Computer Graphics (UCS505)

Project on

3D Helicopter Simulator

Submitted By

Danveer	**102203272**
Samarth Singh Adhikari	**102203303**

3C22

B.E. Third Year – COE

Submitted To:

Ms. Jasmine Kaur

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

Table of Contents

Introduction to Project

The 3D Helicopter Simulator Project is a computer graphics endeavor that uses OpenGL and GLUT libraries to render a 3D environment where a helicopter can be manipulated and observed in flight. The project incorporates various elements such as terrain, buildings, a helipad, and the helicopter itself, along with user-controlled viewing angles and movements.

Key Features:

1. Helicopter Model: The project includes a detailed 3D model of a helicopter, complete with a rotating rotor, tail, landing legs, and a tail fan.
2. Landscape: The scene features a terrain with grass, buildings, and a helipad for takeoff and landing.
3. Dynamic Movement: Users can control the helicopter's movement, including forward tilt, left tilt, right tilt, and upward movement.
4. Viewing Options: Different viewing perspectives are available, such as ground view, pilot view, landscape view, and follow view.
5. Lighting and Shading: The project incorporates lighting effects and shading to enhance the realism of the scene.

Functionality:

- Users can start and stop the helicopter's motion.
- Navigation controls allow users to change the viewing angle and position.
- The simulation provides a visually immersive experience of flying a helicopter in a virtual environment.

Technical Details:
The code is written in C++ and utilizes OpenGL and GLUT libraries for graphics rendering and user interface. It employs transformations, lighting models, geometric primitives, and menu-driven interactions to create the simulation.

Computer Graphics Concepts Used

The project harnesses several fundamental Computer Graphics concepts to create a dynamic and immersive visual experience. One of the core concepts utilized is Graphics Primitives, where color manipulation plays a crucial role. Through the glColor functions, the code sets colors for different objects, enhancing their visual appeal and distinguishing them within the scene. This enables the creation of a vibrant and visually engaging environment, essential for effective graphics rendering.

Another key concept applied in the code is 3-D Geometric Transformations, particularly translation and rotation. Translation is employed to move objects and manipulate the viewpoint within the 3D space. This functionality adds dynamism to the scene, allowing for fluid movements and transitions. Rotation, on the other hand, brings objects to life by enabling them to spin and revolve, as seen with the helicopter and various other elements within the scene. These transformations collectively contribute to creating a dynamic and interactive 3D environment.

Viewing & Clipping in 2-D and Three Dimensional Viewing & Clipping are pivotal concepts for rendering scenes realistically. The code implements Window to Viewport transformation using gluLookAt and related transformations, facilitating the proper alignment and display of objects within the viewport. Additionally, 3-D Viewing, Projections, and Perspective projections are implemented through gluPerspective and gluLookAt, enhancing the immersive nature of the graphics by creating a sense of depth and perspective. These techniques are essential for creating realistic and immersive visual experiences in 3D graphics applications.

Lastly, Illumination Models & Surface Rendering concepts are employed to enhance the realism and visual appeal of the scene. Lighting and illumination models are implemented using OpenGL's lighting functions, allowing for the realistic portrayal of light sources and their interactions with surfaces. This adds depth, realism, and visual interest to the scene, making it more engaging and visually appealing to the viewer.

In conclusion the following concepts were utilized:

1. Color Manipulation
2. 3-D Geometric Transformations
3. Window to Viewport Transformation
4. 3-D Viewing, Projections, Parallel and Perspective Projections
5. Lighting and Illumination Models

User Defined Functions

1. **draw_heli():** This function is responsible for drawing the entire helicopter by calling other functions to draw its various components like the body, rotor, tail, legs, and tail fan. It sets the material properties for the helicopter's appearance.
2. **draw_body():** Draws the main body of the helicopter using a scaled sphere to represent its shape.
3. **draw_rotor():** Draws the rotor blades of the helicopter. It creates a central rod and four rotor blades rotating around it, achieving a realistic rotor effect.
4. **draw_tail():** Draws the tail structure of the helicopter using line segments to outline its shape.
5. **draw_leg():** Draws the supporting legs of the helicopter using lines to represent their structure and position.
6. **draw_tail_fan():** Draws the fan at the tail end of the helicopter, which is a smaller version of the rotor blades specifically designed for the tail.
7. **draw_grass():** Draws the ground texture using green lines to simulate a grassy surface.
8. **draw_sky():** Draws the sky using polygons to create a background. It uses different shades of blue to represent the sky's color gradient.
9. **draw_helipad():** Draws the landing pad for the helicopter. It includes a torus shape for the pad itself and additional lines to mark its boundaries.
10. **draw_building():** Draws buildings in the scene by calling the draw_house() function. It creates multiple instances of houses at different positions to simulate a cityscape.
11. **draw_house():** Draws a single house structure, including its base and roof using cubes and cylinders.
12. **polygon(int a, int b, int c, int d):** Draws a polygon based on the indices of four vertices. This function is used to draw the sky's polygons.
13. **fly_effect():** Implements special effects for helicopter movement. It includes tilting effects when the helicopter moves forward or turns.
14. **light():** Sets up lighting and material properties for the scene, including ambient, diffuse, and specular lighting components.
15. **init():** Initializes the OpenGL environment, enabling features like depth testing, lighting, and the creation of the main window.
16. **display():** Handles the display of the entire scene. It includes rendering the ground, sky, buildings, helicopter, and other elements based on the current viewer's perspective.
17. **reshape(int w, int h):** Handles window resizing by adjusting the viewport and projection matrix to maintain the correct aspect ratio.
18. **keyboard(unsigned char key, int x, int y):** Processes keyboard inputs for camera movement (changing viewer position) and helicopter rotation.
19. **Specialkey(int key, int x, int y):** Processes special key inputs for helicopter movement (up, down, left, right) and activates tilt effects.

