Lab10 for CPT205 Computer Graphics (Texture Mapping)

There are four sample programs for this lab: **texture mapping using a procedure to create the textural pattern, mipmapping, texture mapping from an image file by pixel, and mapping images onto objects**. Please first run these programs and try to understand what is done and what functions are used. You can further practise with the functions and see the effects. Then you can start working on your Assessment 2.

1) Sample code for texture mapping where a procedure creates a textural pattern

In the sample program, the texture consisting of alternating white and black squares, like a checkerboard - is generated by the program. The program applies this texture to two squares, which are then rendered in perspective, one of them facing the viewer squarely and the other tilting back at 45 degrees.

The checkerboard texture is generated in the routine ***makeCheckImage()***, and all the texture-mapping initialisation occurs in the routine ***myinit()***. The single, full-resolution texture map is specified by ***glTexImage2D()***, whose parameters indicate the size, type, location, and other properties of the image.

The next four calls to ***glTexParameter*()*** specify how the texture is to be wrapped (see "Repeating and Clamping Textures") and how the colours are to be filtered if there is not an exact match between pixels in the texture and pixels on the screen (see "Controlling Filtering").

Next, ***glTexEnv*()*** sets the drawing mode to GL_DECAL so that the textured polygons are drawn using the colours from the texture map (rather than taking into account what colour the polygons would have been drawn without the texture). Finally, ***glEnable()*** turns on texturing.

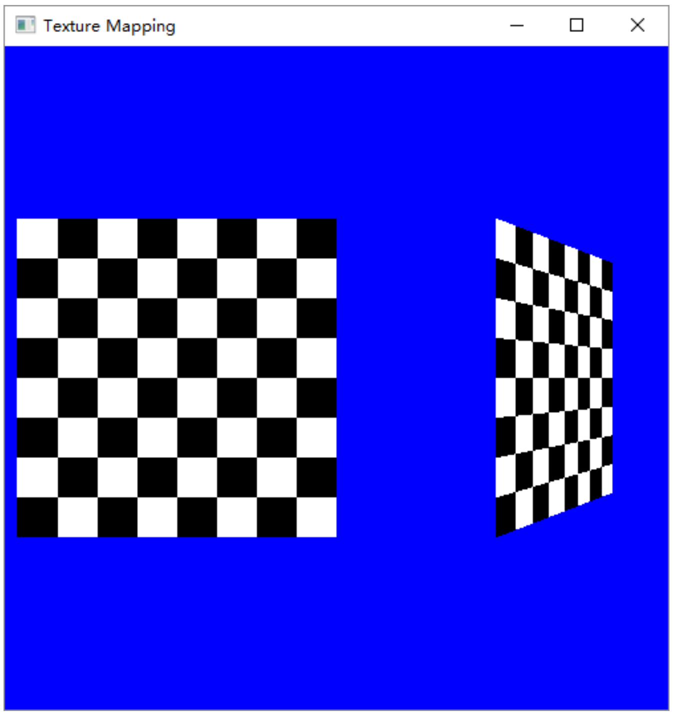
The routine ***display()*** draws the two polygons. Note that texture co-ordinates are specified along with vertex co-ordinates.

The ***glTexCoord*()*** command behaves similarly to the ***glNormal()*** command: it sets the current texture co-ordinates; any subsequent vertex command has those texture co-ordinates associated with it until ***glTexCoord*()*** is called again.

The checkerboard image on the tilted polygon might look wrong when you compile and run it on your machine - for example, it might look like two triangles with different projections of the checkerboard image on them. If so, try to set the parameter GL_PERSPECTIVE_CORRECTION_HINT to GL_NICEST and running the example again. To do this, use ***glHint()***.

