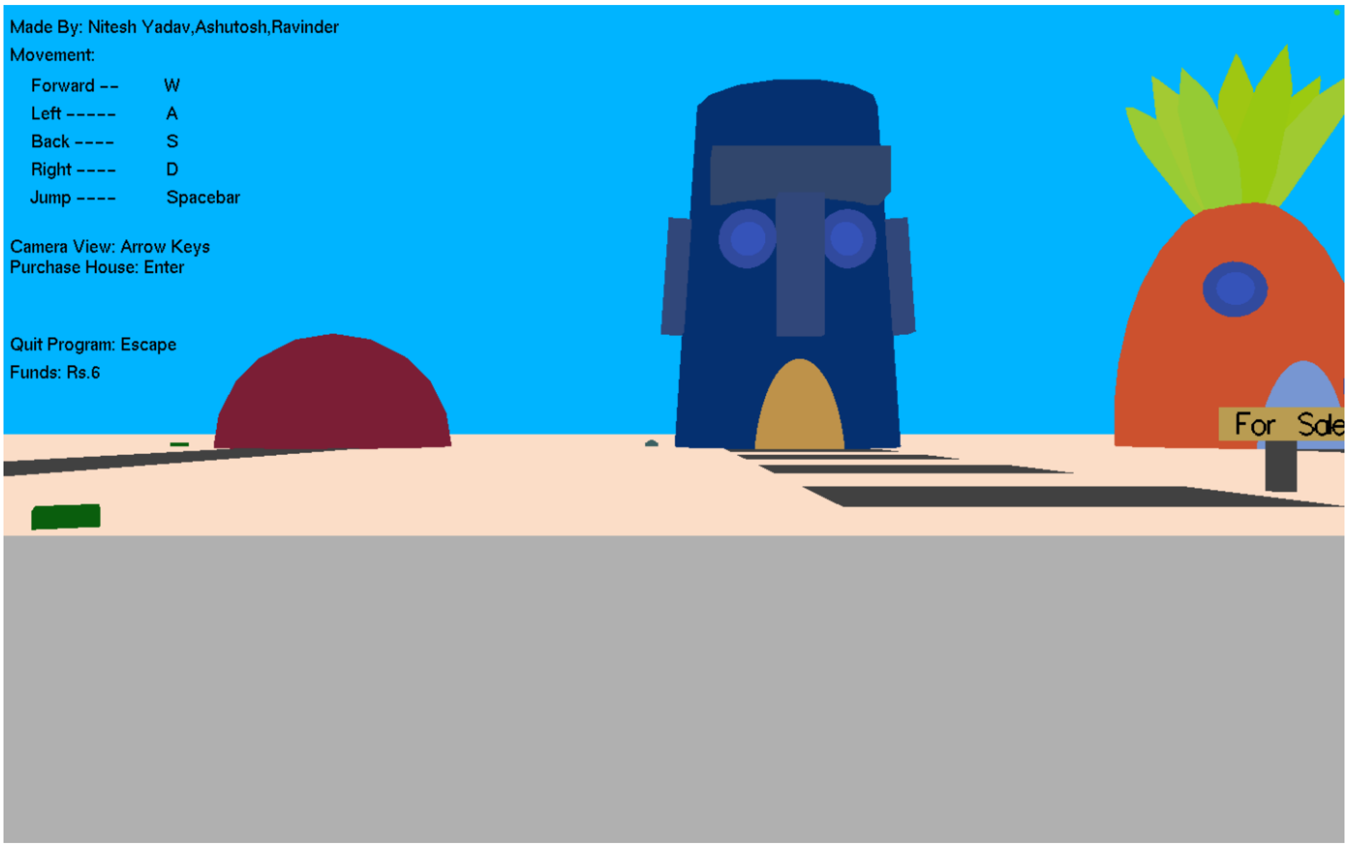
```
glPopMatrix();

// Door glPushMatrix();
glColor3ub(175, 128, 58); glTranslatef(0.0f, -
1.0f, 4.75f); glRotatef(-90.0f, 0, 1.0f, 0.0f);
glScalef(0.2f, 2.0f, 0.8f); glutSolidSphere(2.3f, 30, 30);
glPopMatrix();

// Nose glPushMatrix();
glColor3ub(37, 53, 103); glTranslatef(0,
8.0f, 4.1f); glScalef(1.8f, 7.0f, 2.5f);
glutSolidCube(1); glPopMatrix();

// Eyebrow glPushMatrix();
glColor3ub(37, 53, 89);
glTranslatef(0, 11.0f, 4.2f); glRotatef(90.0f, 0, 0.0f,
1.0f);
glScalef(1.8f, 7.0f, 2.5f);
glutSolidCube(1); glPopMatrix();

// Left eye glPushMatrix();
glColor3ub(37, 53, 140);
glTranslatef(2.0f, 8.5f, 4.0f); glRotatef(5.0f, 0, 1.0f, 0.0f);
glScalef(1.0f, 1.0f, 2.0f); glutSolidTorus(0.25, 0.95, 20, 20);
glPopMatrix();
glPushMatrix(); glColor3ub(40, 60,
170);
glTranslatef(2.0f, 8.5f, 4.0f); glRotatef(15.0f, 0, 1.0f,
0.0f); gluDisk(baseObj, 0, 1, 20, 20); glPopMatrix();

// Right Eye glPushMatrix();
glColor3ub(37, 53, 140);
glTranslatef(-2.0f, 8.5f, 4.0f); glRotatef(-5.0f,
0, 1.0f, 0.0f);
glScalef(1.0f, 1.0f, 2.0f); glutSolidTorus(0.25, 0.95, 20, 20);
glPopMatrix();
glPushMatrix();
```

```
        glColor3ub(40, 60, 170); glTranslatef(-2.0f,
        8.5f, 4.0f); glRotatef(-15.0f, 0, 1.0f, 0.0f);
        gluDisk(baseObj, 0, 1, 20, 20); glPopMatrix();

        gluDeleteQuadric(baseObj);
}

// Leaves for Spongebob's House
void drawLeaf(float size, float rotx, float roty, float rotz, int
seed) {
        std::srand(seed);
        glPushMatrix();
        glColor3ub(150 - (rand() % 25), 215 - (rand() % 25), 40 -
(rand() % 25));
        glTranslatef(0.0f, 10.0f, 0.0f); glRotatef(rotx, 1.0f, 0.0f,
        0.0f); glRotatef(roty, 0.0f, 1.0f, 0.0f); glRotatef(rotz,
        0.0f, 0.0f, 1.0f); glScalef(.7f, 5.0f, 0.7f);
        glutSolidSphere(size, 10, 10); glPopMatrix();
}


void drawSpongebobHouse() {
        GLUquadricObj* baseObj = gluNewQuadric();
        gluQuadricDrawStyle(baseObj, GLU_FILL);

        // Structure glPushMatrix();
        glColor3ub(190, 60, 35);
        glTranslatef(0.0f, 1.0f, 0.0f); glScalef(1.0f, 2.0f, 1.0f);
        glutSolidSphere(5, 30, 30); glPopMatrix();

        // Door glPushMatrix();
        glColor3ub(100, 130, 200); glTranslatef(0.0f, -
        1.0f, 4.75f); glRotatef(-90.0f, 0, 1.0f, 0.0f);
