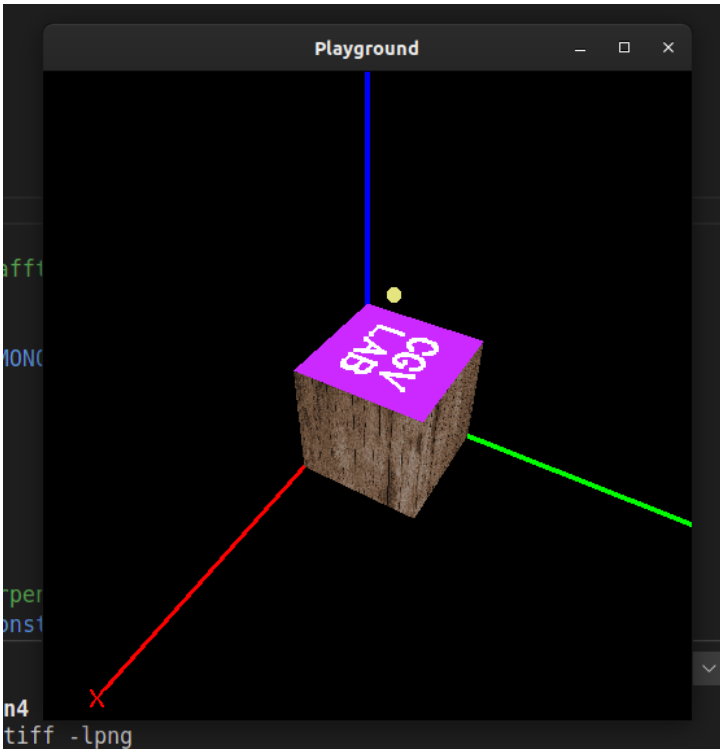
CS352 Assignment 3
3D Textured cube with Camera rotation and Lighting

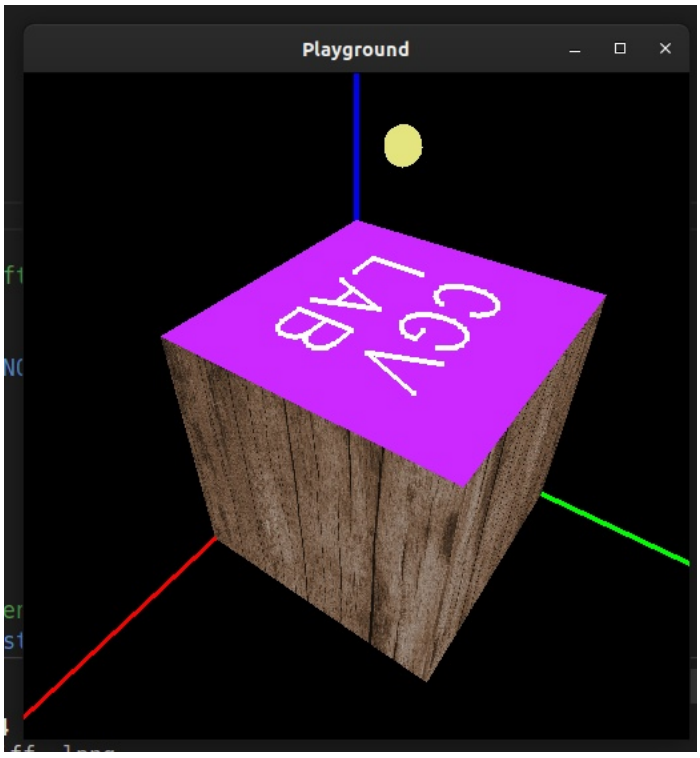
Ayush Agrawal
180001011
11-05-2021

ScreenShots:



Initial View of Cube.
X,Y,Z axis are labelled.