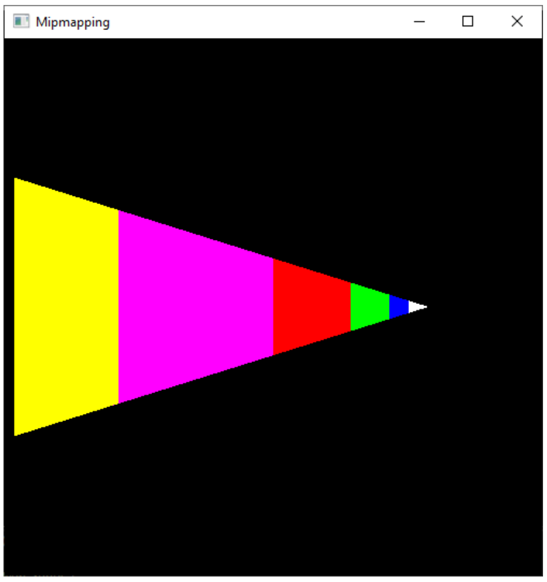
```
/*Function:
Texture mapping uses a procedure to create the textural pattern.
*The texture consisting of alternating white and black squares,
*like a checkerboard - is generated by the program. The program
*applies this texture to two squares, which are then rendered in
*perspective, one of them facing the viewer squarely and the
*other tilting back at 45 degrees.
~~*SDK doc: https://www.opengl.org/sdk/docs~~
*/

#define FREEGLUT_STATIC
#include <GL/freeglut.h>
#include <stdlib.h>
#include <stdio.h>
// define a 64*64*3 plane for checkerboard
#define checkImageWidth 64
#define checkImageHeight 64
GLubyte checkImage[checkImageWidth][checkImageHeight][3];

// make a checkerboard
void makeCheckImage(void) {
    int i, j, c;
    for (i = 0; i < checkImageWidth; i++) {
        for (j = 0; j < checkImageHeight; j++) {
            c = (((i & 0x8) == 0) ^ ((j & 0x8) == 0)) * 255;  // bitwise AND and OR operations, which
            checkImage[i][j][0] = (GLubyte)c;                 // are faster than normal calculations
            checkImage[i][j][1] = (GLubyte)c;
            checkImage[i][j][2] = (GLubyte)c;
        }
    }
}
```

```
// all the texture-mapping initialisation occurs here
void myinit(void) {
    // specify clear values for the color buffers
    // Specify the red, green, blue and alpha values used when the color buffers are cleared.
    // The initial values are all 0.
    glClearColor(0.0, 0.0, 1.0, 0.0);
    // enable or disable server-side GL capabilities
    glEnable(GL_DEPTH_TEST);
    // specify the value used for depth buffer comparisons
    // GL_LEQUAL: Passes if the incoming depth value is less than or equal to the stored depth value.
    glDepthFunc(GL_LEQUAL);
    // make a checkerboard
    makeCheckImage();
    // set pixel storage modes
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    // specify a two-dimensional texture image
    // GL_UNPACK_ALIGNMENT: Specifies the alignment requirements for the start of each pixel row
    // in memory.
    // The allowable values are 1 (byte-alignment), 2 (rows aligned to even-numbered bytes),
    // 4 (word-alignment), and 8 (rows start on double-word boundaries).
    glTexImage2D(GL_TEXTURE_2D, 0, 3, checkImageWidth, checkImageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE,
&checkImage[0][0][0]);
    // set texture parameters
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    // set texture environment parameters
    ~~// details: https://www.opengl.org/sdk/docs/man2/xhtml/glTexEnv.xml2~~
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    glEnable(GL_TEXTURE_2D);
    // select flat or smooth shading
    // details: https://www.opengl.org/sdk/docs/man2/xhtml/glShadeModel.xml
    glShadeModel(GL_FLAT);
}

// specify texture co-ordinates
void display(void) {
    // clear buffers to preset values
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // delimit the vertices of a primitive or a group of like primitives
    // GL_QUADS: Treats each group of four vertices as an independent quadrilateral
    glBegin(GL_QUADS);
    // specify the ordinates of checkerboard facing the viewer squarely
    // set the current texture coordinates
    glTexCoord2f(0.0, 0.0);
    // specify a vertex, X Y Z
    glVertex3f(-2.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0);
    glVertex3f(-2.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0);
    glVertex3f(0.0, 1.0, 0.0);
    glTexCoord2f(1.0, 0.0);
    glVertex3f(0.0, -1.0, 0.0);

    // specify the ordinates of checkerboard tilting back at 45 degrees
    glTexCoord2f(0.0, 0.0);
    glVertex3f(1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0);
    glVertex3f(1.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0);
    glVertex3f(2.41421, 1.0, -1.41421);
    glTexCoord2f(1.0, 0.0);
    glVertex3f(2.41421, -1.0, -1.41421);
    glEnd();

    // force execution of GL commands in finite time
    glFlush();
}

//
void myReshape(GLsizei w, GLsizei h) {
    // set the viewport
    glViewport(0, 0, w, h);
    // specify which matrix is the current matrix
    // GL_PROJECTION: Applies subsequent matrix operations to the projection matrix stack.
    glMatrixMode(GL_PROJECTION);
    // replace the current matrix with the identity matrix
    glLoadIdentity();
    // set up a perspective projection matrix
    gluPerspective(60.0, 1.0 * (GLfloat)w / (GLfloat)h, 1.0, 30.0);

    // GL_MODELVIEW: Applies subsequent matrix operations to the modelview matrix stack.
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // multiply the current matrix by a translation matrix
    // The current matrix is multiplied by this translation matrix, with the product
    // replacing the current matrix
    glTranslatef(0.0, 0.0, -3.6);
}

//
void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(500, 500);
    if (!glutCreateWindow("Texture Mapping"))
        exit(0);
    myinit();
    // glutReshapeFunc sets the reshape callback for the current window
    glutReshapeFunc(myReshape);
    // sets the display callback for the current window
    glutDisplayFunc(display);
    // enters the GLUT event processing loop
    // This routine should be called at most once in a GLUT program.
    // Once called, this routine will never return. It will call as
    // necessary any callbacks that have been registered.
    glutMainLoop();
}
```



