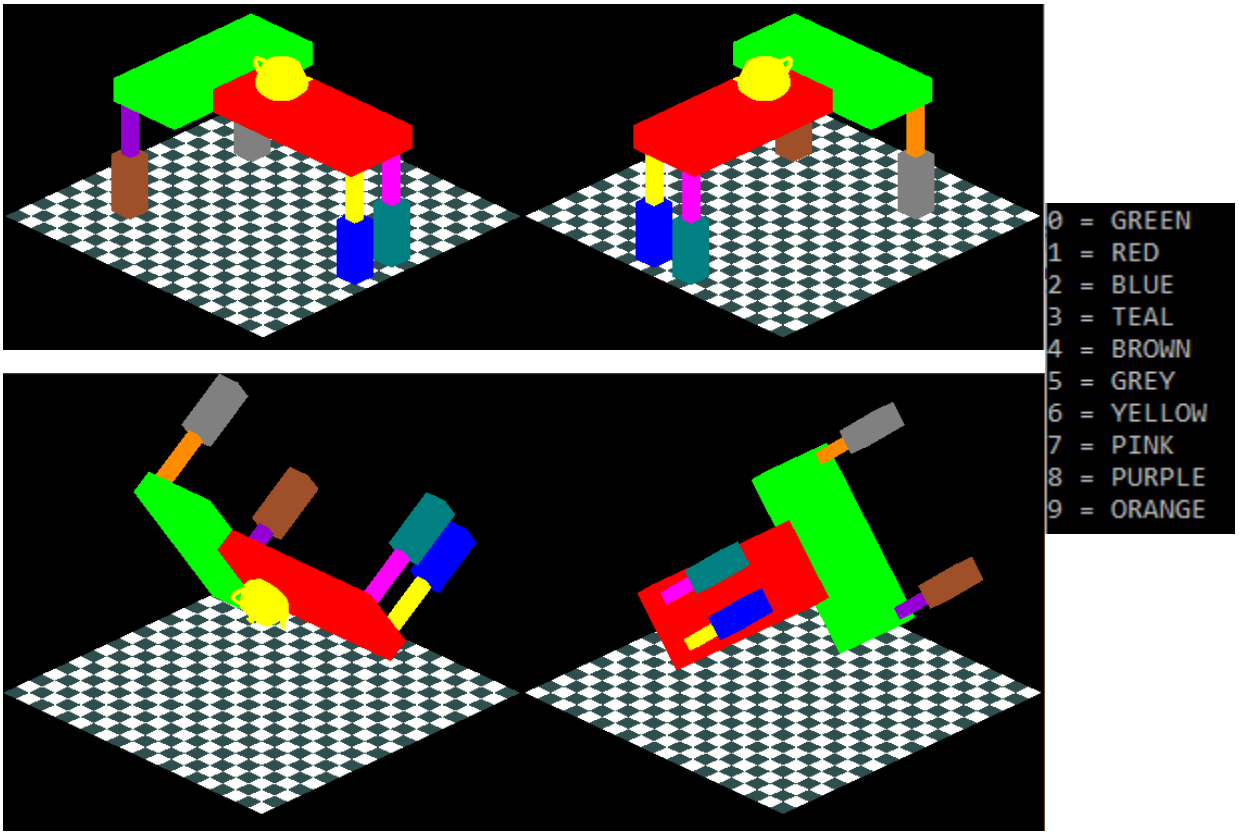
Daniel Frey
CS 4800-001
Programming Assignment #2
Table Rotate

- To make visualization easier, I drew lines along the x, y and z axes. This helped me to understand the coordinates better.
- I changed the floor by increasing the number of the checkerboard pattern. To facilitate the process, I used nested for loops to draw quads.
- I divided the viewport so that I could render the image twice, but with differing views.
- Rotating
    - I kept track of where each piece's origin is located.
    - To rotate about the LCS, I translated to the LCS's origin, rotated, then translated back to the main origin. This made it easier so that the order of translations, scaling, and rotations did not cause an issue.