        glScalef(0.2f, 2.0f, 0.8f); glutSolidSphere(2.3f, 30, 30);
        glPopMatrix();

        // Window 1 glPushMatrix();
        glColor3ub(37, 53, 140);
        glTranslatef(-2.1f, 6.5f, 3.7f); glRotatef(-20.0f, 2.5f,
        2.5f, 0.0f);
        glScalef(1.0f, 1.0f, 1.0f); glutSolidTorus(0.25, 0.95, 20, 20);
        glPopMatrix();
        glPushMatrix(); glColor3ub(40, 60,
        170);
        glTranslatef(-2.1f, 6.5f, 3.75f); glRotatef(-20.0f,
        3.5f, 5.5f, 0.0f); gluDisk(baseObj, 0, 1, 20, 20);
        glPopMatrix();

        // Window 2 glPushMatrix();
        glColor3ub(37, 53, 140);
        glTranslatef(2.7f, 2.5f, 4.35f); glRotatef(30.0f, 0.0f, 1.0f, 0.0f);
        glScalef(1.0f, 1.0f, 1.0f); glutSolidTorus(0.25, 0.95, 20, 20);
        glPopMatrix();
        glPushMatrix(); glColor3ub(40, 60,
        170);
        glTranslatef(2.7f, 2.5f, 4.50f); glRotatef(30.0f, 0, 1.0f, 0);
        gluDisk(baseObj, 0, 1, 20, 20); glPopMatrix();

        // Leaf
        drawLeaf(1.5, 0, 0, 0, 1);
        drawLeaf(1.5, 30.0f, 0, 0, 2);
        drawLeaf(1.5, -30.0f, 0, 0, 3);
        drawLeaf(1.5, 0, 0, 30.0f, 4);
        drawLeaf(1.5, 0, 0, -30.0f, 5);
        drawLeaf(1.5, 24.0f, 0, 30.0f, 6);
        drawLeaf(1.5, -24.0f, 0, -30.0f, 7);
        drawLeaf(1.5, 24.0f, 0, -30.0f, 8);
        drawLeaf(1.5, -24.0f, 0, 30.0f, 9);
}


void drawSign(string text) {
        glPushMatrix(); glColor3ub(50,
        50, 50);
        glTranslatef(0.0f, -0.5f, 0.0f); glScalef(0.3f,
        3.5f, 0.3f); glutSolidCube(1);   glPopMatrix();
        glPushMatrix(); glColor3ub(170, 140, 65);
        glTranslatef(0.0f, 1.0f, 0.05f); glScalef(1.75f,
        0.5f, 0.3f); glutSolidCube(1);   glPopMatrix();
        glPushMatrix();
        glColor3ub(0, 0, 0); glLineWidth(3);
        glTranslatef(-0.8f, 0.85f, 0.22f); glScalef(0.003f, 0.003f, 0.3f);
        for (size_t i = 0; i < text.length(); i++) glutStrokeCharacter(font,
                text[i]);
        glPopMatrix();
}


void keyDown(unsigned char key, int x, int y) {
        if (key == 27) exit(0);

        if (key == 32) {
                if (!is_jumping) { is_jumping = true;
                        on_ground = false;
                }
        }