Yellow Dot represents
Light Source

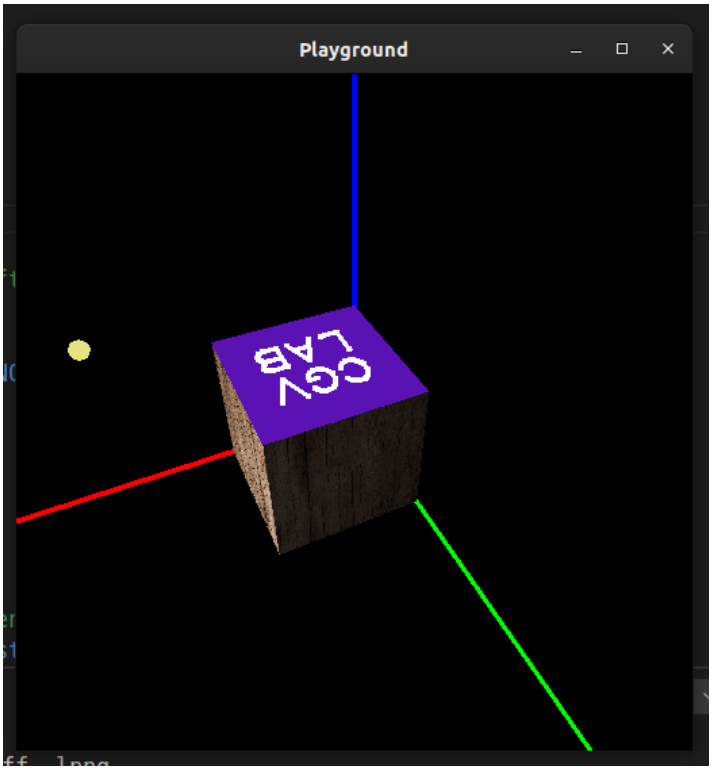


Zoomed – in view

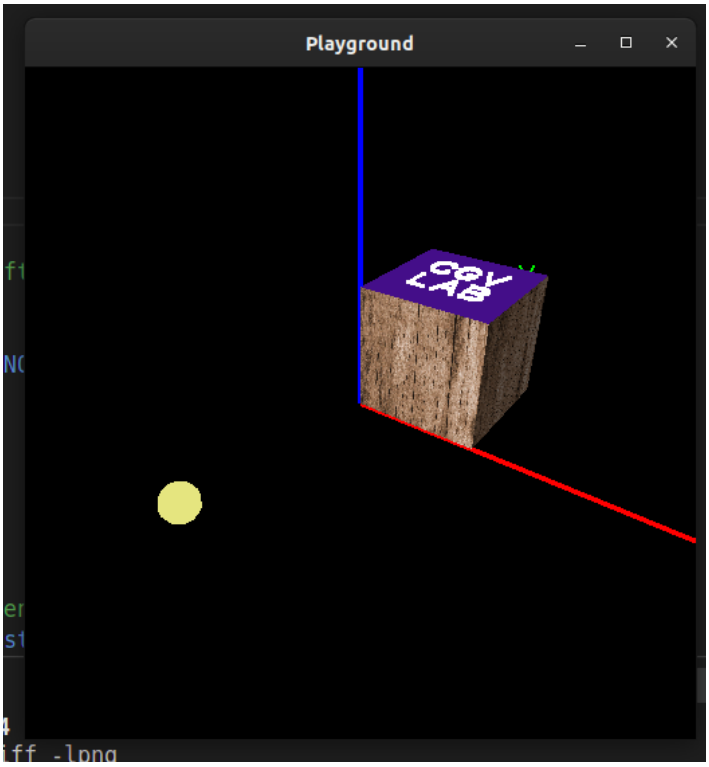
Yellow Dot represents
Light Source

When Light Source is in

