Assignment 2b

CS4610

Michael Rallo – msr5zb – 12358133

3/13/2017

Assignment 2b

The *purpose* of this assignment was to familiarize ourselves with manipulating objects in space. We were to learn how to manipulate the object via rotation, translation, as well as using different perspectives/views to see how our view of the object changes.

Objectives:

1. On top of Assignment 2a, define a virtual camera in a 3D virtual scene, specifying its position, orientation and field of view.
2. Using your graphical user interface (GUI) such as GLUT or equivalent, together with the mouse and keyboard, interactively perform the following tasks:
    a. Translate the model / camera in X, Y and Z directions.
    b. Rotate the model / camera around X, Y, and Z axes.
    c. Rotate the model /camera according to the moving direction and distance of the mouse.
    d. Zoom in and zoom out view of the model.

Approach: I created a function that would load in the OBJ file data into global arrays to be rendered later on. In order to ensure that the object was displayed correctly, I kept track of the min/max positions of the object and transitions/scaled it to fit in the view. By default I am using the projection display mode of which the field of vision is set to 55. Note the scalar function helps keep everything in view by default. I also added in keyboard functions in order to swap between different objects and display modes. Translation and camera views manipulation was also added in. Translating the model was easily done, I simply translated in space ‘x’ amount based on the key inputted. In order to rotate the model, I kept track of the onclick cords (before) and computed the new cords of the mouse as it dragged – from which I rotated the object accordingly. To zoom, I adjusted the camera’s field of view. All of these actions and methods will be discussed in further detail below.

The Header File (OpenGLDefaults.h)

First and foremost, I have decided to include a header file to be used for this assignment's, as well as future assignments', libraries. Note this file include OpenGl basic libraries, as well as printing for debugging and math for easy/complex calculations.

```
#ifndef OPENGLDEFAULTS
#define OPENGLDEFAULTS
#define WIN32
#define _CRT_SECURE_NO_DEPRECAT

/*Standards*/
#include <stdio.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <iostream>

/*OpenGL and Common*/
#include<vector>
#ifdef USEGLEW
#include <GL/glut.h>
#endif
#define GL_GLECT_PROTOTYPES
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif


#endif
```

Main

```
int main(int argc, char* argv[]) {

    //Setups
    loadObject("../Objs/cube.obj");
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(windowWidth, windowHeight);
    glutCreateWindow(windowName);

    //Inputs
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(windowKey);
    glutSpecialFunc(windowSpecial);
    glutMouseFunc(mouseActions);
    glutMotionFunc(myMotion);

    glutMainLoop();
    return 0;

}
```

Our main function is similar to that from Assignment 2a. Note by default we load in the cube object file.

Global Variables

```
#include "OpenGLDefaults.h"
#define PI 3.1415926535898
#define Cos(yAngle) cos(PI/180*(yAngle))
#define Sin(yAngle) sin(PI/180*(yAngle))

/*Michael Rallo msr5zb 12358133*/

/*Env Globals*/
double dim = 4;
char* windowName = "Perspective";
int windowWidth = 600;
int windowHeight = 600;


/*State Globals*/
enum Axis { AXIS_ON, AXIS_OFF };
Axis axis = AXIS_ON;
double xAngle = 0;
double yAngle = 0;
double zAngle = 0;
double aspectRatio = 1;
double scaler = 1;
double xTranslate = 0;
double yTranslate = 0;
double zTranslate = 0;

int xMouse;
int yMouse;
int oldX;
int oldY;
int zIndexEnabled = 0;

bool isPressed = false;

int dragging = 0;
enum ViewMode {VIEW_PROJECTION, VIEW_ORTHOGONAL};
ViewMode viewMode = VIEW_PROJECTION;
int fieldOfView = 55;
```

For this assignment, I will be using global variables for the window and view values. I will also be adding in an XYZ Grid in order to “see” the object more clearly in 3d space. Vertices and Faces from the file will be loaded into the vertices and faces vectors. The DisplayType will keep track of how we will be displaying our object. Note the fieldOfView will be used for our perspective (projection) view. Also note the variables we will be using to keep track of mouse positioning.