        if (key == 'a')                         // Look left
                is_left = true;
        else if (key == 'd')                    // Look right
                is_right = true;
        else if (key == 'w')                    // Move forward
                is_forward = true;
        else if (key == 's')                    // Move backward
                is_backward = true;

        if (bySpongebobHouse) {
                if (key == 13 && funds >= 1000) { funds  -=
                        1000; spongebob_sold = true;
                }
        }
        if (byPatrickHouse) {
                if (key == 13 && funds >= 7) { funds -= 7;
                        patrick_sold = true;
                }
        }
        if (bySquidwardHouse) {
                if (key == 13 && funds >= 1000000) { funds  -=
                        1000000; squidward_sold = true;
                }
        }
}


void keyUp(unsigned char key, int x, int y) {
        if (key == 'a')                         // Stop moving left is_left =
                false;
        else if (key == 'd')                    // Stop moving right is_right
                = false;
        else if (key == 'w')                    // Stop moving forward
                is_forward = false;
        else if (key == 's')                    // Stop moving backward
                is_backward = false;
}


void specialDown(int key, int x, int y) {
        switch (key) {
        case GLUT_KEY_LEFT:
                is_look_left = true;                    // Look left
                break;
        case GLUT_KEY_RIGHT:
                is_look_right = true;                   // Look right
                break;
        case GLUT_KEY_UP:
                is_look_up = true;                      // Look up
                break;
        case GLUT_KEY_DOWN:
                is_look_down = true;                    // Look down
                break;

        }
}


void specialUp(int key, int x, int y) {
        switch (key) {
        case GLUT_KEY_LEFT:
                is_look_left = false;                   // Stop looking left
                break;
        case GLUT_KEY_RIGHT:
                is_look_right = false;  // Stop looking right
                break;
        case GLUT_KEY_UP:
