Assignment 3b

CS4610

Michael Rallo – msr5zb – 12358133

3/23/2017

Assignment 3b

The *purpose* of this assignment was to familiarize yourself with the OpenGL command of the texture mapping.

Objectives:

1. Read a obj file cube.obj and an image file (.jpg) as a texture (here are two sample images: checker-board, mandrill, you can also use any other images you like) , and render the object with the texture mapped.
2. The texture should be modulated by the shading computation.

Approach: By adding the stb_image.h library, I was able to load in textures. Once an image file was loaded in, values for the texture mapping/binding were implemented. The most difficult part of this assignment was coming up with an algorithm to calculate the glTexCoord2f() parameter values. I developed an algorithm that looks at planes a face's vertices are on and then computes appropriate coordinates for the texture to object map.

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.

stb_image.h was also added in to handle image loading and manipulation.

```
#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 3a. 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;
enum AspectRatioType {AUTO, CUSTOM};
AspectRatioType aspectRatioType = AUTO;
double aspectRatio = 1;
double scaler = 1;

/*View and Positioning*/
int fieldOfView = 70;
double near = 4;
double far = 4;
double xAngle = 0;
double yAngle = 0;
double zAngle = 0;
double xTranslate = 0;
double yTranslate = 0;
double zTranslate = 0;

/*Dragging*/
int xMouse;
int yMouse;
int oldX;
int oldY;
int dragging = 0;
bool isPressed = false;
```

```
/*Light Stuff*/
enum LightMode{ON, OFF};
LightMode light0Mode = ON;
LightMode light1Mode = ON;
double lightDistance = 5;
float lightY = 0;
double lightX = 90;
double light2X = 45;
double lightAmbient = 35;
double lightDiffuse = 100;
double lightSpecular = 0;
double lightEmission = 0;
float lightShininess = 0;
float globalAmbientLight = 15;
float shinyVector[1];
double materialAmbient = 35;
double materialDiffuse = 100;
double materialSpecular = 0;
float white[] = { 1.0, 1.0, 1.0, 1.0 };
float red[] = { 1.0, 0.0, 0.0, 1.0 };
float green[] = { 0.0, 1.0, 0.0, 1.0 };
float blue[] = { 0.0, 0.0, 1.0, 1.0 };
float yellow[] = { 1.0, 1.0, 0.0, 1.0 };
float purple[] = { 1.0, 0.0, 1.0, 1.0 };
float cyan[] = { 0.0, 1.0, 1.0, 1.0 };

/*Project Globals*/
std::vector<GLfloat*> vertices;
std::vector<GLfloat*> fnormals;
std::vector<GLfloat*> vnormals;
std::vector<GLint*> faces;
double maxX = 0, maxY = 0, maxZ = 0, minX = 0, minY = 0, minZ = 0;

/*Display Types*/
enum DisplayType { POINT, VECTOR, FACES };
DisplayType displayType = VECTOR;

enum ShadeType{FLAT, SMOOTH};
ShadeType shadeType = SMOOTH;
```

For this assignment, I will be using global variables in order to change various settings (named conventionally). Note the fieldOfView, near, and far to handle clipping and camera view. Lighting variables for the lights, environment, and objects were also created. Positions for lights will also be manipulated so we can see the full effects of lighting in our program.

Drawing the XYZ Grid (Extras)

```
/*Draws the 3d Axis Grid to the Screen*/
void drawAxis() {
    if (axis == AXIS_ON) {
        glDisable(GL_LIGHTING);
        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();
        if (light0Mode == ON || light1Mode == ON) {
            glEnable(GL_LIGHTING);
        }
    }
}
```

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.

Loading the File (loadObject)

```
/*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");
        return;
    }
    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') {
            //Set Vertices
            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);
    normalize();
    scale();
}
```

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;
}
```

LightSources

This is our function to create our light sources. Note to adjustable RGBA values for each light. Also note how we use the scalar on the light source to keep a constant size throughout objects. Light Sources are toggle-able as well.