Drawing the XYZ Grid (Extras)

```
/*Draws the 3d Axis Grid to the Screen*/
void drawAxis() {
    if (axis == AXIS_ON) {
        double axisLength = 3;
        glColor3f(1.0, 1.0, 1.0);
        glBegin(GL_LINES);
        glVertex3d(0, 0, 0);
        glVertex3d(axisLength, 0, 0);
        glVertex3d(0, 0, 0);
        glVertex3d(0, axisLength, 0);
        glVertex3d(0, 0, 0);
        glVertex3d(0, 0, axisLength);
        glEnd();
    }
}
```

This is a simple function to draw a grid at the origin of our view in order for us to see the object more clearly. This can be toggled on and off with the “i” key. By default, it is on.

```
/*Loads the .OBJ file given*/
void loadObject(char* path) {

    //Grab File
    FILE *file = fopen(path, "r");
    if (file == NULL) {
        printf("Impossible to open the file !\n");
    }
    printf("Grabbed file successfully!\n");

    //Data Holders
    char c;
    GLfloat f1, f2, f3, *arrayfloat;
    GLint d1, d2, d3, *arrayint;
    vertices.clear();
    faces.clear();

    int initFlag = -1;
    while (!feof(file)) {
        //Get Datas
        fscanf(file, "%c", &c);

        //Store Vertices
        if (c == 'v') {
            arrayfloat = new GLfloat[3];
            fscanf(file, "%f %f %f", &f1, &f2, &f3);
            arrayfloat[0] = f1;
            arrayfloat[1] = f2;
            arrayfloat[2] = f3;
            vertices.push_back(arrayfloat);

            //Set Mins and Maxes, will be used for Scaling and Translating.
            if (initFlag == -1) {
                minX = f1;
                maxX = f1;
                minY = f2;
                maxY = f2;
                minZ = f3;
                maxZ = f3;
                initFlag = 0;
            }
            if (f1 > maxX) { maxX = f1; }
            if (f1 < minX) { minX = f1; }
            if (f2 > maxY) { maxY = f2; }
            if (f2 < minY) { minY = f2; }
            if (f3 > maxZ) { maxZ = f3; }
            if (f3 < minZ) { minZ = f3; }
        }

        //Store Faces
        else if (c == 'f') {
            arrayint = new GLint[3];
            fscanf(file, "%d %d %d", &d1, &d2, &d3);
            arrayint[0] = d1;
            arrayint[1] = d2;
            arrayint[2] = d3;
            faces.push_back(arrayint);
        }
    }
    fclose(file);
}
```

Loading the File (loadObject)

This function take the object the as a parameter and sets our global vertices and faces variables with the data the OBJ file contains. This function also sets the min/max values that will be later used for scaling/transitioning our object.

Scaling (Scale)

This scale method finds the greatest distance between the X, Y, and Z axis and uses that as a scaler for this Object. The reason we use the longest distance is so that we can scale everything equally whilst still being in our view.

```
/*Scales the object that has been Loaded in*/
void scale() {
    /*Scale*/
    double distanceX = abs(maxX - minX);
    double distanceY = abs(maxY - minY);
    double distanceZ = abs(maxZ - minZ);

    //Find the max distance in order to find best scaler.
    double maxDistance;
    if (distanceX > distanceY && distanceX > distanceZ) {
        maxDistance = distanceX;
    }
    else if (distanceY > distanceX && distanceY > distanceZ) {
        maxDistance = distanceY;
    }
    else {
        maxDistance = distanceZ;
    }
    //Calculate Scaler
    scaler = (dim - 0.5) / maxDistance;
}
```

Display

Our display function is pretty simple. We start by rotating the object slightly in order to achieve the best starting position for our object in 3d space. Note for our perspective/projection view we use gluLookAt(camera cords, origin, up vector).

Our glTranslated function takes the midpoint of our object's X, Y, and Z points in order to translate our object to the origin of our view.