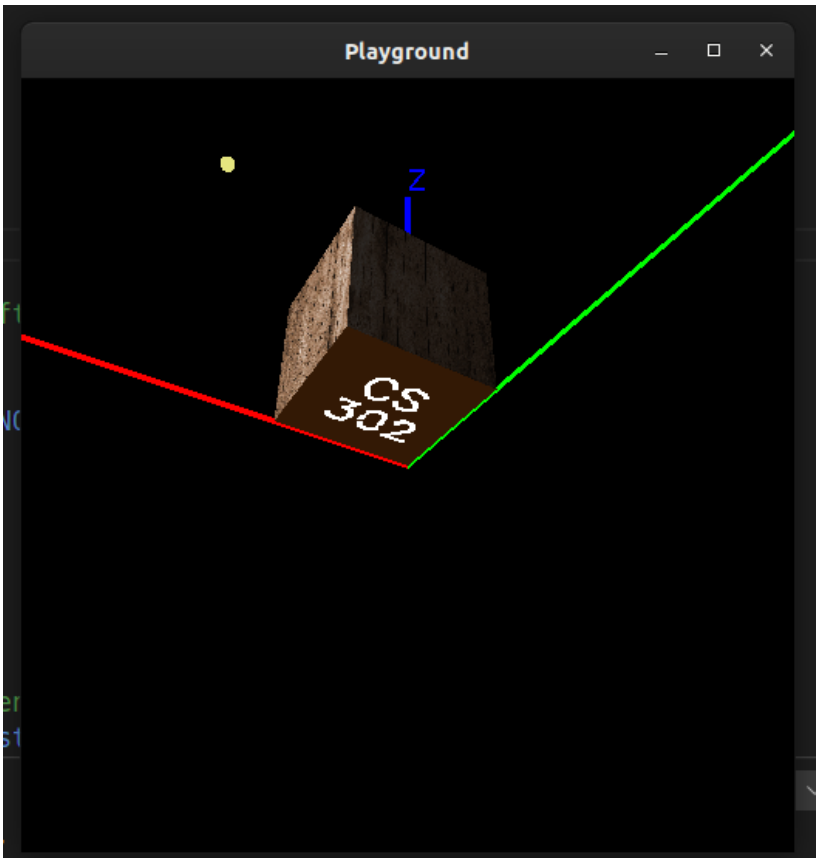
X-Direction, the corresponding Face is lit up and the other Y-axis (green) is dimmed.



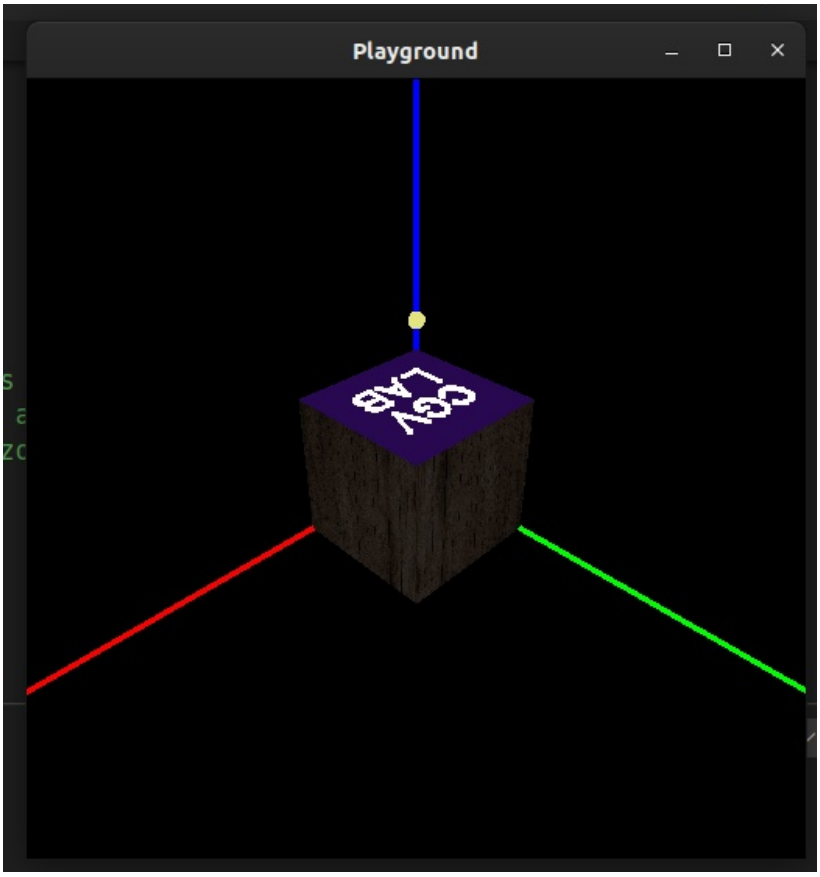
When Light moves far away, The top surface also dims