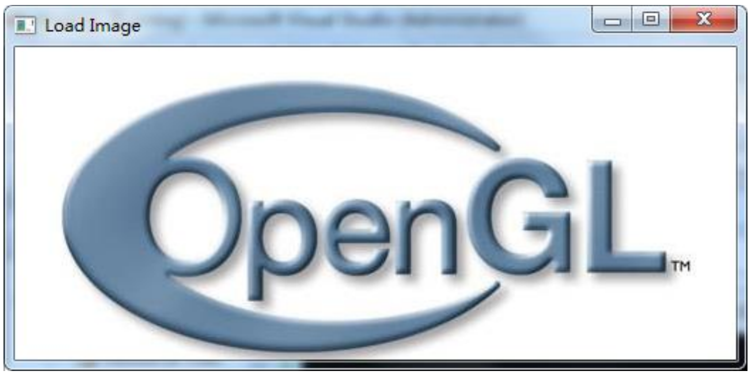
2) Sample code for mipmapping

```
/*Function: Texture with mipmapping
~~* SDK doc: https://www.opengl.org/sdk/docs~~
*/

#define FREEGLUT_STATIC
#include <GL/freeglut.h>
#include <stdlib.h>
#include <stdio.h>

GLbyte mipmapImage32[32][32][3];
GLbyte mipmapImage16[16][16][3];
GLbyte mipmapImage8[8][8][3];
GLbyte mipmapImage4[4][4][3];
GLbyte mipmapImage2[2][2][3];
GLbyte mipmapImage1[1][1][3];

// create the texture image
void loadImages(void) {
    int i, j;

    // yellow
    for (i = 0; i < 32; i++) {
        for (j = 0; j < 32; j++) {
            mipmapImage32[i][j][0] = 255.0;
            mipmapImage32[i][j][1] = 255.0;
            mipmapImage32[i][j][2] = 0.0;
        }
    }

    // pink
    for (i = 0; i < 16; i++) {
        for (j = 0; j < 16; j++) {
            mipmapImage16[i][j][0] = 255;
            mipmapImage16[i][j][1] = 0;
            mipmapImage16[i][j][2] = 255;
        }
    }

    // red
    for (i = 0; i < 8; i++) {
        for (j = 0; j < 8; j++) {
            mipmapImage8[i][j][0] = 255;
            mipmapImage8[i][j][1] = 0;
            mipmapImage8[i][j][2] = 0;
        }
    }

    // green
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
            mipmapImage4[i][j][0] = 0;
            mipmapImage4[i][j][1] = 255;
            mipmapImage4[i][j][2] = 0;
        }
    }

    // blue
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++) {
            mipmapImage2[i][j][0] = 0;
            mipmapImage2[i][j][1] = 0;
            mipmapImage2[i][j][2] = 255;
        }
    }

    // white
    mipmapImage1[0][0][0] = 255;
    mipmapImage1[0][0][1] = 255;
    mipmapImage1[0][0][2] = 255;
}

// all the texture mipmapping initialisation occurs here
void myinit(void) {
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glShadeModel(GL_FLAT);
    //set the texture and create the texture image
    loadImages();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, 32, 32, 0, GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage32[0][0][0]);
    glTexImage2D(GL_TEXTURE_2D, 1, 3, 16, 16, 0, GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage16[0][0][0]);
    glTexImage2D(GL_TEXTURE_2D, 2, 3, 8, 8, 0, GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage8[0][0][0]);
    glTexImage2D(GL_TEXTURE_2D, 3, 3, 4, 4, 0, GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage4[0][0][0]);
    glTexImage2D(GL_TEXTURE_2D, 4, 3, 2, 2, 0, GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage2[0][0][0]);
    glTexImage2D(GL_TEXTURE_2D, 5, 3, 1, 1, 0, GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage1[0][0][0]);

    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    glEnable(GL_TEXTURE_2D);
}

void display(void) {
    // Clear the colour and depth buffers
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // set the Texture and vertex coordinates
    glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
    glTexCoord2f(0.0, 8.0); glVertex3f(-2.0, 1.0, 0.0);
    glTexCoord2f(8.0, 8.0); glVertex3f(2000.0, 1.0, -6000.0);
    glTexCoord2f(8.0, 0.0); glVertex3f(2000.0, -1.0, -6000.0);
    glEnd();
    glFlush();
}

void myReshape(GLsizei w, GLsizei h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, 1.0 * (GLfloat)w / (GLfloat)h, 1.0, 30000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // glTranslate moves the coordinate system origin to the point specified by (x, y, z)
    glTranslatef(0.0, 0.0, -3.6);
}

void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Texture Mapping");
    myinit();
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutMainLoop();
}
```