- Picking
    - I attempted to use ray intersection picking, color picking, and regular mouse click picking. I ran into issues about properly calculating the number of hits that mapped to a specific object.
    - Instead, I opted for keyboard picking where the user, instead of clicking, uses the keyboard to select a number from 0 to 9 to rotate about a specific table object. I added a legend to help the user pick.



```
#include <GL/glut.h>
#include <fstream>
#include <iostream>
#include <math.h>

#define BUFSIZE 512

// constants to relate numbers to table pieces
const char GREEN = '0', RED = '1', BLUE = '2', TEAL = '3', BROWN = '4', GREY = '5', YELLOW = '6', PINK = '7', PURPLE = '8', ORANGE = '9';

// window dimensions
const int WIDTH = 960, HEIGHT = 480;

struct OriginPoint{
	float x, y, z;
}green, red, blue, teal, brown,
grey, yellow, pink, purple, orange, objRot, objOrigin;  // table obj structs

float topWidth, topThick, legThick, legLen, angle, minRot, maxRot;
bool rotating = false, rotatingX = false, rotatingY = false, rotatingZ = false;
char keyPressed;

// Makes a checkerboard floor that's 20x20
void makeFloor(float thickness){
	float currX, currZ, yAdj, adj = .0014f, xRght, zBttm;
	float baseFloorSize = 1.4f;
	float numSquares = 10;
	float chkSize = 1.0f / (baseFloorSize * numSquares) - adj;	// -adj to make fit nicely
	float baseVal = 1.0f / baseFloorSize;		// base value for starting x and starting z positions

	// main floor
	glPushMatrix();	//push and pop to not mess with checkerboard pattern
		glColor3ub(47, 79, 79);	//dark slate grey
		glScalef(1, thickness, 1);
		glutSolidCube(baseFloorSize);
	glPopMatrix();

	yAdj = thickness + (thickness * -.3f + .00005f);		// -.3 for previous adjustment, +.00005 to make slightly above
	adj *= 10;		// small adjustment to make checkerboard fit better

	// checkerboard pattern
	// top row: (-x, -z) -> (x, -z) start from first position; (-z) -> (z)
	// second row: (-x, -z) -> (x, -z) start from second position; (-z) -> (z)
	// outer loop updates z row, inner loop goes across z row for each x position
	for (int i = 0; i < 20; i++){
		// determine if odd or even row for starting x position
		if (i % 2 == 0){
			// draw row that starts from first position
			currX = -baseVal;
		}
		else{
			// draw row that starts from second position
			currX = -baseVal + chkSize;
		}

		currZ = -baseVal + chkSize * i;		// how far to adjust z for each row
		zBttm = currZ + chkSize;	// z bottom position
```

```
		// from starting position, draw square every other spot
		for (currX; currX < (baseVal - chkSize); currX = currX + (2 * chkSize)){
			xRght = currX + chkSize;	// x right position

			glBegin(GL_QUADS);
				glColor3ub(255, 255, 255);		// white
				glVertex3f(currX + adj, yAdj, zBttm + adj);	// btm left
				glVertex3f(xRght + adj, yAdj, zBttm + adj);	// btm right
				glVertex3f(xRght + adj, yAdj, currZ + adj);	// top right
				glVertex3f(currX + adj, yAdj, currZ + adj);	// top left
			glEnd();
		}
	}
}

// Creates one table leg of (thickness, length)
void tableLeg(float thick, float len){
	glScalef(thick, len, thick);
	glutSolidCube(1.0);
}

// Creates a table top (width, thickness, length)
void tableTop(float topWidth, float topThick, float topLength){
	glScalef(topWidth, topThick, topLength);	//define size
	glutSolidCube(1.0);		//apply above to cube

}

// Rotates about a point
void rotateAboutPt(){
	float xO, yO, zO;  // origin points
	bool top = false, leg = false;		// rotating which piece

	// start rotating from x then y then z
	if (rotating){
		// set origin
		xO = objOrigin.x; yO = objOrigin.y; zO = objOrigin.z;

		// determine LCS for x y z
		switch (keyPressed){
			//IF NOT TOP, THEN LEG
			// tops: x=z, y=y, z=x
			case GREEN:
			case RED:
				top = true;
				break;
			default:
				break;
		}

		glTranslatef(xO, yO, zO);	// translate to object origin