When Light is in over Z axis
The Lower face is dim



One with minimum Light
Intensity (Brightness)



Code:-
Broken into 3 files,
imageio.h and **imageio.cpp** - Sir Shared these files, so I have not pasted the code here.

Last file is **Q1.cpp:**

```
#include <GL/glut.h>
#include <cmath>
#include <iostream>
#include "imageio.h"

using namespace std;

// Window Properties
const double WIDTH = 500;
const double HEIGHT = 500;

// For Keyboard (x,y,z) key Rotations
int rotate_x = 0;
int rotate_y = 0;
int rotate_z = 0;

// Position of eye in glLookAt function
double eyeX = 3;
double eyeY = 3;
double eyeZ = 3;

// Will be used to calculate mouse-drag distance.
int old_x = 0;
int old_y = 0;
int valid = 0;

// The distance of eye from origin.
// when moving the camera, this remains constant.
double viewRadius = sqrt(27);

// 3d coords govern the movement of camera.
// drag distance is converted to angle.
double theta = 45;
double azimuthal = 45;
// then new eye positions are calculated.

// Position of light.
GLfloat light_x = 0.5;
GLfloat light_y = 0.5;
GLfloat light_z = 1.5;

// Intensity of diffused light
GLfloat light_intensity = 0.5;

// how much does light remain strong
GLfloat light_attenuate = 1.0;

// window ID
int window;

// List of the 8 vertices of the cube
double vertices[8][3] = {
    {0, 0, 0},
```

```
    {0, 0, 1},
    {1, 0, 1},
    {1, 0, 0},
    {0, 1, 0},
    {0, 1, 1},
    {1, 1, 1},
    {1, 1, 0},
};

// array of frequently used colors.
GLfloat colors[8][3] = {
    {0.0,0.0,0.0}, // black
    {1.0,0.0,0.0}, // red
    {0.0,1.0,0.0}, // green
    {0.0,0.0,1.0}, // blue
    {1.0,1.0,0.0}, // yellow
    {1.0,0.0,1.0}, // pink
    {0.0,1.0,1.0}, // cyan
    {1.0,1.0,1.0}, // white
};

// List of vertex that constitute a particular face
int ffront[4]   = {1, 2, 6, 5};
int fback[4]    = {0, 3, 7, 4};
int ftop[4]     = {4, 5, 6, 7};
int fbottom[4]  = {0, 1, 2, 3};
int fleft[4]    = {0, 1, 5, 4};
int fright[4]   = {2, 3, 7, 6};

// The normals of each face.
// Used for reflection of light
float nfront[4]   = {0, 0, 1};
float nback[4]    = {0, 0, -1};
float ntop[4]     = {0, 1, 0};
float nbottom[4]  = {0, -1, 0};
float nright[4]   = {1, 0, 0};
float nleft[4]    = {-1, 0, 0};

// number of textures, and respective files.
// Wooden texture covers 4 sides.
const int nTextures = 1;
char files[nTextures][37] = {"images/wood_hd.png"};
GLuint handles[nTextures];

// Hold the size of png opened.
int texImageWidth, texImageHeight;

// Constants that will be written on back face.
char bTextL1[] = "CS";
char bTextL2[] = "302";

// Constants that will be written on back face.
char fTextL1[] = "CGV";
char fTextL2[] = "LAB";


// Helper function.
double deg2rad(double deg){
    return (deg * M_PI)/(double)180;
```

```
}

// Opens png files, and returns the pixel matrix.
GLubyte *makeTexImage( char *loadfile ) {
   int i, j, c, width, height;
   GLubyte *texImage;

   texImage = loadImageRGBA( (char *) loadfile, &width, &height);
   texImageWidth = width;
   texImageHeight = height;

   return texImage;
}

// Recalculate Eye position after mouse drag.
void rotateEyes(double dx, double dy){
    azimuthal += dx;
    theta += dy;

    if( azimuthal >= 360 ) azimuthal -= 360;
    if( theta >= 360 ) theta -= 360;

    // 3D coordinate formalae.
    eyeX = viewRadius * cos(deg2rad(azimuthal)) * sin(deg2rad(theta));