3) Sample code for mapping an image by pixel

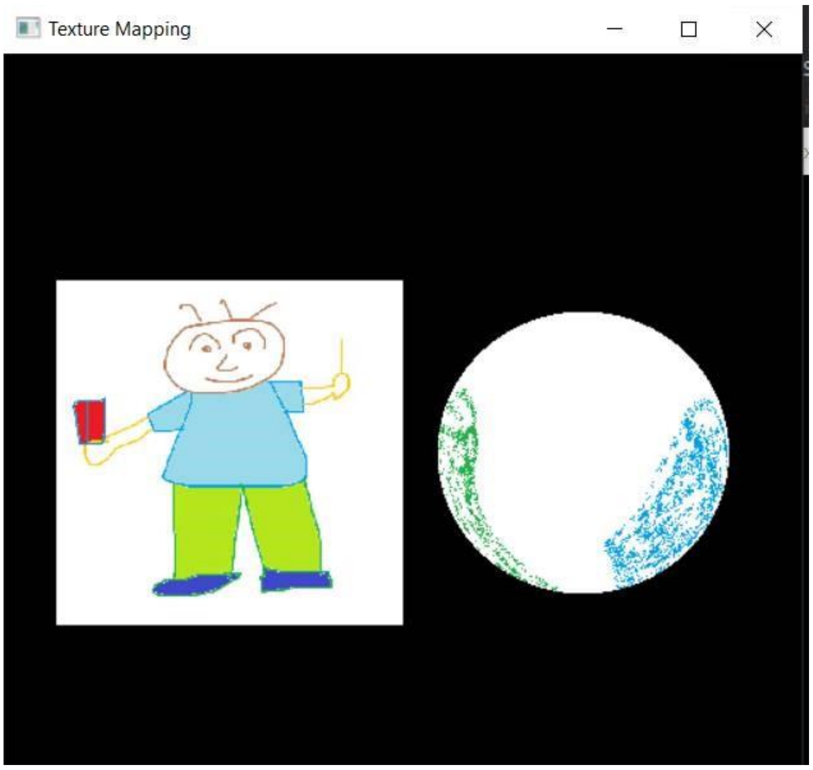
```
/*
* Function: mapping an image by pixel
~~* SDK doc: https://www.opengl.org/sdk/docs~~
* Reference: http://blog.sina.com.cn/s/blog_7bfd28570100v31g.html
*/

#define FREEGLUT_STATIC
#include <GL/freeglut.h>
#include <stdlib.h>
#include <stdio.h>

static GLint imagewidth;
static GLint imageheight;
static GLint pixellength;
static GLubyte* pixeldata;

void myinit(void) {
    // Read in and open an image file
    FILE* pfile = NULL;  // The image file should be placed in the same folder with the source code.
    fopen_s(&pfile, "Lab10Logo.bmp", "rb");  // read in binary mode and keep the image file in the folder
    if (pfile == 0) exit(0);                 // where the source code is saved

    // get the width and height of image
    // reposition stream position indicator
    // SEEK_SET: Beginning of file
    fseek(pfile, 0x0012, SEEK_SET);  // skip 18 bits from beginning
    // get the width of image
    fread(&imagewidth, sizeof(imagewidth), 1, pfile);
    // get the height of image
    fread(&imageheight, sizeof(imageheight), 1, pfile);

    // count the length of the image by pixel
    // pixel data consists of three colors red, green and blue (Windows implement BGR)
    pixellength = imagewidth * 3;
    // pixel data width should be an integral multiple of 4, which is required by the .bmp file
    while (pixellength % 4 != 0)
        pixellength++;  // 4 bits per colour?? Not needed!!
    // pixel data length = width * height
    pixellength *= imageheight;

    // malloc for the image by pixel
    pixeldata = (GLubyte*)malloc(pixellength);  // allocate memory
    if (pixeldata == 0) exit(0);

    // read the data of image as pixel
    fseek(pfile, 54, SEEK_SET);  // not 0x00!!
    fread(pixeldata, pixellength, 1, pfile);

    // close file
    fclose(pfile);
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  // Clear the colour and depth buffers
    // write a block of pixels to the frame buffer
    glDrawPixels(imagewidth, imageheight, GL_BGR_EXT, GL_UNSIGNED_BYTE, pixeldata);
    glFlush();
}

void myReshape(GLsizei w, GLsizei h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, 1.0 * (GLfloat)w / (GLfloat)h, 1.0, 30000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
    myinit();
    glutInitWindowSize(imagewidth, imageheight);
    glutCreateWindow("Loading Image by Pixel");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutMainLoop();
}
```