20. **idle():** Manages idle state actions, such as rotor animation, helicopter movement, and scene updates.
21. **Releasekey(int key, int x, int y):** Handles key release events to stop specific actions initiated by key presses.
22. **my_menu(int id):** Manages menu options for starting, stopping, and exiting the simulation, as well as choosing different viewing perspectives.
23. **view_menu(int id):** Handles different viewing options, allowing the user to switch between ground, pilot, landscape, and follow views.
24. **stop_helicopter_sound( ):** Stops any currently playing helicopter sound by calling PlaySound with NULL, and resets the sound_playing flag.
25. **init( ):** Initializes OpenGL settings by enabling depth testing, lighting, and Light 0 for the 3D scene rendering.
26. **light( ):** Configures the lighting environment, including global ambient light, light source properties (position, ambient, diffuse, specular), and material shininess.
27. **display( ):** The main rendering function that sets the camera view, applies lighting, and draws all scene elements such as grass, sky, helicopter, helipad, and buildings.

Code

```
#include <stdlib.h>
#include <gl/glut.h>
#include <math.h>
#include <string.h>
#include <windows.h>
#include <mmsystem.h>

GLint sub_menu[4];
float angle_rad;
//Functions
void draw_heli();
void draw_body();
void draw_rotor();
void draw_tail();
void draw_leg();
void draw_tail_fan();
void draw_grass();
void draw_helipad();
void draw_building();
void draw_sky();
void polygon(int a, int b, int c, int d);
void draw_house();
void fly_effect();
void my_menu(int id);
void view_menu(int id);
void play_helicopter_sound();

//Lighting & Shading Constants
GLfloat global_ambient[] = { 0.1,0.1,0.1,1.0 };
GLfloat light0_pos[] = { 2.0,6.0,3.0,0.0 };
GLfloat ambient0[] = { 1.0,1.0,1.0,1.0 };
GLfloat diffuse0[] = { 1.0,1.0,1.0,1.0 };
GLfloat specular0[] = { 1.0,1.0,1.0,1.0 };
GLfloat material_specular[] = { 0.8,0.8,0.8,1.0 };
GLfloat baseAmbientLight[] = { 0.2,0.2,0.2,1.0 };
GLfloat baseDiffuseLight[3][4] = { {0.54,0.17,0.89,1.0},{1.0,0.0,0.0,1.0},{1.0,0.83,0.61,1.0} };
GLfloat roofAmbientLight[] = { 0.2,0.2,0.2,1.0 };
GLfloat roofDiffuseLight[3][4] = { {1.0,0.89,0.88,1.0},{0.61,1.0,1.0,1.0},{1.0,0.55,0.0,1.0} };
GLfloat helicopterAmbientLight[] = { 0.2,0.2,0.2,1.0 };
GLfloat helicopterDiffuseLight[3][4] = {
{0.50,0.85,0.0,1.0},{0.74,0.72,0.42,1.0},{0.39,0.72,1.0,1.0} };
//Movement Coordinates
GLfloat myx = 0.0, myy = 0.0, myz = 0.0, rotation = 0.0, rotor_angle = 0.0, heli_tilt = 0.0;
//Coordinates of Sky Cube
GLfloat vertices[8][3] = { {-500.0,-50.0,-500.0},{500.0,-50.0,-500.0},{500.0,500.0,-500.0},{-
500.0,500.0,-500.0},{-500.0,-50.0,500.0},{500.0,-50.0,500.0},{500.0,500.0,500.0},{-
500.0,500.0,500.0} };
```

```
//Color of Sky Cube
GLdouble colors[8][4] = {
{0.89,1.0,1.0,0.5},{0.89,1.0,1.0,0.5},{0.4,0.88,1.0,0.5},{0.4,0.89,1.0,0.5},{0.89,1.0,1.0,0.5},{0.89,1.0,1
.0,0.5},{0.4,0.88,1.0,0.5},{0.4,0.88,1.0,0.5} };
//Flags
int start = 0, stop = 0, forward_tilt = 0, left_tilt = 0, right_tilt = 0, bc = 0, hc = 0;
int sound_playing = 0; // Flag to track if sound is currently playing
DWORD last_sound_time = 0; // Time when sound was last played

//Viewer
GLfloat viewer[] = { 0.0,60.0,-5.0 };
//Viewer Coordinates
GLfloat at[] = { 0.0,0.0,0.0 };

// Function to play helicopter sound
void play_helicopter_sound() {
	// Only play sound if in start state and not already playing or if sound timeout expired
	if (start && (!sound_playing || (GetTickCount() - last_sound_time > 18000))) {
		PlaySound(TEXT("sounds/helicopter-sound-effect-241421.wav"), NULL,
SND_FILENAME | SND_ASYNC);
		sound_playing = 1;
		last_sound_time = GetTickCount();
	}
}

// Function to stop helicopter sound
void stop_helicopter_sound() {
	PlaySound(NULL, NULL, 0);
	sound_playing = 0;
}

void init(void) {
	glEnable(GL_DEPTH_TEST);
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
}
void light(void) {
	glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient);
	glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
	glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);
	glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
	glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
	glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
	glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, material_specular);
	glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 128.0);
}
void display(void) {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
	glLoadIdentity();
	gluLookAt(viewer[0], viewer[1], viewer[2], at[0], at[1], at[2], 0.0, 1.0, 0.0);
	light();