```
void drawLight() {
    if (light0Mode == ON || light1Mode == ON) {

        //Light Properties
        float Ambient0[] = { 0.01*lightAmbient, 0.0, 0.0, 1.0 };
        float Diffuse0[] = { 0.01*lightDiffuse, 0.0, 0.0, 1.0 };
        float Specular0[] = { 0.01*lightSpecular, 0.0, 0.0, 1.0 };
        float Position0[] = { (lightDistance*Sin(lightX)) / (scaler*1.7), (lightY) / (scaler*1.7), (lightDistance*Cos(lightX)) / (scaler*1.7), 1.0 / (scaler * 4) };

        float Ambient1[] = { 0.0, 0.0*lightAmbient, 0.01, 1.0 };
        float Diffuse1[] = { 0.0, 0.01*lightDiffuse, 0.0, 1.0 };
        float Specular1[] = { 0.0, 0.01*lightSpecular, 0.0, 1.0 };
        float Position1[] = { -(lightDistance*Sin(lightX)) / (scaler*1.7), (lightY) / (scaler*1.7), (lightDistance*Cos(lightX)) / (scaler*1.7), 1.0 / (scaler * 4) };
```

```
    //Light 1
    if (light0Mode == ON) {
        glDisable(GL_LIGHTING);
        glColor3fv(red);
        sphere(Position0[0], Position0[1], Position0[2], Position0[3], 0);
        glEnable(GL_LIGHTING);

        glEnable(GL_LIGHT0);
        glLightfv(GL_LIGHT0, GL_SPECULAR, Specular0);
        glLightfv(GL_LIGHT0, GL_AMBIENT, Ambient0);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, Diffuse0);
        glLightfv(GL_LIGHT0, GL_POSITION, Position0);
    }
    else {
        glDisable(GL_LIGHT0);
    }

    //Light 2
    if (light1Mode == ON) {
        glDisable(GL_LIGHTING);
        glColor3fv(green);
        sphere(Position1[0], Position1[1], Position1[2], Position1[3], 0);
        glEnable(GL_LIGHTING);

        glEnable(GL_LIGHT1);
        glLightfv(GL_LIGHT1, GL_SPECULAR, Specular1);
        glLightfv(GL_LIGHT1, GL_AMBIENT, Ambient1);
        glLightfv(GL_LIGHT1, GL_DIFFUSE, Diffuse1);
        glLightfv(GL_LIGHT1, GL_POSITION, Position1);
    }
    else {
        glDisable(GL_LIGHT1);
    }

}
else {
    glDisable(GL_LIGHTING);
}
```

LightSource Spheres

This function simply puts a sphere where the light source radiates from.

```
//Vertex Helper function for Spheres
void vertex(double th, double ph) {
    double x = Sin(th)*Cos(ph);
    double y = Cos(th)*Cos(ph);
    double z =        Sin(ph);
    glNormal3d(x, y, z);
    glVertex3d(x, y, z);
}

//Spheres to Represent Lights
void sphere(double x, double y, double z, double r, double rot) {
    int th, ph;
    float yellow[] = { 1.0, 1.0, 0.0, 1.0 };

    glMaterialfv(GL_FRONT, GL_SHININESS, shinyVector);
    glMaterialfv(GL_FRONT, GL_SPECULAR, yellow);

    glPushMatrix();

    glTranslated(x, y, z);
    glScaled(r, r, r);
    glRotated(rot, 0, 1, 0);

    for (ph = -90; ph < 90; ph += 5) {
        glBegin(GL_QUAD_STRIP);
        for (th = 0; th <= 360; th += 2 * 5) {
            vertex(th, ph);
            vertex(th, ph + 5);
        }
        glEnd();
    }
    glPopMatrix();
}
```

Normal Vectors

As described above, this is our ‘normalize’ function that calculates the normal vectors for all vertexes.

```
void normalize() {
    vnormals.clear();
    std::vector<GLfloat*> uVector, vVector;
    GLfloat *uArray, *vArray, *nArray, *nvArray;

    //Calculate Vertex Normals!
    for (GLfloat vertex = 0; vertex < vertices.size(); vertex++) {