From there, depending on the set displayType, the object will be displayed accordingly (default is set to vectors because it looks really cool).

```
void display() {

    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glLoadIdentity();

    //Adjust According to View Mode
    if (viewMode == VIEW_PROJECTION) {
        double Ex = -2*dim*Sin(yAngle)*Cos(xAngle);
        double Ey = +2*dim          *Sin(xAngle);
        double Ez = +2*dim*Cos(yAngle)*Cos(xAngle);
        //printf("EYE: %f, %f, %f\n", Ex, Ey, Ez);
        printf("Up: %f, Z:%f\n", Cos(xAngle), Sin(zAngle));
        gluLookAt(Ex, Ey, Ez, 0, 0, 0, 0, Cos(xAngle), 0);
        //gluLookAt(xAngle, yAngle, zAngle, 0, 0, 0, 0, Cos(xAngle), 0);


    }
    else {
        //Angled Rotation for Better View of 3d Objects
        printf("Up: %f, Z:%f\n", yAngle, zAngle);

        //Set Inputted Rotation
        glRotated(xAngle, 1, 0, 0);
        glRotated(yAngle, 0, 1, 0);
        if (zIndexEnabled) {
            glRotated(zAngle, 0, 0, 1);
        }
    }

    //Draw Axis
    drawAxis();

    //Colors/Sizes
    glColor3f(0.0, 1.0, 0.0);
    glPointSize(4);
```

```
    //Scale
    glScaled(scaler, scaler, scaler);

    //Move Object to Center
    glTranslated(xTranslate, yTranslate, zTranslate);
    glTranslated(-(minX + maxX) / 2, -(minY + maxY) / 2, -(minZ + maxZ) / 2);

    //Display with Desired Type
    switch (displayType) {
    case POINT:
        glBegin(GL_POINTS);
        for (int i = 0; i < vertices.size(); i++) {
            glVertex3fv(vertices[i]);
        }
        glEnd();
        break;

    case VECTOR:
        glBegin(GL_LINES);
        for (int i = 0; i < faces.size(); i++) {
            glVertex3fv(vertices[faces[i][0] - 1]);
            glVertex3fv(vertices[faces[i][1] - 1]);
            glVertex3fv(vertices[faces[i][1] - 1]);
            glVertex3fv(vertices[faces[i][2] - 1]);
            glVertex3fv(vertices[faces[i][2] - 1]);
            glVertex3fv(vertices[faces[i][0] - 1]);
        }
        glEnd();
        break;

    case FACES:
        glBegin(GL_TRIANGLES);
        for (int i = 0; i < faces.size(); i++) {
            glVertex3fv(vertices[faces[i][0] - 1]);
            glVertex3fv(vertices[faces[i][1] - 1]);
            glVertex3fv(vertices[faces[i][2] - 1]);
        }
        glEnd();
```

Reshape

```
/*Display Details*/
void project() {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    //Adjust according to ViewMode Active
    if (viewMode == VIEW_PROJECTION) {
        gluPerspective(fieldOfView, aspectRatio, dim / 4, 4 * dim);
    }
    else {
        glOrtho
        (-dim*aspectRatio, +dim*aspectRatio, -dim, +dim, -dim, +dim);
    }

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

/*Updates Display, Keeping Aspect Ratio if Window is changed*/
void reshape(int width, int height) {
    aspectRatio = (height > 0) ? (double)width / height : 1;
    glViewport(0, 0, width, height);
    project();
}
```

Our reshape function is very basic. Depending on the view enabled, it will set it to be Perspective/Projection or Orthographic.