	draw_grass();
	glTranslatef(myx + 2, myy + 2, myz + 2);
	draw_sky();
	glTranslatef(-myx + 2, -myy + 2, -myz + 2);
	draw_heli();
	draw_helipad();
	draw_building();
	glFlush();
	glutSwapBuffers();
}
void draw_grass() {
	glPushMatrix();
	glDisable(GL_LIGHTING);
	glColor3f(0.13333, 0.545098, 0.13333);
	double i;
	glBegin(GL_LINES);
	for (i = -2500.0; i <= 2500; i += 0.5) {
		glVertex3d(-2500, -11, i);
		glVertex3d(2500, -11, i);
		glVertex3d(i, -11, -2500);
		glVertex3d(i, -11, 2500);
	}
	glEnd();
	glEnable(GL_LIGHTING);
	glPopMatrix();
}
void polygon(int a, int b, int c, int d) {
	glPushMatrix();
	glDisable(GL_LIGHTING);
	glBegin(GL_POLYGON);
	glColor4dv(colors[a]);
	glVertex3fv(vertices[a]);
	glColor4dv(colors[b]);
	glVertex3fv(vertices[b]);
	glColor4dv(colors[c]);
	glVertex3fv(vertices[c]);
	glColor4dv(colors[d]);
	glVertex3fv(vertices[d]);
	glEnd();
	glEnable(GL_LIGHTING);
	glPopMatrix();
}
void draw_sky() {
	glPushMatrix();
	polygon(0, 3, 2, 1);
```

```
        polygon(2, 3, 7, 6);
        polygon(0, 4, 7, 3);
        polygon(1, 2, 6, 5);
        polygon(4, 5, 6, 7);
        glPopMatrix();
}
void fly_effect() {
        if (forward_tilt) {
                if (cos(rotation * 3.14 / 180) > cos(45 * 3.14 / 180) || cos(rotation * 3.14 /
                        180) < cos(135 * 3.14 / 180)) {
                        glRotatef(10, cos(rotation * 3.14 / 180), 0.0, 0.0);
                }
                else {
                        glRotatef(-10, 0.0, 0.0, sin(rotation * 3.14 / 180));
                }
        }
        if (right_tilt) {
                heli_tilt = 30;
        }
        else if (left_tilt) {
                heli_tilt = -30;
        }
}
void draw_heli() {
        glPushMatrix();
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, helicopterAmbientLight);
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, helicopterDiffuseLight[hc]);
        glTranslatef(myx, myy, myz);
        fly_effect();
        glRotatef(rotation, 0.0, 1.0, 0.0);
        glTranslatef(-myx, -myy, -myz);
        glTranslatef(myx, myy, myz);
        glRotatef(heli_tilt, 0.0, 0.0, 1.0);
        draw_body();
        draw_rotor();
        draw_tail();
        draw_leg();
        draw_tail_fan();
        glPopMatrix();
}
void draw_body() {
        glPushMatrix();
        glScalef(1.0, 1.0, 2.0);
        glutSolidSphere(5.0, 32, 32);
        glScalef(1.0, 1.0, 0.5);
        glPopMatrix();
}
void draw_rotor() {
```

```
	glPushMatrix();
	glDisable(GL_LIGHTING);
	glColor4f(0.75, 0.75, 0.75, 1.0);//silver
	glLineWidth(5.0);
	glBegin(GL_LINES);
	glVertex3f(0.0, 0.0, 0.0);
	glVertex3f(0.0, 10.0, 0.0);
	glEnd();
	glScalef(2.0, 1.0, 2.0);
	for (int i = 0; i < 4; i++) {
		glPushMatrix();
		glRotatef(rotor_angle, 0.0, 1.0, 0.0);
		glTranslatef(0.0, 10.0, 0.0);
		glRotatef(30.0, 1.0, 0.0, 0.0);
		glTranslatef(0.0, -10.0, 0.0);
		glBegin(GL_QUADS);
		glVertex3f(0.0, 10.0, -0.5);
		glVertex3f(10.0, 10.0, -1.0);
		glVertex3f(10.0, 10.0, 1.0);
		glVertex3f(0.0, 10.0, 0.5);
		glEnd();
		glPopMatrix();
		glRotatef(90.0, 0.0, 1.0, 0.0);
	}
	glEnable(GL_LIGHTING);
	glPopMatrix();
}
void draw_tail() {
	glPushMatrix();
	glDisable(GL_LIGHTING);
	glColor4f(0.4, 0.4, 0.4, 1.0);
	glBegin(GL_LINE_LOOP);
	glVertex3f(1.2, -2.0, -9.0);
	glVertex3f(1.2, 2.0, -9.0);
	glVertex3f(0.2, 0.2, -29.0);
	glVertex3f(0.2, -0.2, -29.0);
	glEnd();
	glBegin(GL_LINE_LOOP);
	glVertex3f(-1.2, -2.0, -9.0);
	glVertex3f(-1.2, 2.0, -9.0);
	glVertex3f(-0.2, 0.2, -29.0);
	glVertex3f(-0.2, -0.2, -29.0);
	glEnd();
	glBegin(GL_LINE_LOOP);
	glVertex3f(1.2, 2.0, -9.0);
	glVertex3f(-1.2, 2.0, -9.0);
	glVertex3f(-0.2, 0.2, -29.0);
	glVertex3f(0.2, 0.2, -29.0);
```

```
        glEnd();
        glBegin(GL_LINE_LOOP);
        glVertex3f(1.2, -2.0, -9.0);
        glVertex3f(-1.2, -2.0, -9.0);
        glVertex3f(-0.2, -0.2, -29.0);
        glVertex3f(0.2, -0.2, -29.0);
        glEnd();
        glEnable(GL_LIGHTING);
        glPopMatrix();
}
void draw_leg() {
        glPushMatrix();
        glDisable(GL_LIGHTING);
        glColor4f(0.7, 0.7, 0.7, 1.0);
        glBegin(GL_LINES);
        glVertex3f(5, -7, 7);
        glVertex3f(5, -7, -7);
        glVertex3f(5, -7, 7);
        glVertex3f(5, -6, 8);
        glVertex3f(5, -7, -7);
        glVertex3f(5, -6, -8);
        glVertex3f(0, 0.0, 5);
        glVertex3f(5, -7.0, 5);
        glVertex3f(0, 0.0, -5);
        glVertex3f(5, -7.0, -5);
        glVertex3f(-5, -7, 7);
        glVertex3f(-5, -7, -7);
        glVertex3f(-5, -7, 7);
        glVertex3f(-5, -6, 8);
        glVertex3f(-5, -7, -7);
        glVertex3f(-5, -6, -8);
        glVertex3f(0, 0.0, 5);
        glVertex3f(-5, -7.0, 5);
