Daniel Meyer

CSE 420-01

Homework 3

**Homework 3 Report**

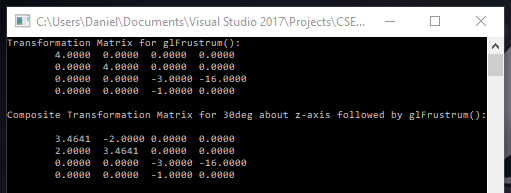
**Part 1: (success)**

A = (2,2,1) B = (1,-2,0)

Proj**B** **A** = ( **A.B** / |**B**|2 ) **B =** [-2 / sqrt(5)] \* (1,-2,0) = (-0.894, 1.789, 0)

Perp**B** **A** = **A** - Proj**B** **A =** (2,2,1) – (-0.894, 1.789, 0) = (1.106, 0.211, 1)

**Part 2: (success)**



glLoadIdentity();

glFrustum(-1.0, 1.0, -1.0, 1.0, 4.0, 8.0);

glGetFloatv(GL\_MODELVIEW\_MATRIX, &p[0][0]);

cout << "Transformation Matrix for glFrustrum():" << endl;

print\_mat(p);

glPushMatrix();

glLoadIdentity();

glRotatef(30, 0, 0, 1);

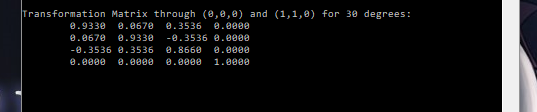
glFrustum(-1.0, 1.0, -1.0, 1.0, 4.0, 8.0);

glGetFloatv(GL\_MODELVIEW\_MATRIX, &p[0][0]);

cout << "Composite Transformation Matrix for 30deg about z-axis followed by glFrustrum():" << endl;

print\_mat(p);

**Part 3: (success)**



glLoadIdentity();

glTranslatef(1, 1, 0);

glRotatef(30, 1, 1, 0);

glTranslatef(-1, -1, 0);

glGetFloatv(GL\_MODELVIEW\_MATRIX, &p[0][0]);

cout << "Transformation Matrix through (0,0,0) and (1,1,0) for 30 degrees:" << endl;

print\_mat(p);

**Part 4: (success)**

-1 ≤ x ≤ 2, 6 ≤ y ≤ 8, and 0 ≤ z ≤ 6

Viewpoint @ (8, 6, 9)

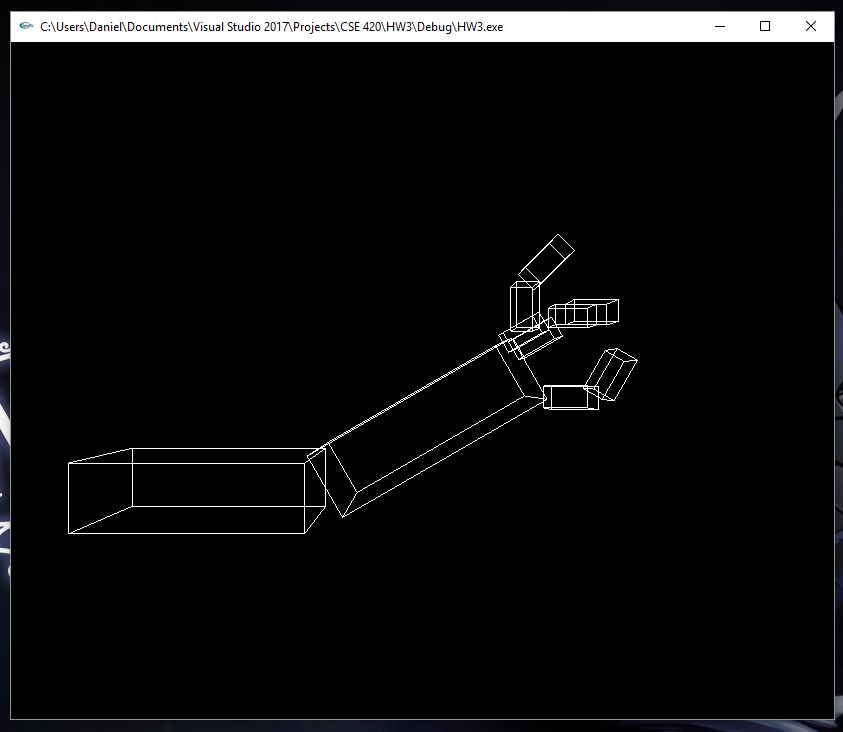
Center @ (0.5, 7, 3)