                is_look_up = false;                     // Stop looking up
                break;
        case GLUT_KEY_DOWN:
                is_look_down = false;                   // Stop looking down
                break;
        }
}
```

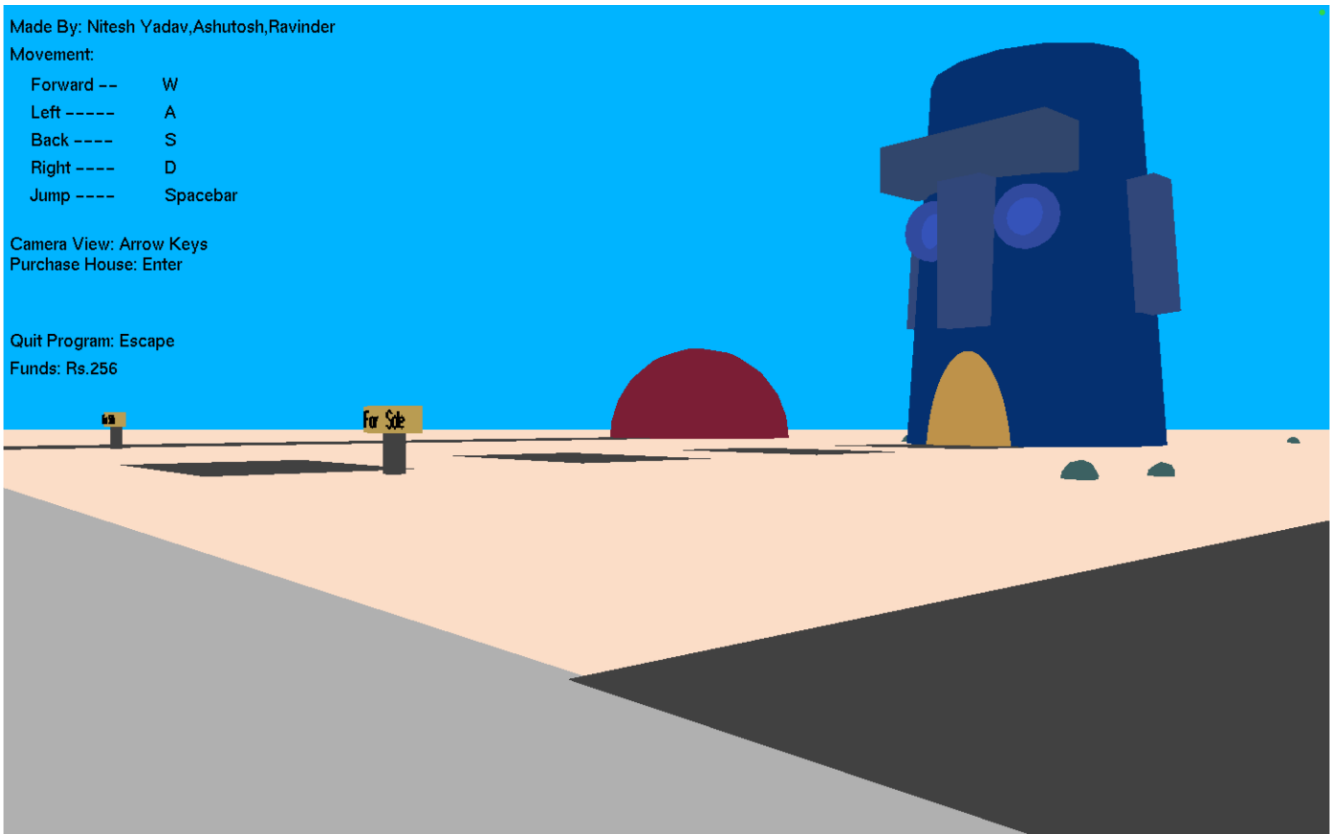
```
void jump() {
        if (jump_energy > 0) {
                y += jump_force * jump_energy; jump_energy -=
                1;
        }
}


void cameraHandler() {
        if (is_look_left) { angle -=
                x_sens; lx = sin(angle); lz
                = -cos(angle);
        }
        else if (is_look_right) { angle += x_sens;
                lx = sin(angle); lz = -
                cos(angle);
        }

        if (is_look_up) y +=
                y_sens;
        else if (is_look_down) y -=
                y_sens;

        if (is_forward) {
                x += lx * speed; z += lz *
                speed;
        }
        else if (is_backward) { x -= lx *
                speed;
                z -= lz * speed;
        }

        if (is_left) {
                x += lz * speed; z += -lx *
                speed;
        }
        else if (is_right) { x -= lz *
                speed; z -= -lx * speed;

        }

        if(is_jumping) jump();

        // Gravity
        if(!on_ground) y -=
                speed;

        if (y <= 1.0f) { y = 1.0f;
                is_jumping = false; on_ground =
                true; jump_energy = energy_max;
        }
}


void changeSize(int w, int h) {
        // Prevent division by zero when the window is too short
        if (h == 0)
                h = 1;

        float ratio = float(w * 1.0 / h);

        // Use the projection matrix glMatrixMode(GL_PROJECTION);

        // Reset matrix
        glLoadIdentity();

        // Set the viewport to be the entire window glViewport(0, 0, w, h);

        // Set the correct perspective gluPerspective(45.0f, ratio, 0.1f,
        100.0f);

        // Get back to the modelview glMatrixMode(GL_MODELVIEW);
}


// Draws text displayed on screen
void renderText() {
        glDisable(GL_TEXTURE_2D);
        glMatrixMode(GL_PROJECTION);
        glPushMatrix(); glLoadIdentity();
        gluOrtho2D(0.0, windowWidth, 0.0, windowHeight); glMatrixMode(GL_MODELVIEW);
        glPushMatrix();
        glLoadIdentity();