        //Calculate Face normals of all Faces that includes our vertex
        fnormals.clear();
        for (int i = 0; i < faces.size(); i++) {

            if ((faces[i][0] - 1) == vertex || (faces[i][1] - 1) == vertex || (faces[i][2] - 1) == vertex) {
                //Cauclate Normal Vector
                uArray = new GLfloat[3];
                uArray[0] = vertices[faces[i][1] - 1][0] - vertices[faces[i][0] - 1][0];
                uArray[1] = vertices[faces[i][1] - 1][1] - vertices[faces[i][0] - 1][1];
                uArray[2] = vertices[faces[i][1] - 1][2] - vertices[faces[i][0] - 1][2];
                uVector.push_back(uArray);

                vArray = new GLfloat[3];
                vArray[0] = vertices[faces[i][2] - 1][0] - vertices[faces[i][0] - 1][0];
                vArray[1] = vertices[faces[i][2] - 1][1] - vertices[faces[i][0] - 1][1];
                vArray[2] = vertices[faces[i][2] - 1][2] - vertices[faces[i][0] - 1][2];
                vVector.push_back(vArray);

                nArray = new GLfloat[3];
                nArray[0] = (uArray[1] * vArray[2]) - (uArray[2] * vArray[1]);
                nArray[1] = (uArray[2] * vArray[0]) - (uArray[0] * vArray[2]);
                nArray[2] = (uArray[0] * vArray[1]) - (uArray[1] * vArray[0]);

                //Divide Normal by Magnitude
                double mag = sqrt((nArray[0] * nArray[0]) + (nArray[1] * nArray[1]) + (nArray[2] * nArray[2]));
                nArray[0] = nArray[0] / mag;
                nArray[1] = nArray[1] / mag;
                nArray[2] = nArray[2] / mag;

                fnormals.push_back(nArray);
            }
        }
        //Average by Adding and Dividing by Magnitude
        nvArray = new GLfloat[3];
        nvArray[0] = 0;
        nvArray[1] = 0;
        nvArray[2] = 0;
        for (int f = 0; f < fnormals.size(); f++) {
            nvArray[0] += fnormals[f][0];
            nvArray[1] += fnormals[f][1];
            nvArray[2] += fnormals[f][2];
        }
        double mag = sqrt((nvArray[0] * nvArray[0]) + (nvArray[1] * nvArray[1]) + (nvArray[2] * nvArray[2]));
        nvArray[0] = nvArray[0] / mag;
        nvArray[1] = nvArray[1] / mag;
        nvArray[2] = nvArray[2] / mag;
        vnormals.push_back(nvArray);
    }
}
```

Display

Our display function has been updated to only handle perspective camera view. Important variables are also set here (depending on flags set). One of which is the Shadetype(GL_SMOOTH/GL_FLAT). Also notice the adjusted material values put in place for the objects to be created.

```
void display() {

    //Display Variables
    float globalAmbientLightArray[4] = { 0.01*globalAmbientLight, 0.01*globalAmbientLight, 0.01*globalAmbientLight, 1.0};
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, globalAmbientLightArray);

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);

    if (shadeType == SMOOTH){glShadeModel(GL_SMOOTH);}
    else {glShadeModel(GL_FLAT);}

    glLoadIdentity();

    //Perspective/Camera View
    double Ex = -2 * dim*Sin(yAngle)*Cos(xAngle);
    double Ey = +2 * dim            *Sin(xAngle);
    double Ez = +2 * dim*Cos(yAngle)*Cos(xAngle);
    gluLookAt(Ex, Ey, Ez, 0, 0, 0, Cos(xAngle), 0);


    //Draw Axis
    drawAxis();
    glPointSize(4);

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

    //Light
    drawLight();

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

    //Material Stuffs
    glColor3f(1.0, 1.0, 1.0);
    GLfloat materialAmbientArray[] = { 0.0, 0.0, 0.01*materialAmbient, 1.0 };
    GLfloat materialDiffuseArray[] = { 0.0, 0.0, 0.01*materialDiffuse, 1.0 };
    GLfloat materialSpecularArray[] = { 0.0, 0.0, 1.01*materialSpecular, 1.0 };
    GLfloat shine = 100.0;

    glMaterialfv(GL_FRONT, GL_AMBIENT, materialAmbientArray);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, materialDiffuseArray);
    glMaterialfv(GL_FRONT, GL_SPECULAR, materialSpecularArray);
    glMaterialf(GL_FRONT, GL_SHININESS, shine);
```