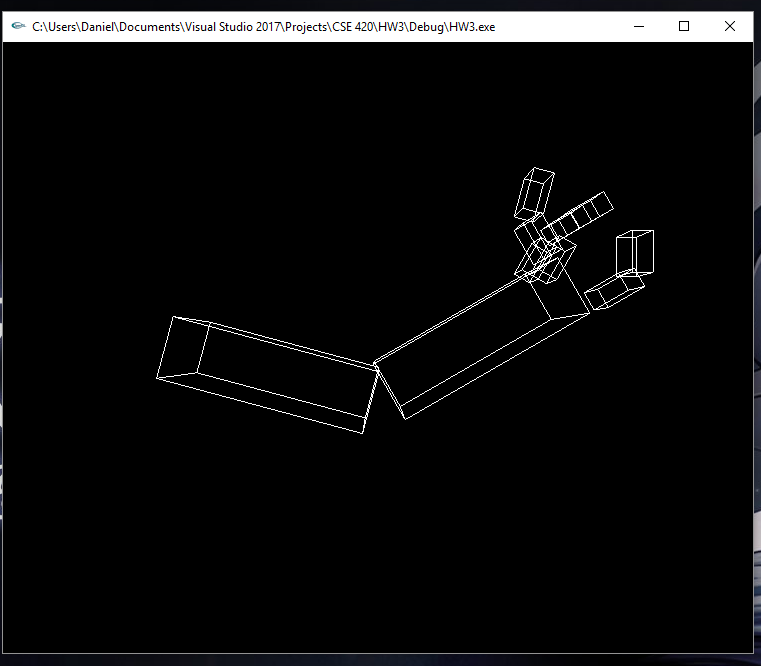
r = ½ \* size = sqrt[ (2 – 0.5)2 + (8 - 7)2 + (6 - 3)2 ] = sqrt(12.25) = 3.5

d = sqrt[ (8 - 0.5)2 + (6 - 7)2 + (9 - 3)2 ] = sqrt(93.25) = 9.656

theta = 2arcTan( r / d ) = 2arcTan( 3.5 / 9.656 ) = 2 \* 19.924 = 39.848 degrees

**Part 5: (success)**





static int shoulder = 0, elbow = 0, hand = 0;

static int fingers = 0, thumb = 0;

static int armX = 0, armY = 0;

void init(void)

{

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL\_FLAT);

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glPushMatrix();

glTranslatef((GLfloat)armX, armY, 0);

glTranslatef(-1.0, 0.0, 0.0);

glRotatef((GLfloat)shoulder, 0.0, 0.0, 1.0);

glTranslatef(1.0, 0.0, 0.0);

glPushMatrix();

glScalef(2.0, 0.6, 1.0);

// glScalef( 2.0, 0.1, 1.0 );

glutWireCube(1.0);

glPopMatrix();

glTranslatef(1.0, 0.0, 0.0);

glRotatef((GLfloat)elbow, 0.0, 0.0, 1.0);

glRotatef(30, 0.0, 0.0, 1.0);

glTranslatef(1.2, 0.0, 0.0);

glPushMatrix();

glScalef(2.0, 0.6, 1.0);

glutWireCube(1.0);

glPopMatrix();

//thumb

glTranslatef(1.0, 0.0, 0.0);

glRotatef((GLfloat)thumb, 0.0, 0.0, 1.0);

glRotatef(-30, 0.0, 0.0, 1.0);

glTranslatef(0.4, -0.25, 0.3);

glPushMatrix();

glScalef(0.4, 0.2, 0.3);

glutWireCube(1.0);

glPopMatrix();

//FINGER 1

glTranslatef(-0.4, 0.8, 0.0);

glRotatef(90, 0.0, 0.0, 1.0);

glPushMatrix();

glScalef(0.4, 0.2, 0.3);

glutWireCube(1.0);

glPopMatrix();