        drawText("Made By: Nitesh Yadav", 5, 580, blackColor, helvetica);
        drawText("Movement:", 5, 560, blackColor, helvetica); drawText("Forward -- ", 21,
        538, blackColor, helvetica); drawText("W", 120, 538, blackColor, helvetica);
        drawText("Left ----- ", 21, 518, blackColor, helvetica); drawText("A", 122, 518,
        blackColor, helvetica); drawText("Back ---- ", 21, 498, blackColor, helvetica);
        drawText("S", 122, 498, blackColor, helvetica); drawText("Right ---- ", 21, 478,
        blackColor, helvetica); drawText("D", 122, 478, blackColor, helvetica);
        drawText("Jump ---- ", 20, 458, blackColor, helvetica); drawText("Spacebar", 122,
        458, blackColor, helvetica);
        drawText("Camera View: Arrow Keys", 5, 423, blackColor, helvetica);
        drawText("Purchase House: Enter", 5, 408, blackColor, helvetica);
        drawText("Quit Program: Escape", 5, 353, blackColor, helvetica);
        drawText("Funds: Rs." + to_string(funds), 5, 333, blackColor, helvetica);

        if (byPatrickHouse) {
                drawText("Home for sale: Patrick's Home", 10, 140, whiteColor, roman);
                drawText("Estimate: Rs 10.00", 10, 115, whiteColor,
roman);
                drawText("Address: 120 Conch Street", 10, 90, whiteColor,
roman);
                drawText("Realtor: Mr. Krabs", 10, 65, whiteColor, roman); drawText("Contact info:
                9976-417-816-KRAB", 10, 40,
    whiteColor, roman);
                        drawText("Comments: It's a rock.", 10, 15, whiteColor,
roman);
                drawSaleInfoSign();
        }

        if (bySquidwardHouse) {
                drawText("Home for sale: Squidward's Home", 10, 140, whiteColor, roman);
                drawText("Estimate: Rs 1,000,000.00", 10, 115, whiteColor,
roman);
                drawText("Address: 122 Conch Street", 10, 90, whiteColor,
roman);
                drawText("Realtor: Mr. Krabs", 10, 65, whiteColor, roman); drawText("Contact info:
                9987-417-816-KRAB", 10, 40,
whiteColor, roman);
                drawText("Comments: Clarinetists only.", 10, 15, whiteColor, roman);
                drawSaleInfoSign();
        }

        if (bySpongebobHouse) {