```
case FACES:
    loadTexture();
    glEnable(GL_NORMALIZE);
    glBegin(GL_TRIANGLES);

    for (int i = 0; i < faces.size(); i++) {

        glNormal3fv(vnormals[faces[i][0] - 1]);
        handleTexCoord(i, vertices[faces[i][0] - 1]);
        glVertex3fv(vertices[faces[i][0] - 1]);

        glNormal3fv(vnormals[faces[i][1] - 1]);
        handleTexCoord(i, vertices[faces[i][1] - 1]);
        glVertex3fv(vertices[faces[i][1] - 1]);

        glNormal3fv(vnormals[faces[i][2] - 1]);
        handleTexCoord(i, vertices[faces[i][2] - 1]);
        glVertex3fv(vertices[faces[i][2] - 1]);

    }
    glEnd();
```

Note the handleTexCoord() call. This function is described on the next page. Also note the shading computation.

HandleTexCoord

```
void handleTexCoord(int faceIndex, GLfloat* workingVertex){
    //Given Three Vertexes, find which position it is in (Bottom Left, Bottom Right, Top Left, Top Right)
    //Also note images are loaded in upside-down
    GLfloat* v0 = vertices[faces[faceIndex][0] - 1];
    GLfloat* v1 = vertices[faces[faceIndex][1] - 1];
    GLfloat* v2 = vertices[faces[faceIndex][2] - 1];

    if(workingVertex == v0) {
        //XY Plane
        if (v0[2] == v1[2] && v0[2] == v2[2]){
            //Top Right
            if((v0[1] - v1[1] > 0 || v0[1] - v2[1] > 0) && (v0[0] - v1[0] > 0 || v0[0] - v2[0] > 0)){glTexCoord2f(0, 0);}
            //Top Left
            if((v0[1] - v1[1] > 0 || v0[1] - v2[1] > 0) && (v0[0] - v1[0] < 0 || v0[0] - v2[0] < 0)){glTexCoord2f(1, 0);}
            //Bottom Right
            if((v0[1] - v1[1] < 0 || v0[1] - v2[1] < 0) && (v0[0] - v1[0] > 0 || v0[0] - v2[0] > 0)){glTexCoord2f(0, 1);}
            //Bottom Left
            if((v0[1] - v1[1] < 0 || v0[1] - v2[1] < 0) && (v0[0] - v1[0] < 0 || v0[0] - v2[0] < 0)){glTexCoord2f(1, 1);}
        }

        //XZ Plane
        else if (v0[1] == v1[1] && v0[1] == v2[1]) {
            //Top Right
            if ((v0[0] - v1[0] > 0 || v0[0] - v2[0] > 0) && (v0[2] - v1[2] > 0 || v0[2] - v2[2] > 0)) { glTexCoord2f(0, 0); }
            //Top Left
            if ((v0[0] - v1[0] > 0 || v0[0] - v2[0] > 0) && (v0[2] - v1[2] < 0 || v0[2] - v2[2] < 0)) { glTexCoord2f(1, 0); }
            //Bottom Right
            if ((v0[0] - v1[0] < 0 || v0[0] - v2[0] < 0) && (v0[2] - v1[2] > 0 || v0[2] - v2[2] > 0)) { glTexCoord2f(0, 1); }
            //Bottom Left
            if ((v0[0] - v1[0] < 0 || v0[0] - v2[0] < 0) && (v0[2] - v1[2] < 0 || v0[2] - v2[2] < 0)) { glTexCoord2f(1, 1); }
        }

        //YZ Plane
        else if (v0[0] == v1[0] && v0[0] == v2[0]) {
            //Top Right
            if ((v0[1] - v1[1] > 0 || v0[1] - v2[1] > 0) && (v0[2] - v1[2] > 0 || v0[2] - v2[2] > 0)) { glTexCoord2f(0, 0); }
            //Top Left
            if ((v0[1] - v1[1] > 0 || v0[1] - v2[1] > 0) && (v0[2] - v1[2] < 0 || v0[2] - v2[2] < 0)) { glTexCoord2f(1, 0); }
            //Bottom Right
            if ((v0[1] - v1[1] < 0 || v0[1] - v2[1] < 0) && (v0[2] - v1[2] > 0 || v0[2] - v2[2] > 0)) { glTexCoord2f(0, 1); }
            //Bottom Left
            if ((v0[1] - v1[1] < 0 || v0[1] - v2[1] < 0) && (v0[2] - v1[2] < 0 || v0[2] - v2[2] < 0)) { glTexCoord2f(1, 1); }
        }