//FINGER 2

glTranslatef(0.0, 0.0, -0.3);

glRotatef(-60, 0.0, 0.0, 1.0);

glTranslatef(0.0, -0.3, 0.0);

glPushMatrix();

glScalef(0.4, 0.2, 0.3);

glutWireCube(1.0);

glPopMatrix();

//Finger 3

glTranslatef(0.0, 0.0, -0.3);

glTranslatef(0.0, 0.1, 0.0);

glPushMatrix();

glScalef(0.4, 0.2, 0.3);

glutWireCube(1.0);

glPopMatrix();

//Finger tip 3

glTranslatef(0.5, -0.1, 0.0);

glRotatef(-30, 0.0, 0.0, 1.0);

glPushMatrix();

glScalef(0.4, 0.2, 0.3);

glutWireCube(1.0);

glPopMatrix();

//Finger tip 2

glTranslatef(0.1, 0.0, 0.3);

glPushMatrix();

glScalef(0.4, 0.2, 0.3);

glutWireCube(1.0);

glPopMatrix();

//Finger tip 1

glTranslatef(-0.5, 0.4, 0.3);

glRotatef(45, 0.0, 0.0, 1.0);

glPushMatrix();

glScalef(0.4, 0.2, 0.3);

glutWireCube(1.0);

glPopMatrix();

//Thumb tip

glTranslatef(-0.3, -1.1, 0.0);

glRotatef(15, 0.0, 0.0, 1.0);

glPushMatrix();

glScalef(0.4, 0.2, 0.3);

glutWireCube(1.0);

glPopMatrix();

glPopMatrix();

glutSwapBuffers();

}

void reshape(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(65.0, (GLfloat)w / (GLfloat)h, 1.0, 20.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0, 0.0, -5.0);

}

void keyboard(unsigned char key, int x, int y)

{

switch (key) {

case 's':

shoulder = (shoulder + 5) % 360;

glutPostRedisplay();

break;

case 'S':

shoulder = (shoulder - 5) % 360;

glutPostRedisplay();

break;

case 'e':

elbow = (elbow + 5) % 360;

glutPostRedisplay();

break;

case 'E':

elbow = (elbow - 5) % 360;

glutPostRedisplay();

break;

case 'h':

hand = (hand + 5) % 360;

glutPostRedisplay();

break;

case 'H':

hand = (hand - 5) % 360;

glutPostRedisplay();

break;

case 'f':

fingers = (fingers + 5) % 360;

thumb = (thumb - 5) % 360;

glutPostRedisplay();

break;

case 'F':

fingers = (fingers - 5) % 360;

thumb = (thumb + 5) % 360;

glutPostRedisplay();

break;

case 'a':

armX = (armX + 1) % 360;

glutPostRedisplay();

break;

case 'A':

armX = (armX - 1) % 360;

glutPostRedisplay();

break;

case 'z':

armY = (armY + 1) % 360;

glutPostRedisplay();

break;

case 'Z':

armY = (armY - 1) % 360;

glutPostRedisplay();

break;

case 27:

exit(0);

break;

default:

break;

}

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow(argv[0]);