                drawText("Home for sale: Spongebob's Home", 10, 140, whiteColor, roman);
                drawText("Estimate: Rs 1,000.00", 10, 115, whiteColor,
roman);
                drawText("Address: 124 Conch Street", 10, 90, whiteColor,
roman);
                drawText("Realtor: Mr. Krabs", 10, 65, whiteColor, roman); drawText("Contact info:
                9911-417-816-KRAB", 10, 40,
whiteColor, roman);
                drawText("Comments: Beware Gary the Snail.", 10, 15, whiteColor, roman);
                drawSaleInfoSign();
        }

        glMatrixMode(GL_PROJECTION);
        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();
        glEnable(GL_TEXTURE_2D);
}


// Draws the advertisement info for each house
void drawSaleInfoSign() { glPolygonMode(GL_FRONT_AND_BACK,
        GL_FILL); glBegin(GL_POLYGON);
        glColor3d(1.00, 0.0, 0.0);
        glVertex2f(GLfloat(5), GLfloat(5));
        glVertex2f(GLfloat(5), GLfloat(160));
        glVertex2f(GLfloat(380), GLfloat(160));
        glVertex2f(GLfloat(380), GLfloat(5)); glEnd();
}


// Text function
void drawText(std::string text, int x, int y, int rgb[3], void* font) {
        draw_text = text; glColor3ub(rgb[0], rgb[1],
        rgb[2]); glRasterPos2i(x, y);
        for (size_t i = 0; i < draw_text.length(); i++) glutBitmapCharacter(font, draw_text[i]);
}


// Draws a single leaf given information from the plantMatrix void drawPlantLeaf(float size,
float rotx, float roty, float rotz, int parent_index, int index) {
        float   r   =   plantMatrix[parent_index][0][0];   float   g   =
        plantMatrix[parent_index][0][1];        float        b        =
        plantMatrix[parent_index][0][2];
        float   offset   =   plantMatrix[parent_index][index][0];   float   r_rand   =
        plantMatrix[parent_index][index][1];        float        g_rand        =
        plantMatrix[parent_index][index][2];        float        b_rand        =
        plantMatrix[parent_index][index][3];

        glPushMatrix();
        glColor3ub(GLubyte(r - r_rand), GLubyte(g - g_rand), GLubyte(b
- b_rand));
        glTranslatef(0.0f, 0.0f, 0.0f); glRotatef(rotx + offset, 1.0f, 0.0f,
        0.0f); glRotatef(roty + offset, 0.0f, 1.0f, 0.0f); glRotatef(rotz +
        offset, 0.0f, 0.0f, 1.0f);
        glScalef(.7f, 5.0f, 0.7f);
