FACULTY OF COMPUTER AND INFORMATION SCIENCES, AIN SHAMS UNIVERSITY

Lab 2

2D Transformation and Orthogonal projection

OBJECTIVES:

1. **Be able to utilize Orthogonal projection in OpenGL.**
2. **Be able to perform 2D transformations using OpenGL.**

1. ORTHOGONAL PROJECTION IN OPENGL

In this section we will explain the part of code they just stiffed in the application. First we will explain the idea that OpenGL is stack based library, if we didn't do so in the first section. This means it uses matrix stack for projection, transformations, etc… So, we need to let the OpenGL know which matrix stack I'm altering now by calling *glMatrixMode*.

Then we clear the projection matrix stack by calling *glLoadIdentity.* Finally, set the maximum viewing rectangle to be displayed in the control. Calling *gluOrtho2D* with the appropriate parameters will do that purpose.

```
/**
Creates the main window, registers event handlers, and
initializes OpenGL stuff.
*/
void InitGraphics(int argc, char *argv[]) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
	//Create an 800x600 window with its top-left corner at pixel
(100, 100)
	glutInitWindowPosition(100, 100); //pass (-1, -1) for Window-
Manager defaults
	glutInitWindowSize(800, 600);
	glutCreateWindow("OpenGL Lab");
	//OnDisplay will handle the paint event
	glutDisplayFunc(OnDisplay);
	// here is the setting of the idle function
	glutIdleFunc(OnDisplay);
	// here is the setting of the key function
	glutKeyboardFunc(OnKeyPress);
	glutSpecialFunc(OnSpecialKeyPress);

	SetTransformations();

	glutMainLoop();
}
/**
Sets the logical coordinate system we will use to specify
our drawings.
*/
void SetTransformations() {
	//set up the logical coordinate system of the window: [-100, 100]
x [-100, 100]
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(-100, 100, -100, 100);
	glMatrixMode(GL_MODELVIEW);
}
```

2. 2D TRANSFORMATIONS USING OPENGL

Following the skeleton of the first lab, we'll add some modifications to the paint event. First we clear the transformation matrix stack each time we render. Then we translate according to some variable controlled over the key down event as will be illustrated below. Then rotate the rectangle around its pivot, in our case it's the lower corner of the rectangle, according to another variable controlled in the key down event.

```
/**
Handles the paint event. This event is triggered whenever
our displayed graphics are lost or out-of-date.
ALL rendering code should be written here.
*/
void OnDisplay()
{
	// pushes the current matrix stack down by one,
	// duplicating the current matrix.
	// glPushMatrix and glPopMatrix are used here instead of
glLoadIdentity.
	//glPushMatrix();
	// clear the transformation matrix
	glLoadIdentity();

	//set the background color to white
	glClearColor(1, 1, 1, 1);
	//fill the whole color buffer with the clear color
	glClear(GL_COLOR_BUFFER_BIT);

	glTranslatef(fXPos,fYPos,0);
	glRotatef(fRot, 0, 0, 1);
	//drawing code goes here
	glBegin(GL_QUADS);
		glColor3f(1, 0, 0);
		glVertex3f(-50, -50, 0);
		glVertex3f(50, -50, 0);
		glVertex3f(50, 50, 0);
		glVertex3f(-50, 50, 0);
	glEnd();

	// pops the current matrix stack, replacing the
	// current matrix with the one below it on the stack.
	//glPopMatrix();

	// swapping the buffers causes the rendering above to be
	// shown
	glutSwapBuffers();

}
```

The command `glTranslated` (x, y, z) multiply the current matrix by a translation matrix with the x, y & z values. As, we're working with 2D transformations only the z value will be set to zero. Another overload for this function is `glTranslatef` where the first one parameter's are of type double and the last one are floats.

The command `glRotated` (angle, x, y, z) multiply the current matrix by a rotation matrix. This rotation matrix is by the "angle" around the vector (x, y, z). As, we're working with 2D transformations we need to rotate objects about the Z-Axis, that's why we input to the function the vector (0, 0, 1). Another overload for this function is `glRotatef` where the first one parameter's are of type double and the last one are floats.

The command `glScaled` (x, y, z) multiply the current matrix by a scaling matrix. This scaling matrix is by values of x, y & z along the X, Y & Z-Axis respectively. As, we're working with 2D transformations only the z value will be set to zero. Another overload for this function is `glScalef` where the first one parameter's are of type double and the last one are floats.

The commands `glPushMatrix` and `glPopMatrix` describtion is as follows:

There is a stack of matrices for each of the matrix modes. In GL_MODELVIEW mode, the stack depth is at least 32. In the other two modes, GL_PROJECTION and GL_TEXTURE, the depth is at least 2. The current matrix in any mode is the matrix on the top of the stack for that mode.

`glPushMatrix` pushes the current matrix stack down by one, duplicating the current matrix. That is, after a `glPushMatrix` call, the matrix on top of the stack is identical to the one below it.

`glPopMatrix` pops the current matrix stack, replacing the current matrix with the one below it on the stack.

Initially, each of the stacks contains one matrix, an identity matrix. It is an error to push a full matrix stack, or to pop a matrix stack that contains only a single matrix. In either case, the error flag is set and no other change is made to GL state.

ERRORS:

GL_STACK_OVERFLOW is generated if `glPushMatrix` is called while the current matrix stack is full.

GL_STACK_UNDERFLOW is generated if `glPopMatrix` is called while the current matrix stack contains only a single matrix.

GL_INVALID_OPERATION is generated if `glPushMatrix` or `glPopMatrix` is executed between the execution of `glBegin` and the corresponding execution of `glEnd`.

In this example I've added both `glLoadIdentity and glPushMatrix` with `glPopMatrix` to illustrate their usages. In my opinion, try first the simplest case with the students, the one using `glLoadIdentity` and proceed with the whole program. After they've understood everything POP the 2nd solution to them.

Regarding key handling, we've set the key handling callback functions in the initialize graphics `glutKeyboardFunc and glutSpecialFunc`. The key press events will be as follows:

```
/**
Handles the key press. This event is whenever
a normal ASCII character is being pressed.
*/
void OnKeyPress(unsigned char key, int x, int y)
{
	if (key == 27)
		exit(0);
	switch(key)
	{
	case 'a'://		a key
	case 'A':
		fXPos -= 0.5;
		break;
	case 'd'://		d key
	case 'D':
		fXPos += 0.5;
		break;
	case 'w'://		w key
	case 'W':
		fYPos += 0.5;
		break;
	case 's'://		s key
	case 'S':
		fYPos -= 0.5;
		break;
	case 'e':
	case 'E':
		fRot += 0.1;
		break;
	case 'q':
	case 'Q':
		fRot -= 0.1;
		break;
	};
}
/**
Handles the special key press. This event is whenever
a special key is being pressed.
*/
void OnSpecialKeyPress(int key, int x, int y)
{
	switch(key)
	{
	case GLUT_KEY_LEFT://		Left function key
		fXPos -= 0.5;
		break;
	case GLUT_KEY_RIGHT://		Right function key
		fXPos += 0.5;
		break;
	case GLUT_KEY_UP://		Up function key
		fYPos += 0.5;
		break;
	case GLUT_KEY_DOWN://		Down function key
		fYPos -= 0.5;
		break;
	};
}
```

You can now compile and run the program.