        //Multi-Planar
        else { glTexCoord2f(0, 0); }

    }
    else if (workingVertex == v1) {
        //XY Plane
```

This function was used to calculate the appropriate glTexCoords() for a given face. It compares vertices via planes and then calculates a triangle that closest represents its intended texturing map. Note images in OpenGL are loaded in upside down – which was taken into account in the algorithm.

Loading the Texture

```
int loadTexture() {
    int x, y, n;
    int force_channels = 4;
    unsigned char* image_data;
    image_data = stbi_load(textures[currentTexture], &x, &y, &n, force_channels);
    if (image_data) {
        static GLuint texName;
        glEnable(GL_TEXTURE_2D);

        glGenTextures(1, &texName);
        glBindTexture(GL_TEXTURE_2D, texName);

        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, x, y, 0, GL_RGBA, GL_UNSIGNED_BYTE, image_data);

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        // select modulate to mix texture with color for shading
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
        free(image_data);
        return texName;
    }
    else {
        printf("Failed &n\n");
        return 0;
    }

}
void initializeTextures() {

    textures[0] = "../Textures/checker.jpg";
    textures[1] = "../Textures/mandrill.jpg";
    currentTexture = 0;
}
```

Using the stb_image.h function stbi_load() I was able to easily load the data of a desired image. I kept the different textures in an array so that user will be able to choose from multiple textures.

Reshape

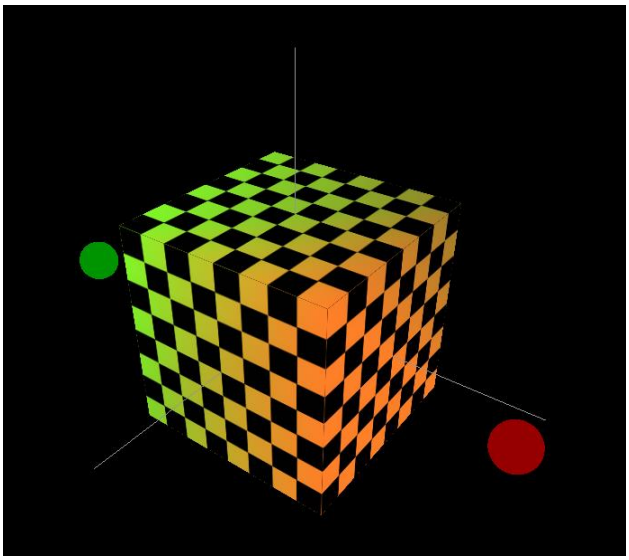
```
/*Display Details*/
void project() {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    //Adjust according to ViewMode Active
    gluPerspective(fieldOfView, aspectRatio, dim / near, far * dim);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

/*Updates Display, Keeping Aspect Ratio if Window is changed*/
void reshape(int width, int height) {