		if (rotatingX){
			// z=x for any piece
			glRotatef(angle, 0, 0, 1);	// rotate about 'x' in LCS

			if (angle == maxRot){
				rotatingX = false;
```

```
                        rotatingY = true;
                        angle = minRot;
                    }
                }

                if (rotatingY){
                    if (top)
                        glRotatef(angle, 0, 1, 0);    // rotate about 'y' in LCS
                    else
                        glRotatef(angle, 1, 0, 0);    // rotate about 'y' in LCS

                    if (angle == maxRot){
                        rotatingY = false;
                        rotatingZ = true;
                        angle = minRot;
                    }
                }

                if (rotatingZ){
                    if (top)
                        glRotatef(angle, 1, 0, 0);    // rotate about 'z' in LCS
                    else
                        glRotatef(angle, 0, 1, 0);    // rotate about 'z' in LCS

                    if (angle == maxRot){
                        rotatingZ = false;
                        rotating = false;
                        angle = minRot;
                    }
                }

                angle += 5;
                glTranslatef(-xO, -yO, -zO);// translate back to origin
            }
}

// Creates the table top and table legs
void table(float topWidth, float topThick, float legThick, float legLen, GLenum mode){
        float xPosAdj = .25f, topsXAdj = .022f;         //x-pos for upper table top and lower table top
        float yAdj = .52431f;         // adjustment for y so table sits on floor

        // push and pop table as a whole to treat as one object for rotating
        glPushMatrix();
            rotateAboutPt();
            //create upper table top
            glPushMatrix();
                glColor3ub(0, 255, 0);       //green
                green.x = -xPosAdj - topsXAdj;
                green.y = yAdj;
                green.z = 0;
                glTranslatef(green.x, green.y, green.z);        //define position upperPos, legLen, 0
                glLoadName(GREEN);
                tableTop(topWidth / 1.5f, topThick, topWidth * 1.5f);            //width, thickness, length
            glPopMatrix();

            //create lower table top
            glPushMatrix();
                glColor3ub(255, 0, 0);       //red
```

```
        red.x = topWidth - xPosAdj + topsXAdj;
        red.y = yAdj;
        red.z = 0;
        glTranslatef(red.x, red.y, red.z);      // lowerPos, legLen, 0
        glLoadName(RED);
        tableTop(topWidth * 1.5f, topThick, topWidth / 1.5f);           //width, thickness, length
glPopMatrix();

//create lower table legs
//front-most lower table leg on lower table top
glPushMatrix();
        glColor3ub(0, 0, 255);      //blue
        blue.x = topWidth * 1.8f - legThick / 2.0f - xPosAdj;
        blue.y = topThick - legLen + yAdj;
        blue.z = legThick;
        glTranslatef(blue.x, blue.y, blue.z);
        glLoadName(BLUE);
        tableLeg(legThick, legLen / 2.0f);
glPopMatrix();

//back-most lower table leg on lower table top
glPushMatrix();
        glColor3ub(0, 128, 128);    //teal
        teal.x = topWidth * 1.8f - legThick / 2.0f - xPosAdj;
        teal.y = topThick - legLen + yAdj;
        teal.z = -legThick;
        glTranslatef(teal.x, teal.y, teal.z);
        glLoadName(TEAL);
        tableLeg(legThick, legLen / 2.0f);
glPopMatrix();

//front-most lower table leg on upper table top
glPushMatrix();
        glColor3ub(160, 80, 40);    //brown
        brown.x = -topWidth / 2.0f + legThick * 1.1f - xPosAdj;
        brown.y = topThick - legLen + yAdj;
        brown.z = topWidth / 1.5f;
        glTranslatef(brown.x, brown.y, brown.z);
        glLoadName(BROWN);
        tableLeg(legThick, legLen / 2);
glPopMatrix();