Keyboard Input

Key: Esc exits the program.
Keys: 1, 2, 3 loads in different Objects.
Keys: a, s, d changes display type.
Key: i toggles the XYZ grid.
Keys: +, - Zooms In/Out.
Keys: m,n Scales Objects.
Key: p switches from Perspective to Orthographic (Default is Perspective).
Key: o enables Z-Axis Rotation for Orthographic View.
Array Keys adjusts/rotates object/view incrementally.
Click and Drag to Rotate Object, further you drag quicker it rotates.

```
if (key == 27) exit(0);
switch (key) {

//Easy Exit
case 27: //ESC
    exit(0);
    break;

//Object Loaders
case '1':
    loadObject("../Objs/cube.obj");
    break;
case '2':
    loadObject("../Objs/pig.obj");
    break;
case '3':
    loadObject("../Objs/teapot.obj");
    break;
```

```
//Translations
case 'x':
    xTranslate += 0.1;
    break;
case 'X':
    xTranslate -= 0.1;
    break;
case 'y':
    yTranslate += 0.1;
    break;
case 'Y':
    yTranslate -= 0.1;
    break;
case 'z':
    zTranslate += 0.1;
    break;
case 'Z':
    zTranslate -= 0.1;
    break;

//Enable 360 Degree Rotation for Ortho View
case 'o':
    if (zIndexEnabled) { zIndexEnabled = 0; }
    else { zIndexEnabled = 1; }
    break;

//Reset
case 'r':
    xAngle = 0;
    yAngle = 0;
    zAngle = 0;
    xTranslate = 0;
    yTranslate = 0;
    zTranslate = 0;
    scale();
    break;
```

```
//Display Types
case 'a':
    displayType = POINT;
    break;
case 's':
    displayType = VECTOR;
    break;
case 'd':
    displayType = FACES;
    break;

//Toggle Axis
case 'i':
    if (axis == AXIS_ON) { axis = AXIS_OFF; }
    else { axis = AXIS_ON; }
    break;

//Toggle Perspective
case 'p':
    if (viewMode == VIEW_PROJECTION) { viewMode = VIEW_ORTHOGONAL; }
    else { viewMode = VIEW_PROJECTION; }
    break;

//Scale
case 'm':
    scaler += 0.01;
    break;

case 'n':
    scaler -= 0.01;
    break;

//Zooms
case 43://+
    if (viewMode == VIEW_PROJECTION) { fieldOfView--; }
    else { dim -= 0.05; }
    break;

case 45://-
    if (viewMode == VIEW_PROJECTION) { fieldOfView++; }
    else { dim += 0.05; }
    break;
```

Object Rotation and Arrow Keys

```
/*Mouse Movement*/
void myMotion(int x, int y) {
    if (isPressed)
    {
        //Rotate Algs, Note Z is only used for Ortho Z-Enabled View
        yAngle += 0.01*(x - oldX);
        xAngle += 0.01*(y - oldY);
        zAngle += 0.001*((x - oldX)*(y - oldY));

        xAngle = fmod(xAngle, 360);
        yAngle = fmod(yAngle, 360);
        zAngle = fmod(zAngle, 360);
    }
    project();
    glutPostRedisplay();
}
```

```
/*Mouse Actions*/
void mouseActions(int button, int state, int x, int y) {

    switch (button) {
    case GLUT_LEFT_BUTTON:
        //Keep track of Coords when Rotating
        if (state == GLUT_DOWN){
            oldX = x;
            oldY = y;
            isPressed = true;
        }
        break;
    default:
        isPressed = false;
        break;
}

    project();
    glutPostRedisplay();
}
```

```
/*Arrow Key Functionality to Move the Display for the Object*/
/*Used for Precise Rotation!*/
void windowSpecial(int key, int x, int y) {
    printf("Special Key is: %d\n", key);
    switch (key) {
    case GLUT_KEY_RIGHT:
        yAngle += 5;
        break;
    case GLUT_KEY_LEFT:
        yAngle -= 5;
        break;
    case GLUT_KEY_UP:
        xAngle += 5;
        break;
    case GLUT_KEY_DOWN:
        xAngle -= 5;
        break;
    default:break;
    }

    xAngle = fmod(xAngle, 360);
    yAngle = fmod(yAngle, 360);

    project();
    glutPostRedisplay();
}
```

Pushing the arrows keys will rotate our objects a set amount. Clicking and dragging will rotate our objects more and quicker as you drag the mouse.

The Output

The following are samples of outputs. Note how the grid helps us see how the objects was centered and how the points, vectors, and faces are displayed. Also notice how we can move our objects in space to get a better perspective of how the object is built.

I did not bump into any remaining issues. Future possibilities would have been to add a light source so that we can see the faces on objects much better.