init();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

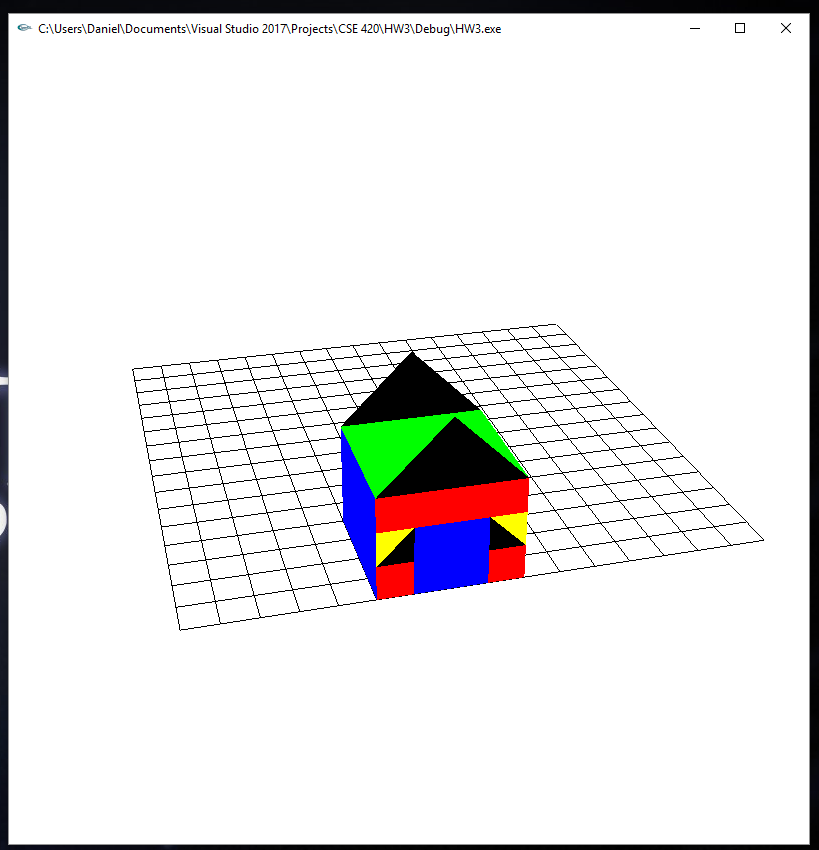
glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}

**Extra Credit: (success)**



#include <stdio.h>

#include <GL/glut.h>

float tx = 0, ty = 0, tz = 0; // translations

float ax = -45, ay = 0, az = 0; // rotations

bool triangles = false;

class Quads

{

public:

float x[4], y[4], z[4];

float r, g, b;

};

Quads Q[100];

int nQuads = 0;

int sideQ = 0;

class Triangles

{

public:

float x[3], y[3], z[3];

float r, g, b;

};

Triangles T[100];

int nTriangles = 0;

int sideT = 0;

void addQuad()

{

Q[nQuads].x[sideQ] = tx;

Q[nQuads].y[sideQ] = ty;

Q[nQuads].z[sideQ] = tz;

sideQ++;

if (sideQ > 3) {

nQuads++;

sideQ = 0;

}

}

void drawQuads()

{

for (int i = 0; i < nQuads; i++) {

glColor3f(Q[i].r, Q[i].g, Q[i].b);

glBegin(GL\_QUADS);

for (int j = 0; j < 4; j++)

glVertex3f(Q[i].x[j], Q[i].y[j], Q[i].z[j]);

glEnd();

}

}

void addTriangle()

{

T[nTriangles].x[sideT] = tx;

T[nTriangles].y[sideT] = ty;

T[nTriangles].z[sideT] = tz;

sideT++;

if (sideT > 2)

{

nTriangles++;

sideT = 0;

}

}

void drawTriangle()

{

for (int i = 0; i < nTriangles; i++) {

glColor3f(T[i].r, T[i].g, T[i].b);

glBegin(GL\_TRIANGLES);

for (int j = 0; j < 3; j++)

glVertex3f(T[i].x[j], T[i].y[j], T[i].z[j]);

glEnd();

}

}

//initialization

void init(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0); //get white background color

glColor3f(0.0f, 0.0f, 0.0f); //set drawing color

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(35, 1.0f, 4.0f, 1000);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void drawGrid()

{

glColor3f(0, 0, 0);

for (int i = 0; i < 34; i++) {

glPushMatrix();

if (i < 17)

glTranslatef(0, i, 0);

else {

glTranslatef(i - 17, 0, 0);

glRotatef(90, 0, 0, 1); //rotate about z-axis by 90 CCW

}

glBegin(GL\_LINES);

glVertex3f(0, 0, 0);

glVertex3f(16, 0, 0);

glEnd();

glPopMatrix();

}

}

void drawCube()

{

glColor3f(1, 0, 0);

glPushMatrix();

glTranslatef(tx, ty, tz);

glutSolidCube(0.5);

glPopMatrix();