        glVertex3f(0, 0.0, -5);
        glVertex3f(-5, -7.0, -5);
        glEnd();
        glEnable(GL_LIGHTING);
        glPopMatrix();
}
void draw_tail_fan() {
        glPushMatrix();
        glTranslatef(0.0, 0.0, -28.0);
        glRotatef(-90.0, 0.0, 0.0, 1.0);
        glScalef(0.15, 0.11, 0.15);
        draw_rotor();
        glPopMatrix();
}
void draw_helipad() {
```

```
	glPushMatrix();
	glDisable(GL_LIGHTING);
	glColor3f(0.125, 0.125, 0.125);
	glTranslatef(0.0, -9.0, -1.0);
	glRotatef(-90.0, 1.0, 0.0, 0.0);
	glScalef(1.0, 1.0, 0.1);
	glutSolidTorus(13.0, 13.0, 100, 100);
	glScalef(1.0, 1.0, 1.0);
	glLineWidth(25.0);
	glColor3f(1.0, 1.0, 1.0);
	glBegin(GL_QUADS);
	glVertex3f(-9.5, -1.5, 16);
	glVertex3f(-9.5, 1.5, 16);
	glVertex3f(9.5, 1.5, 16);
	glVertex3f(9.5, -1.5, 16);
	glVertex3f(9.5, 11.5, 16);
	glVertex3f(9.5, -11.5, 16);
	glVertex3f(11.5, -11.5, 16);
	glVertex3f(11.5, 11.5, 16);
	glVertex3f(-9.5, 11.5, 16);
	glVertex3f(-9.5, -11.5, 16);
	glVertex3f(-11.5, -11.5, 16);
	glVertex3f(-11.5, 11.5, 16);
	glEnd();
	glEnable(GL_LIGHTING);
	glPopMatrix();
}
void draw_building() {
	glPushMatrix();
	float x, z;
	for (z = 100; z <= 1000; z += 150)
		for (x = -400; x <= 400; x += 150) {
			glPushMatrix();
			glTranslatef(x, 0.0, z);
			draw_house();
			glTranslatef(-x, 0.0, -z);
			glTranslatef(-x, 0.0, z);
			draw_house();
			glTranslatef(x, 0.0, -z);
			glPopMatrix();
		}
	glPushMatrix();
	glTranslatef(0.0, 0.0, 1010.0);
	draw_helipad();
	glTranslatef(0.0, 0.0, -1010.0);
	glPopMatrix();
	glPopMatrix();
}
```

```
void draw_house() {
	glPushMatrix();
	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, baseAmbientLight);
	glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, baseDiffuseLight[bc]);
	glTranslatef(0.0, 5.0, 0.0);
	glutSolidCube(30.0);
	glTranslatef(0.0, -5.0, 0.0);
	glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, roofAmbientLight);
	glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, roofDiffuseLight[bc]);
	glTranslatef(0.0, 15.0, 0.0);
	glRotatef(-90.0, 1.0, 0.0, 0.0);
	gluCylinder(gluNewQuadric(), 25.0, 0.0, 25.0, 32, 32);
	glTranslatef(0.0, -15.0, 0.0);
	glPopMatrix();
}

void reshape(int w, int h) {
	glViewport(0, 0, (GLsizei)w, (GLsizei)h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(100, (GLfloat)w / (GLfloat)h, 0.1, 1500.0);
	glMatrixMode(GL_MODELVIEW);
	glutPostRedisplay();
}
void keyboard(unsigned char key, int x, int y) {
	//Ascend
	if (key == 'w') {
		if (myy > 0.0)myy++; if (viewer[1] >
			0)viewer[1] += 1.0; at[1] += 1.0;
	}

	//Descend
	if (key == 's') {
		if (myy > 0.0)myy--; if (viewer[1] >
			0)viewer[1] -= 1.0; at[1] -= 1.0;
	}
	//X-axis controls
	if (key == 'z') { viewer[0] += 1.0; }
	if (key == 'x') { viewer[0] -= 1.0; }
	//Y-axis controls
	if (key == 'c') { viewer[1] += 1.0; }
	if (key == 'v') { viewer[1] -= 1.0; }
	//Z-axis controls
	if (key == 'b') { viewer[2] += 1.0; }
	if (key == 'n') { viewer[2] -= 1.0; }

	//Left Rotation
	if (key == 'a') {
```

```
		rotation++;
		angle_rad = 1 * 3.14 / 180;
		float Xp = myx, Zp = myz, X = viewer[0], Z = viewer[2];
		viewer[2] = (Z - Zp) * cos(angle_rad) - (X - Xp) * sin(angle_rad) + Zp;
		viewer[0] = (X - Xp) * cos(angle_rad) + (Z - Zp) * sin(angle_rad) + Xp;
	}

	//Right Rotataion
	if (key == 'd') {
		rotation--;
		angle_rad = -1 * 3.14 / 180;
		float Xp = myx, Zp = myz, X = viewer[0], Z = viewer[2];
		viewer[2] = (Z - Zp) * cos(angle_rad) - (X - Xp) * sin(angle_rad) + Zp;
		viewer[0] = (X - Xp) * cos(angle_rad) + (Z - Zp) * sin(angle_rad) + Xp;
	}
}
void Specialkey(int key, int x, int y) {
	if (start) {
		// Sound is now managed by play_helicopter_sound() function
		play_helicopter_sound();

		angle_rad = rotation * 3.14 / 180.0;
		if (key == GLUT_KEY_UP) {
			myz += cos(angle_rad); viewer[2] +=
				cos(angle_rad); at[2] += cos(angle_rad); forward_tilt = 1; myx +=
sin(angle_rad); viewer[0] +=
				sin(angle_rad); at[0] += sin(angle_rad);
		}
		if (key == GLUT_KEY_RIGHT) {
			myx -= cos(angle_rad) * 1.0; myz +=
				sin(angle_rad) * 1.0; viewer[0] -= cos(angle_rad) * 1.0; viewer[2] +=
sin(angle_rad) * 1.0; at[0]
				-= cos(angle_rad) * 1.0; at[2] += sin(angle_rad) * 1.0; right_tilt = 1;
		}
		if (key == GLUT_KEY_LEFT) {
			myx += cos(angle_rad) * 1.0; myz -=
				sin(angle_rad) * 1.0; viewer[0] += cos(angle_rad) * 1.0; viewer[2] -=
sin(angle_rad) * 1.0; at[0]
				+= cos(angle_rad) * 1.0; at[2] -= sin(angle_rad) * 1.0; left_tilt = 1;
		}
	}
}
void idle() {
	if (start) {
		// Sound is now managed by play_helicopter_sound() function
		play_helicopter_sound();

		if (rotor_angle < 100)
```

```
                rotor_angle += 200.0;