//back-most lower table leg on upper table top
glPushMatrix();
        glColor3ub(128, 128, 128);          //grey
        grey.x = -topWidth / 2.0f + legThick * 1.1f - xPosAdj;
        grey.y = topThick - legLen + yAdj;
        grey.z = -topWidth / 1.5f;
        glTranslatef(grey.x, grey.y, grey.z);
        glLoadName(GREY);
        tableLeg(legThick, legLen / 2.0f);
glPopMatrix();

//create upper table legs
//front-most upper table leg on lower table top
glPushMatrix();
        glColor3ub(255, 255, 0);    //yellow
        yellow.x = topWidth * 1.8f - legThick / 2.0f - xPosAdj;
```

```
                yellow.y = -topThick * 1.75f + yAdj;
                yellow.z = legThick;
                glTranslatef(yellow.x, yellow.y, yellow.z);
                glLoadName(YELLOW);
                tableLeg(legThick / 2, legLen / 2);
        glPopMatrix();

        //back-most upper table leg on lower table top
        glPushMatrix();
                glColor3ub(255, 0, 255);    //pink
                pink.x = topWidth * 1.8f - legThick / 2.0f - xPosAdj;
                pink.y = -topThick * 1.75f + yAdj;
                pink.z = -legThick;
                glTranslatef(pink.x, pink.y, pink.z);
                glLoadName(PINK);
                tableLeg(legThick / 2.0f, legLen / 2.0f);
        glPopMatrix();

        //front-most upper table leg on upper table top
        glPushMatrix();
                glColor3ub(148, 0, 211);    //purple
                purple.x = -topWidth / 2.0f + legThick * 1.1f - xPosAdj;
                purple.y = -topThick * 1.75f + yAdj;
                purple.z = topWidth / 1.5f;
                glTranslatef(purple.x, purple.y, purple.z);
                glLoadName(PURPLE);
                tableLeg(legThick / 2.0f, legLen / 2.0f);
        glPopMatrix();

        //back-most upper table leg on upper table top
        glPushMatrix();
                glColor3ub(255, 140, 0);    //orange
                orange.x = -topWidth / 2.0f + legThick * 1.1f - xPosAdj;
                orange.y = -topThick * 1.75f + yAdj;
                orange.z = -topWidth / 1.5f;
                glTranslatef(orange.x, orange.y, orange.z);
                glLoadName(ORANGE);
                tableLeg(legThick / 2.0f, legLen / 2.0f);
        glPopMatrix();

        // teapot for fun
        glPushMatrix();
                glColor3ub(255, 255, 0);    //yellow
                glTranslatef(.1f, .12f + yAdj, 0);      // teapot center and on table
                glutSolidTeapot(.1);
        glPopMatrix();

    glPopMatrix();
}

// reads the table shape and joint files
void readFiles(float &topWidth, float &topThick, float &legThick, float &legLen, float &minRot, float &maxRot){
    std::ifstream inFile1("table.txt");
    std::ifstream inFile2("joint_file.txt");

    if (inFile1.is_open() && inFile2.is_open()){
        inFile1 >> topWidth >> topThick >> legThick >> legLen;
        inFile2 >> minRot >> maxRot;
```

```
		inFile1.close();
		inFile2.close();
	}
	else
		std::cout << "Could not open one of the files." << std::endl;

}

// reshapes display
void reshape(int w, int h){
	// Sets the view
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glOrtho(-1, 1, -1, 1, .1, 100.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();

}

// draws the world coordinate axes to help with visualization
void drawAxes(void){
	// draw coordinate lines to help visualize
	glBegin(GL_LINES);
		glColor3f(1.0f, 0.0f, 0.0f);
		glVertex3f(0.0f, 0.0f, 0.0f);
		glVertex3f(1.0f, 0.0f, 0.0f);

		glColor3f(0.0f, 1.0f, 0.0f);
		glVertex3f(0.0f, 0.0f, 0.0f);
		glVertex3f(0.0f, 1.0f, 0.0f);

		glColor3f(0.0f, 0.0f, 1.0f);
		glVertex3f(0.0f, 0.0f, 0.0f);
		glVertex3f(0.0f, 0.0f, 1.0f);
	glEnd();
}

// draws the scene with a table and checkerboard floor
void drawScene(GLenum mode){
	//set view 1
	glViewport(0, 0, WIDTH / 2, HEIGHT);		// where to render on screen = left half