    eyeY = viewRadius * sin(deg2rad(azimuthal)) * sin(deg2rad(theta));
    eyeZ = viewRadius * cos(deg2rad(theta));
}

// can also rotate using arrow keys.
void specialKeys( int key, int x, int y ) {
    if (key == GLUT_KEY_RIGHT)
        rotate_y += 5;
    else if (key == GLUT_KEY_LEFT)
        rotate_y -= 5;
    else if (key == GLUT_KEY_UP)
        rotate_x += 5;
    else if (key == GLUT_KEY_DOWN)
        rotate_x -= 5;
    glutPostRedisplay();
}

// All keyboard keys handler.
// Handles rotation from keys x,y,z & X,Y,Z
// Handles light movement from keys i,j,k & I,J,K
// Handles light intensity from keys +,-
// Handles light attenuation from keys <,>
// Exit window using esc key.
void keyboard_func(unsigned char key, int x, int y ) {


    switch(key) {
        case 'x':
            rotate_x = ( rotate_x + 3 ) % 360;
            break;
        case 'X':
            rotate_x = ( rotate_x - 3 ) % 360;
            break;
        case 'y':
            rotate_y = ( rotate_y + 3 ) % 360;
```

```
            break;
        case 'Y':
            rotate_y = ( rotate_y - 3 ) % 360;
            break;
        case 'z':
            rotate_z = ( rotate_z + 3 ) % 360;
            break;
        case 'Z':
            rotate_z = ( rotate_z - 3 ) % 360;
            break;

        case 'j':
            light_y += 0.1;
            break;
        case 'k':
            light_z += 0.1;
            break;
        case 'i':
            light_x += 0.1;
            break;

        case 'J':
            light_y -= 0.1;
            break;
        case 'K':
            light_z -= 0.1;
            break;
        case 'I':
            light_x -= 0.1;
            break;

        case '+':
            light_intensity = min(light_intensity + 0.1, 1.0);
            break;
        case '-':
            light_intensity = max(light_intensity - 0.1, 0.0);
            break;

        case '<':
            light_attenuate /= 0.9;
            break;
        case '>':
            light_attenuate *= 0.9;
            break;

        case 27: /* escape */
            glutDestroyWindow(window);
            exit(0);
    }
    glutPostRedisplay();
}


// Draw Quadrilateral with given bg color, and texture vertices mapping.
void drawTextureFace(int verts[], float normal[], int cId){
    // Apply white colors so that does not mess with texture.
        glColor3fv(colors[cId]);
    glBegin(GL_QUADS);
        glNormal3fv(normal);
```

```
        glTexCoord2f(0.0, 0.0); glVertex3dv(vertices[verts[0]]);
        glTexCoord2f(1.0, 0.0); glVertex3dv(vertices[verts[1]]);
        glTexCoord2f(1.0, 1.0); glVertex3dv(vertices[verts[2]]);
        glTexCoord2f(0.0, 1.0); glVertex3dv(vertices[verts[3]]);
    glEnd();
}

// Draw Quadrilateral with given bg color, without texture vertices mapping.
void drawColorFace(int verts[], float normal[], int cId){
        glColor3fv(colors[cId]);
    glBegin(GL_QUADS);
        glNormal3fv(normal);
        glVertex3dv(vertices[verts[0]]);
        glVertex3dv(vertices[verts[1]]);
        glVertex3dv(vertices[verts[2]]);
        glVertex3dv(vertices[verts[3]]);
    glEnd();
}

// Write text, starting at specified position.
// the text plain is parallel to xy plane.
void strokef(GLfloat x, GLfloat y, GLfloat z, char *format, ...) {
    if(z == 1) z += 0.01;

    // Handle variable arguments (printf like functionality)
    va_list args;
    char buffer[200], *p;