            else if (rotor_angle < 200)
                rotor_angle += 20.5;
            else if (rotor_angle < 300)
                rotor_angle += 300.0;
            else if (rotor_angle < 400)
                rotor_angle += 300.5;
            else
                rotor_angle += 500.0;
            if (rotor_angle > 1000 && rotor_angle < 100000) {
                if (myy < 100) {
                    myy += 1.0;
                    at[1] += 1.0;
                }
            }
        }
        if (stop) {
            // Stop sound when helicopter is stopping
            if (sound_playing) {
                stop_helicopter_sound();
            }

            if (myy > 0.0) {
                myy -= 1.0;
                at[1] -= 1.0;
                if (viewer[1] > 0)
                    viewer[1] -= 1.0;
            }
            else {
                start = 0;
                stop = 0;
            }
        }
        display();
}
void Releasekey(int key, int x, int y) {
        if (key == GLUT_KEY_UP) { forward_tilt = 0; }
        if (key == GLUT_KEY_RIGHT) { right_tilt = 0; heli_tilt = 0; }
        if (key == GLUT_KEY_LEFT) { left_tilt = 0; heli_tilt = 0; }
}
void my_menu(int id) {
        switch (id) {
        case 1:
            start = 1;
            stop = 0;
            rotor_angle = 0.0;
            viewer[0] = 40.0 + myx;
            viewer[1] = 0.0 + myy;
```

```
                viewer[2] = 0.0 + myz;
                play_helicopter_sound(); // Play sound when starting
                break;
        case 2:
                myy += 3.0;
                at[1] += 3.0;
                // Only play sound if in start state
                if (start) {
                        play_helicopter_sound();
                }
                break;
        case 3:
                stop = 1;
                // Sound will be stopped in idle() function
                break;
        case 4:
                exit(0);
        }
        glutPostRedisplay();
}
void view_menu(int id) {
        switch (id) {
        case 1:
                viewer[0] = 0.0 + myx;
                viewer[1] = 1.0;
                viewer[2] = -2.0 + myz;
                at[0] = myx;
                at[1] = myy;
                at[2] = myz;
                // Only play sound if in start state
                if (start) {
                        play_helicopter_sound();
                }
                break;
        case 2:
                viewer[0] = 0.0 + myx;
                viewer[1] = 5.0 + myy;
                viewer[2] = 15.0 + myz;
                at[0] = viewer[0] + 0.0;
                at[1] = 0.0 + viewer[1];
                at[2] = 15.0 + viewer[2];
                // Only play sound if in start state
                if (start) {
                        play_helicopter_sound();
                }
                break;
        case 3:
                viewer[0] = 0.0 + myx;
```

```
                viewer[1] = 100.0 + myy;
                viewer[2] = -15.0 + myz;
                at[0] = myx;
                at[1] = myy;
                at[2] = myz;
                // Only play sound if in start state
                if (start) {
                        play_helicopter_sound();
                }
                break;
        case 4:
                viewer[0] = 5.0 + myx;
                viewer[1] = 5.0 + myy;
                viewer[2] = -45.0 + myz;
                at[0] = myx;
                at[1] = myy;
                at[2] = myz;
                // Only play sound if in start state
                if (start) {
                        play_helicopter_sound();
                }
                break;
        }
        glutPostRedisplay();
}
int main(int argc, char** argv) {

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGBA);
        glutInitWindowSize(250, 250);
        glutInitWindowPosition(0, 0);
        glutCreateWindow("The Helicopter");
        init();
        glutDisplayFunc(display);
        glutIdleFunc(idle);
        glutKeyboardFunc(keyboard);
        glutSpecialFunc(Specialkey);
        glutSpecialUpFunc(Releasekey);
        glutReshapeFunc(reshape);
        sub_menu[0] = glutCreateMenu(view_menu);
        glutAddMenuEntry("Ground View", 1);
        glutAddMenuEntry("Pilot View", 2);
        glutAddMenuEntry("Landscape View", 3);
        glutAddMenuEntry("Follow View", 4);
        glutCreateMenu(my_menu);
        glutAddMenuEntry("START", 1);
        glutAddSubMenu("Viewing", sub_menu[0]);
        glutAddMenuEntry("STOP", 3);
```

```
        glutAddMenuEntry("Exit", 4);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutMainLoop();
}
```