	glLoadIdentity();
	gluLookAt(1.0, 1.0, 1.0, 0.0, 0.25, 0.0, 0.0, 1.0, 0.0);
	// Creates the floor
	glPushMatrix();
		makeFloor(0.001f);
	glPopMatrix();
	// Create table
	table(topWidth, topThick, legThick, legLen, mode);	//table dimensions: top width, top thickness, leg thickness, leg length

	//drawAxes();

	//set view 2
	glViewport(WIDTH / 2, 0, WIDTH / 2, HEIGHT);		// where to render on screen, right half
	glLoadIdentity();
	gluLookAt(1.0, 1.0, -1.0, 0.0, 0.25, 0.0, 0.0, 1.0, 0.0);
```

```
	// Creates the floor
	glPushMatrix();
		makeFloor(0.001f);
	glPopMatrix();
	// Create table
	table(topWidth, topThick, legThick, legLen, mode);    //table dimensions: top width, top thickness, leg thickness, leg length

	//drawAxes();
}

// Displays the object created
void display(void){
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	drawScene(GL_RENDER);
	glFlush();
}

// Enables specific GL capabilities
void init(){
	glEnable(GL_DEPTH_TEST);		//enable and update depth buffer
	glEnable(GL_COLOR_MATERIAL);	//enable coloring material
	glShadeModel(GL_FLAT);

	glClearColor(0.f, 0.f, 0.f, 1.f);		//set clear color
}

// timer for rotation
void timer(int extra){
	glutPostRedisplay();
	glutTimerFunc(50, timer, 0);
}

// Get keyboard input to select object to rotate
void keyPick(unsigned char key, int x, int y){
	keyPressed = key;
	std::cout << key << std::endl;

	// determine which key relates to which of the 10 table pieces
	switch (key){
		// set origin point
		case GREEN:
			objOrigin = green;
			rotating = true;
			rotatingX = true;
			break;
		case RED:
			objOrigin = red;
			rotating = true;
			rotatingX = true;
			break;
		case BLUE:
			objOrigin = blue;
			rotating = true;
			rotatingX = true;
			break;
		case TEAL:
			objOrigin = teal;
			rotating = true;
```

```
                        rotatingX = true;
                        break;
                case BROWN:
                        objOrigin = brown;
                        rotating = true;
                        rotatingX = true;
                        break;
                case GREY:
                        objOrigin = grey;
                        rotating = true;
                        rotatingX = true;
                        break;
                case YELLOW:
                        objOrigin = yellow;
                        rotating = true;
                        rotatingX = true;
                        break;
                case PINK:
                        objOrigin = pink;
                        rotating = true;
                        rotatingX = true;
                        break;
                case PURPLE:
                        objOrigin = purple;
                        rotating = true;
                        rotatingX = true;
                        break;
                case ORANGE:
                        objOrigin = orange;
                        rotating = true;
                        rotatingX = true;
                        break;
                default:
                        break;
        }

        glutPostRedisplay();
}

int main(int argc, char** argv){
        readFiles(topWidth, topThick, legThick, legLen, minRot, maxRot);

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);       //set initial display mode
        glutInitWindowSize(WIDTH, HEIGHT);          //specify window size
        glutInitWindowPosition(200, 100);  //specify window location
        glutCreateWindow("Table and Checkerboard Floor");//give window title
        init();
        glutKeyboardFunc(keyPick);
        glutReshapeFunc(reshape);           //sets the view for reshaping
        glutTimerFunc(0, timer, 0);             //sets the rotation timer to update the image
        glutDisplayFunc(display);   //set display callback for current window
        glutMainLoop();

        return 0;
}
```