        glutSolidSphere(size, 5, 5); glPopMatrix();
}


// Draws the various leaves of a plant
void drawPlant(float size, int index) { drawPlantLeaf(size, 0, 0, 0,
        index, 2);
        drawPlantLeaf(size, 30.0f, 0, 0, index, 3);
        drawPlantLeaf(size, -30.0f, 0, 0, index, 4);
        drawPlantLeaf(size, 0, 0, 30.0f, index, 5);
        drawPlantLeaf(size, 0, 0, -30.0f, index, 6);
        drawPlantLeaf(size, 24.0f, 0, 30.0f, index, 7);
        drawPlantLeaf(size, -24.0f, 0, -30.0f, index, 8);
        drawPlantLeaf(size, 24.0f, 0, -30.0f, index, 9);
        drawPlantLeaf(size, -24.0f, 0, 30.0f, index, 10);
}


// Creates and stores all the elements of plants into the plantMatrix
void createPlant(int seed) {
        vector<vector<float>> plant;
        vector<float> plant_color;
        std::srand(seed);
        float r = float(rand() % 255); float g =
        float(rand() % 255); float b = float(rand() %
        255); plant_color.push_back(r);
        plant_color.push_back(g);
        plant_color.push_back(b);
        plant.push_back(plant_color);
        vector<float> plant_pos;
        float x = float((rand() % 200) - 100); float z =
        float((rand() % 50) + 5); plant_pos.push_back(x);
        plant_pos.push_back(z); plant.push_back(plant_pos);

        for (size_t i = 0; i < 9; i++) { std::srand(i * seed);
                vector<float>  plant_entity; float offset =
                float(rand() % 5);
                plant_entity.push_back(offset);
                float r_rand = float(rand() % 25); float g_rand =
                float(rand() % 25); float b_rand = float(rand() %
                25); plant_entity.push_back(r_rand);
                plant_entity.push_back(g_rand);
                plant_entity.push_back(b_rand);
                plant.push_back(plant_entity);
        }

        plantMatrix.push_back(plant);
}


// Draws a super cool rock
void drawRock(GLfloat angle, double size, GLint smoothness) { glPushMatrix();
        glColor3ub(47, 79, 79);
        glRotatef(angle, 1.0f, 0, 0); glutSolidSphere(size, smoothness,
        smoothness); glPopMatrix();
}


// Checks if a the user is near a specifc house based on X and Z ranges
void checkByHouses() { byPatrickHouse =
        false; bySquidwardHouse = false;