Output / Screenshots

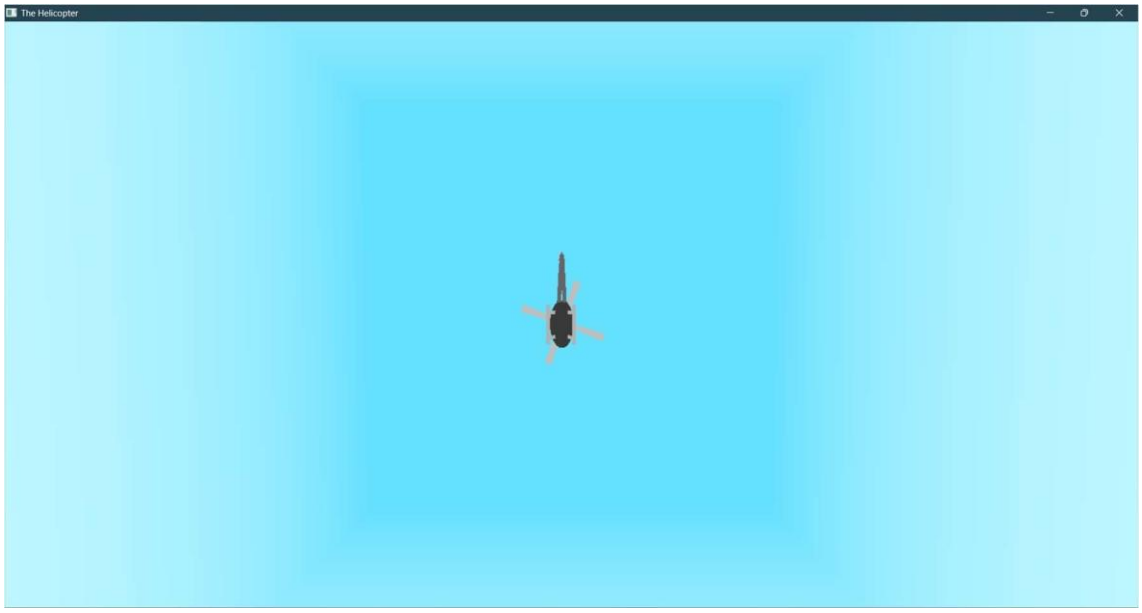
1. Initial Scene

2. Menu

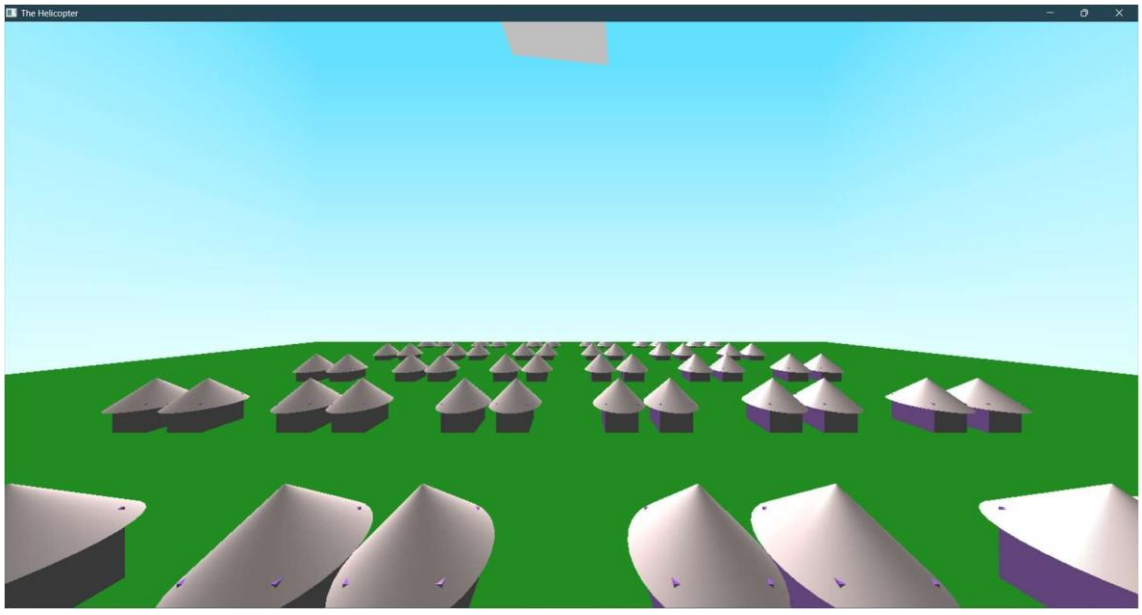
3. Start Animation

4. Viewing SubMenu