    va_start(args, format);
    vsprintf(buffer, format, args);
    va_end(args);

    // Save the current config.
    glPushMatrix();
    glTranslatef(x, y, z);
        //
        if(z == 0) {
        glRotatef(180, 0, 1, 0);
        z = -0.01;
    }
    // Used to adjust font size.
    glScalef(0.002, 0.002, 0.002);
    glPointSize(4);
    glLineWidth(4);

    // Write text in white color
    glColor3fv(colors[7]);

    // disable lighting so that is does not afftect font color.
    glDisable(GL_LIGHTING);
    for (p = buffer; *p; p++){
            glutStrokeCharacter(GLUT_STROKE_MONO_ROMAN, *p);
        }
    glEnable(GL_LIGHTING);

    // Reload the initial configuration
    glPopMatrix();
}
```

```
// Write bitMap text, that always remains perpendicular to camera
void drawString(float x, float y, float z, const char* string) {
   glRasterPos3f(x, y, z);
   glColor3fv(colors[7]);
   for (const char* c = string; *c != '\0'; c++) {
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *c);  // Updates the position
   }
}


void display() {
   glClearColor( 0, 0, 0, 1 );
   // Enable 2d textures.
   glEnable(GL_TEXTURE_2D);

   // Use MODULATE instead of GL_DECAL so that light also affects texture faces.
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

   // Use wooden texture and draw the 4 faces.
   glBindTexture(GL_TEXTURE_2D, handles[0]);
        drawTextureFace(ftop, ntop, 7);
   drawTextureFace(fleft, nleft, 7);
   drawTextureFace(fright, nright, 7);
   drawTextureFace(fbottom, nbottom, 7);
   glDisable(GL_TEXTURE_2D);

   // Constants needed to adjust reflectivity
   GLfloat polished[] = { 100.0 };
   GLfloat dull[] = { 0.0 };

   GLfloat light_position[] = {light_x, light_y, light_z, 1.0 };

   // Ambient light also depends upon intensity
   GLfloat light_ambient[] = {light_intensity, light_intensity, light_intensity, 1.0};

   const GLfloat white[]   = {1.0, 1.0, 1.0, 0.5 };
   const GLfloat blue[]    = {0.5, 0.1, 1.0, 0.5 };
   const GLfloat orange[]  = {1.0, 0.5, 0.1, 0.5 };

        glMaterialfv(GL_FRONT, GL_SHININESS, polished);

        glMaterialfv(GL_FRONT, GL_SPECULAR, orange);
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, orange);
   drawColorFace(fback, nback, 7);
   strokef(0.7, 0.55, 0, bTextL1);
   strokef(0.8, 0.25, 0, bTextL2);
   // Text start positions are hard coded.

        glMaterialfv(GL_FRONT, GL_SPECULAR, blue);
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, blue);
   drawColorFace(ffront, nfront, 7);
   strokef(0.2, 0.55, 1, fTextL1);
   strokef(0.2, 0.25, 1, fTextL2);

   // So that light does not affect other drawings.
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, white);
   // It also prevents from colouring the textured faces.
```

```
// Add one light source
      glMatrixMode(GL_MODELVIEW);
glPushMatrix();
            glTranslatef(light_x, light_y, light_z);

   // Vary attenuation value to ~change intensity
            glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, light_attenuate);

   // Increase ambient light to simulate ~brightness functionality.
            glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
   glLightfv(GL_LIGHT0, GL_POSITION, light_position);

            glDisable(GL_LIGHTING);
            glColor3d(0.9, 0.9, 0.5);
   // Add a sphere that indicates position of light source
            glutSolidSphere(0.05, 10, 10);