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT); //clear screen

glLoadIdentity();

gluLookAt(0.0, 0.0, 40.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glRotatef(ax, 1, 0, 0);

glRotatef(ay, 0, 1, 0);

glRotatef(az, 0, 0, 1);

glTranslatef(-8, -8, 0);

drawGrid();

drawCube();

drawQuads();

drawTriangle();

glFlush(); //send all output to screen

}

void setColor(float r, float g, float b)

{

Q[nQuads].r = r;

Q[nQuads].g = g;

Q[nQuads].b = b;

}

void keyboard(unsigned char key, int mousex, int mousey)

{

switch (key) {

case 'u': // up

tz += 1;

break;

case 'd': // down

tz -= 1;

break;

case 'a': //rotation angle

ax -= 1;

break;

case 'A': //rotation angle

ax += 1;

break;

case 'z': //rotation angle

az -= 1;

break;

case 'Z': //rotation angle

az += 1;

break;

case 32: //space

if (triangles == false)

addQuad();

else if(triangles == true)

addTriangle();

break;

case 'R':

setColor(1, 0, 0);

break;

case 'G':

setColor(0, 1, 0);

break;

case 'B':

setColor(0, 0, 1);

break;

case 'Y':

setColor(1, 1, 0);

break;

case 't':

if (triangles == false)

triangles = true;

else if(triangles == true)

triangles = false;

break;

case 27: // escape

exit(-1);

}

glutPostRedisplay();

}

void specialKey(int key, int mousex, int mousey)

{

switch (key) {

case GLUT\_KEY\_UP:

ty += 1;

break;

case GLUT\_KEY\_DOWN:

ty -= 1;

break;

case GLUT\_KEY\_LEFT:

tx -= 1;

break;

case GLUT\_KEY\_RIGHT:

tx += 1;

break;

}

glutPostRedisplay();

}

void myMouse(int button, int state, int x, int y)

{

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN);

glFlush(); //send all output to screen

}

void reshape(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(35, (GLfloat)w / (GLfloat)h, 4.0f, 1000);

// gluPerspective(65.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0, 0.0, -5.0);

}

/\* Main Loop

\* Open window with initial window size, title bar,

\* RGBA display mode, depth buffer.

\*/

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv); //initialize toolkit

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); //set display mode

glutInitWindowSize(800, 800); //set window size on screen

glutInitWindowPosition(100, 150); //set window position on screen

glutCreateWindow(argv[0]); //open screen widow

init();

glutDisplayFunc(display); //points to display function

glutKeyboardFunc(keyboard);

glutSpecialFunc(specialKey);

glutMouseFunc(myMouse);

glutReshapeFunc(reshape);

glutMainLoop(); //go into perpetual loop

return 0;

}

**Summary:**

This homework assignment focused on matrix multiplication and order of affine transformations. The first part of the assignment asked that we calculate Proj**B** **A** and Perp**B** **A** for the following points: A = (2,2,1) and B = (1,-2, 0) which yielded Proj**B** **A** = (-0.894, 1.789, 0) and Perp**B** **A =** (1.106, 0.211, 1). For the next part of the assignment I had to perform the given glFrustrum command print the corresponding transformation matrix which was then followed by performing a rotation on the matrix about the z-axis and printing the corresponding composite matrix of which both can be seen above. For the third part of the assignment I found the transformation matrix rotating about the axis through two given points and printed the answer found in my OpenGL program I used to verify my answer. For the next part of the assignment I had to find the angle of the FOV for the given viewpoint and x,y,z bounding coordinates. TO find the angle I first had to find the center and calculate the radius and distance. Using the radius and distance I calculated the angle by multiplying the arc tangent of the radius, divided by the distance, by 2. This produced an angle of 39.848 degrees. For the last part of the assignment I had to produce the robot arm introduced in the lecture notes and apply functionality for rotating, swinging, and translating the arm along with finger gripping motions. I also added appropriate keyboard commands to perform each of these actions. Finally, for the extra credit portion, I modified the drawGrid program we had worked on in class to also draw triangles, which can be toggled using ‘t’ and then to showcase this, created a simple house with a roof, windows, and door. Each portion of the assignment compiled and executed correctly and as such feel I earned 80 points (60 for base assignment, 20 for Extra Credit).