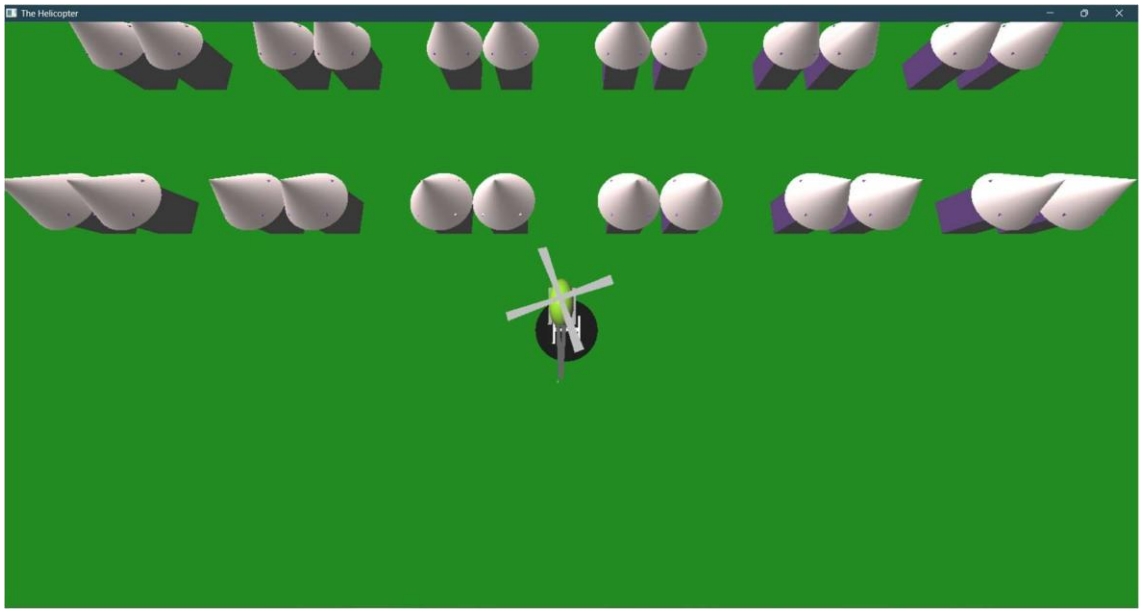
5. Ground View



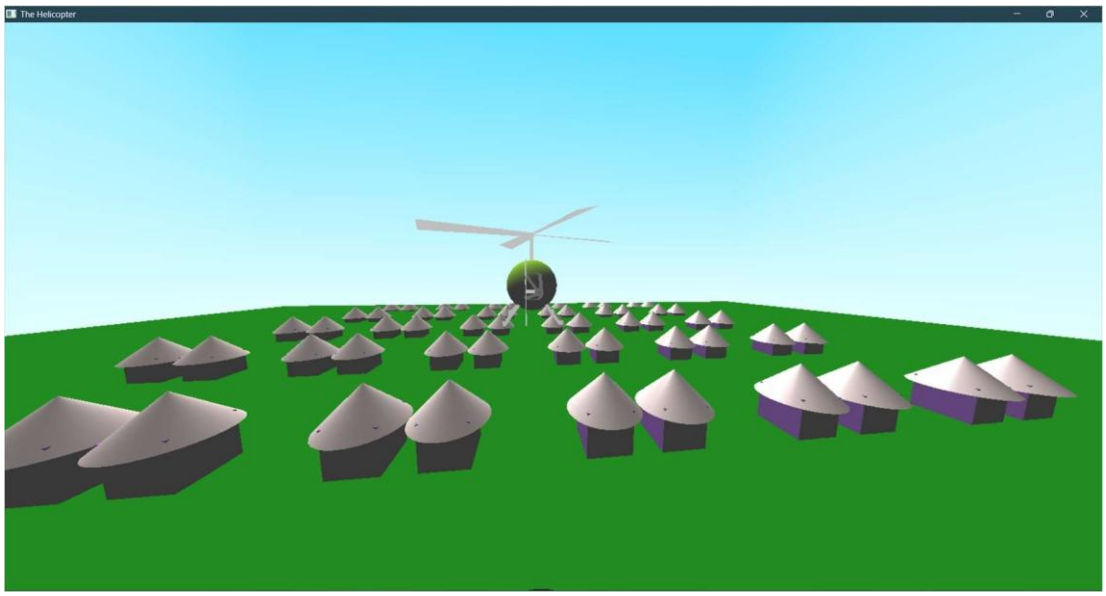
6. Pilot View



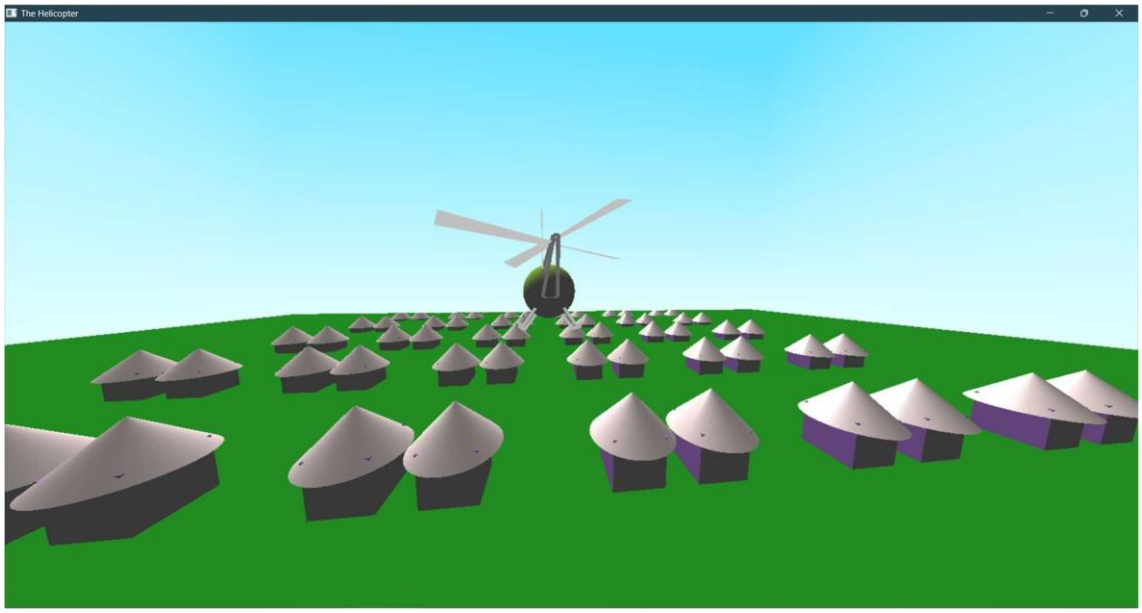
7. Landscape View