            glEnable(GL_LIGHTING);
      glPopMatrix();
glEnable(GL_LIGHT0);


// Set Projection as perspective.
glMatrixMode( GL_PROJECTION );
glLoadIdentity();

gluPerspective( 60, WIDTH / HEIGHT, 0.1, 100 );

      glPushMatrix();
glMatrixMode( GL_MODELVIEW );

// Draw axis in light-disable mode to show full r,g,b colors.
glDisable(GL_LIGHTING);
glColor3d(1, 0, 0);
glBegin(GL_LINES);
   glVertex3d(0, 0, 0);
   glVertex3d(3, 0, 0);
glEnd();
drawString(3.1, 0, 0, "X");
glEnable(GL_LIGHTING);

glDisable(GL_LIGHTING);
glColor3d(0, 1, 0);
glBegin(GL_LINES);
   glVertex3d(0, 0, 0);
   glVertex3d(0, 3, 0);
glEnd();
drawString(0, 3.1, 0, "Y");
glEnable(GL_LIGHTING);

glDisable(GL_LIGHTING);
glColor3d(0, 0, 1);
glBegin(GL_LINES);
   glVertex3d(0, 0, 0);
   glVertex3d(0, 0, 3);
glEnd();
drawString(0, 0, 3.1, "Z");
glEnable(GL_LIGHTING);
```

```
    glLoadIdentity();

    // LookAt position updates with mouse drag.
    gluLookAt (
        eyeX, eyeY, eyeZ,
        0, 0, 0,
        0, 0, 1
    );

    // Do keyboard rotations.
    glRotatef( rotate_x, 1.0, 0.0, 0.0 );
    glRotatef( rotate_y, 0.0, 1.0, 0.0 );
    glRotatef( rotate_z, 0.0, 0.0, 1.0 );

    glutSwapBuffers();
    glFlush();
}

// Triggered when mouse-clicks. Stores initial x,y and whether it was RMB/LMB
void mouse_func (int button, int state, int x, int y) {
    old_x=x;
    old_y=y;

    if(button == 2){
        valid = 2;
    } else {
        valid = state == GLUT_DOWN;
    }
}

// Triggered during drag motion. calculates dx,dy.
// if LMB then updates theta and azimuthal angles.
// otherwise update camera radius to give zooming effect.
void motion_func (int x, int y) {
    if(valid == 2){
        double dx = old_x - x;
        viewRadius += dx/50;

        rotateEyes(0, 0);
        glutPostRedisplay();
    }
    if (valid == 1) {
        int dx = old_x - x;
        int dy = old_y - y;

        double rx = (double)dx/10;
        double ry = (double)dy/10;

        rotateEyes(rx, ry);
        glutPostRedisplay();
    }
    old_x = x;
    old_y = y;
}

// Maps the png images to texture handles.
void initTexture(){
    glShadeModel(GL_FLAT);
```

```
    glEnable(GL_DEPTH_TEST);

    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    //handles is global
    glGenTextures(nTextures, handles);

    // Go over every image, and map the texture to its handle.
    for(int i=0;i<nTextures;i++){
        GLubyte *texImage = makeTexImage(files[i]);

        if(!texImage) cout<<"Error opening "<<files[i]<<"\n";

        glBindTexture(GL_TEXTURE_2D, handles[i]);                   //map it to variable
        // Enable
        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_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
            texImageWidth, texImageHeight,
            0, GL_RGBA, GL_UNSIGNED_BYTE,
            texImage
        );
        delete texImage;                                            //free memory holding texture image
    }
}


int main( int argc, char **argv ){
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE );
    glutInitWindowSize( WIDTH, HEIGHT );
    glutCreateWindow( "Playground" );
    initTexture();

    glutDisplayFunc( display );
    glutSpecialFunc( specialKeys );
    glutKeyboardFunc( keyboard_func );

    glutMouseFunc( mouse_func );
    glutMotionFunc( motion_func );

    glutMainLoop();
    return 0;
}
```