    if (aspectRatioType == AUTO) {
        aspectRatio = (height > 0) ? (double)width / height : 1;
    }
    glViewport(0, 0, width, height);
    project();
}
```

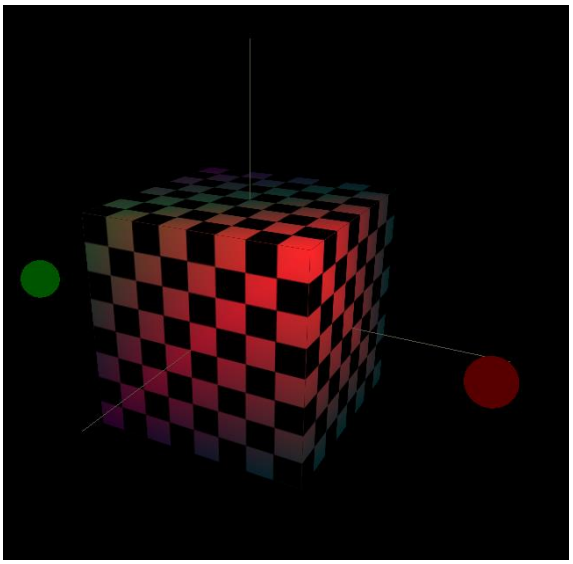
Our reshape function is still very basic and has been redesigned to only handle projection/perspective mode rather than offering an orthographic option. Also note the Aspect Ratio Adjustability.

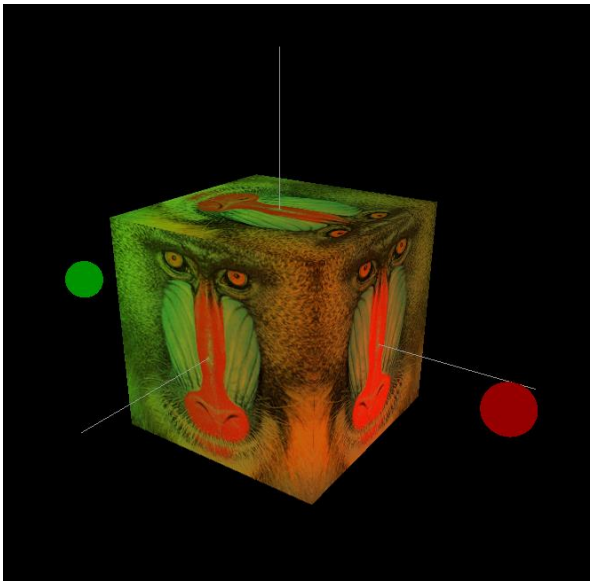
Keyboard Input
Key: Esc exits the program.
Keys: 1,2,3 loads in different Objects.
Keys: a,s,d changes display type.
Key: 4 changes textures of objects.
Key: i toggles the XYZ grid.
Keys: +,- Zooms In/Out (field of view).
Keys: b,B Scales Objects.
Keys: t,T Changes the Aspect Ratio.
Keys: g,G controls the clipping range for the near value.
Keys: f,F controls the clipping range for the far value.
Keys: l,L toggles lights 1 and 2.
Keys: <,> rotates the lights.
Keys: [,],{,} moves lights.
Keys: o,O adjusts the Global Ambience.
Keys: h,H adjusts lightAmbient.
Keys: j,J adjusts the lightDiffuse.
Keys: k,K adjusts the lightSpecular.
Keys: n,N adjusts the materialAmbient.
Keys: c,C adjusts the materialDiffuse.
Keys: v,V adjusts the materiaSpecular.
Key: q toggles the ShadeType.
Keys: x,X,y,Y,z,Z Translates objects.
Key: r resets the object.
Arrow Keys adjusts/rotates object/view incrementally.
Click and Drag to Rotate Object, further you drag quicker it rotates.

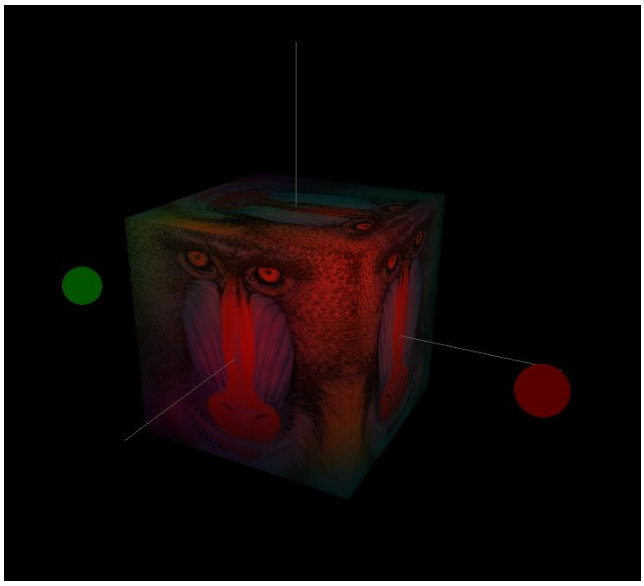
The Output

The following are samples of outputs. Note how the Spheres represent the lightsources. Also note how the smoothing technique drastically affects how our objects looks. Adjusting the ambience, diffuse, and specular yields us very creative and interesting results as well! Textured objects still keep shading properties. Also, because textures are loaded into OpenGL upside-down, I flipped the glTexCoords so they display correctly.









Everything ran pretty smoothly, however coming up with the glTexCoords was pretty difficult.