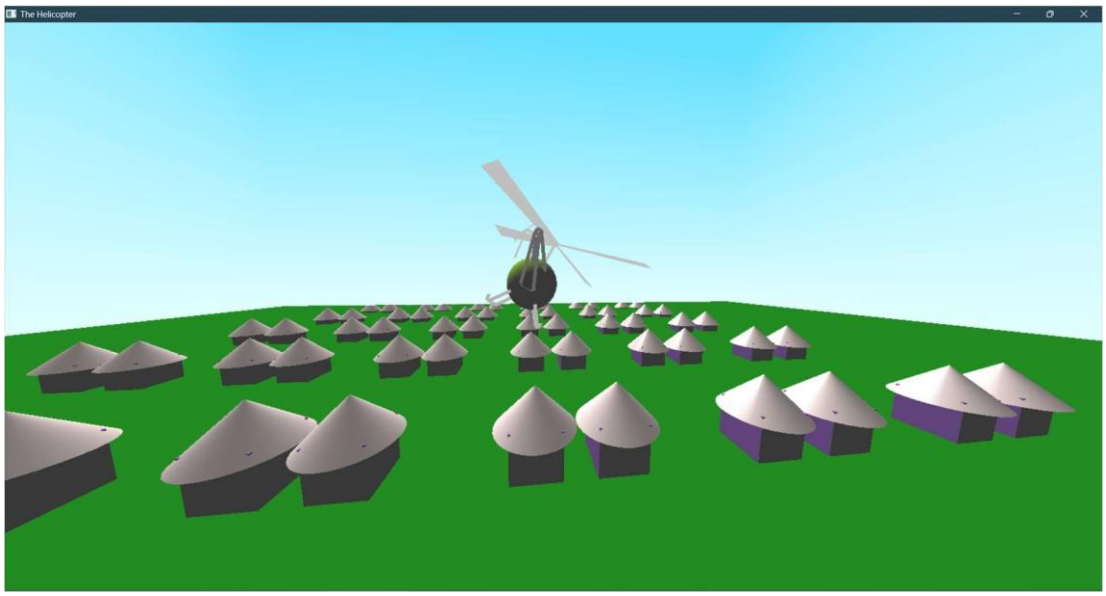
8. Follow View



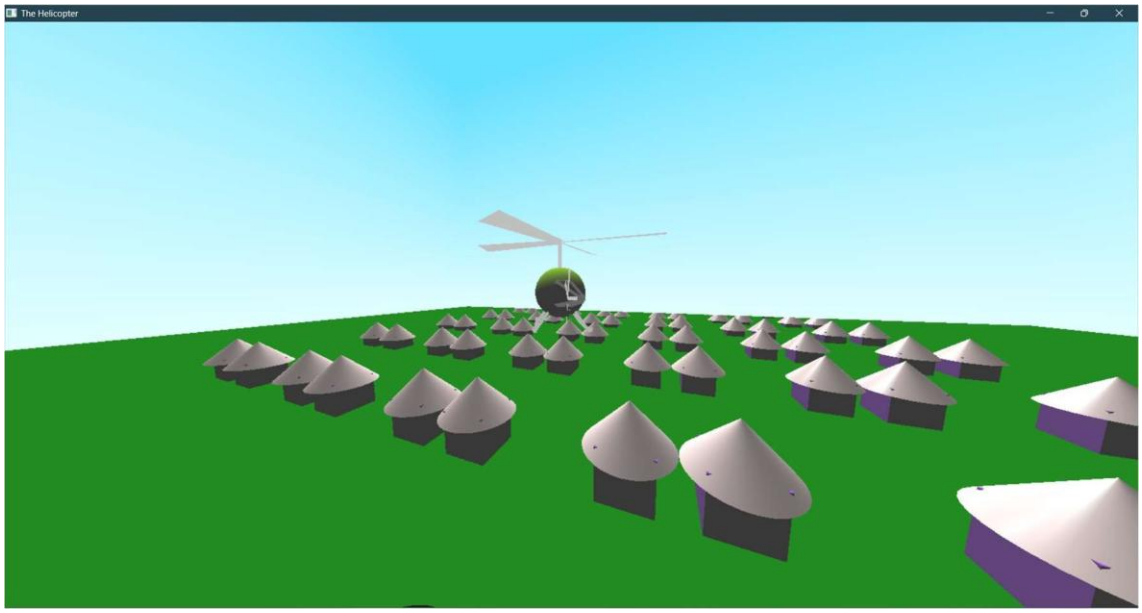
9. Forward Tilt



10. Side Tilt



11. Rotation



12. Stop Animation