        bySpongebobHouse = false;

        if (x > -20 && x < -10 && z > -35 && z < -25) byPatrickHouse =
                true;
        if (x > 0 && x < 10 && z > -35 && z < -25) bySquidwardHouse =
                true;
        if (x > 20 && x < 30 && z > -35 && z < -25) bySpongebobHouse =
                true;
}


// Checks if a the user is near a specifc wad of cash based on X and Z ranges
void checkMoneyCollision() {
        // Stack 1
        if (x >= -7 && x <= -5 && z >= -4 && z <= -3) {
                if (!stack1_collision) funds += 250;

                stack1_collision = true;
        }
        // Stack 2
        if (x >= -17 && x <= -14 && z >= -16 && z <= -14) {
                if (!stack2_collision) funds += 250;

                stack2_collision = true;
        }
        // Stack 3
        if (x >= -23 && x <= -21 && z >= -41 && z <= -39) {
                if (!stack3_collision)
```

```
                    funds += 250;

                stack3_collision = true;
        }
        // Stack 4
        if (x >= -43 && x <= -41 && z >= -37 && z <= -35) {
                if (!stack4_collision) funds += 250;

                stack4_collision = true;
        }
        // Secret Stack
        if (x >= 2 && x <= 4 && z >= -41 && z <= -39) {
                if (!secretstack_collision) funds +=
                        1000000;
                secretstack_collision = true;

        }
}
```

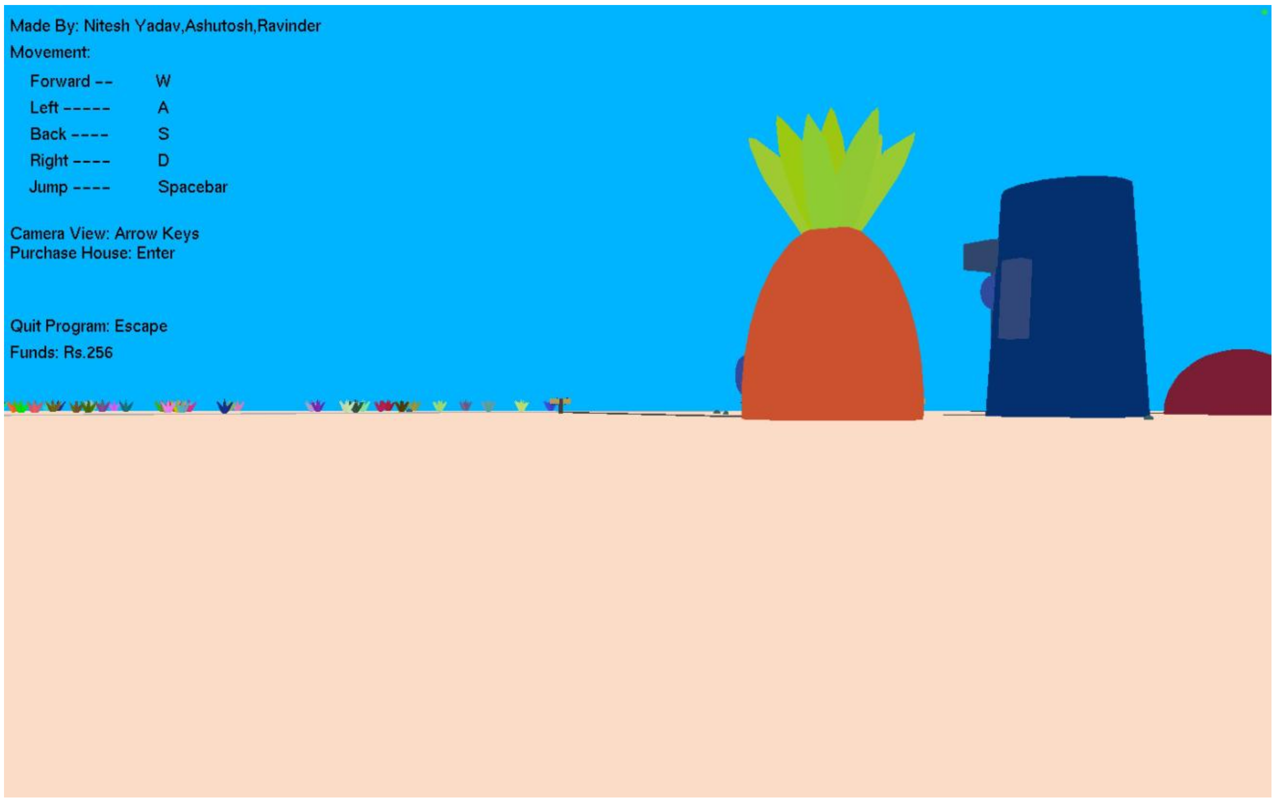
4) Output/ Screen shots:



```
Made By: Nitesh Yadav,Ashutosh,Ravinder
Movement:
   Forward --     W
   Left -----     A
   Back ----      S
   Right ----     D
   Jump ----      Spacebar

Camera View: Arrow Keys
Purchase House: Enter


Quit Program: Escape
Funds: Rs.6
```



```
Made By: Nitesh Yadav,Ashutosh,Ravinder
Movement:
    Forward --       W
    Left -----       A
    Back ----        S
    Right ----       D
    Jump ----        Spacebar

Camera View: Arrow Keys
Purchase House: Enter


Quit Program: Escape
Funds: Rs.256
```



```
Made By: Nitesh Yadav,Ashutosh,Ravinder
Movement:
    Forward --       W
    Left -----       A
    Back ----        S
    Right ----       D
    Jump ----        Spacebar

Camera View: Arrow Keys
Purchase House: Enter


Quit Program: Escape
Funds: Rs.256
```