4) Mapping image onto objects

The following code provides an example of mapping an image file (bmp) to a flat object and to a sphere. The sphere can be rotated to show the mapping effect. Some implementation of OpenGL only allows image files of size in the form of 2^n (n is an integer). If your bmp file is not in this format, you will need to convert it into this format, e.g. by using the Paint program available in Windows.

```
/*
* Function: Additional Sample Code for Texture Mapping an image file (bmp) to a flat object and to a
sphere
*/

#define FREEGLUT_STATIC
#include <GL/freeglut.h>
#include <stdio.h>
#include <stdlib.h>
#include "vector"

using namespace std;

GLint imagewidth0, imagewidth1;
GLint imageheight0, imageheight1;
GLint pixellength0, pixellength1;
vector<GLubyte*>p;  // Similar to GLubyte* for program 3 but for 2 images (so a vector)
GLuint texture[2];
GLfloat angle = 0;


void ReadImage(const char path[256], GLint& imagewidth, GLint& imageheight, GLint& pixellength) {
    GLubyte* pixeldata;
    FILE* pfile;
    fopen_s(&pfile, path, "rb");
    if (pfile == 0) exit(0);

    fseek(pfile, 0x0012, SEEK_SET);
    fread(&imagewidth, sizeof(imagewidth), 1, pfile);
    fread(&imageheight, sizeof(imageheight), 1, pfile);

    pixellength = imagewidth * 3;
    while (pixellength % 4 != 0)pixellength++;
    pixellength *= imageheight;

    pixeldata = (GLubyte*)malloc(pixellength);
    if (pixeldata == 0) exit(0);

    fseek(pfile, 54, SEEK_SET);
    fread(pixeldata, pixellength, 1, pfile);
    p.push_back(pixeldata);  // Similar to glDrawPixels for program 3
    fclose(pfile);
}

void myinit(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glShadeModel(GL_FLAT);
    glEnable(GL_TEXTURE_2D);
    ReadImage("Lab10Image1.bmp", imagewidth0, imageheight0, pixellength0);
    ReadImage("Lab10Image2.bmp", imagewidth1, imageheight1, pixellength1);

    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);  // set pixel storage modes (in the memory)
    glGenTextures(2, &texture[0]);  // number of texture names to be generated and an array of texture
names
    glBindTexture(GL_TEXTURE_2D, texture[0]);  // target to which texture is bound and name of a texture
    glTexImage2D(GL_TEXTURE_2D, 0, 3, imagewidth0, imageheight0, 0, GL_BGR_EXT, GL_UNSIGNED_BYTE, p[0]);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glBindTexture(GL_TEXTURE_2D, texture[1]);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, imagewidth1, imageheight1, 0, GL_BGR_EXT, GL_UNSIGNED_BYTE, p[1]);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
}

void keyboard_input(unsigned char key, int x, int y) {
    if (key == 'r' || key == 'R') {
        if (angle < 360) {
            angle = angle + 30;
        }
        else {
            angle = 0;
        }
    }
    glutPostRedisplay();
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glTranslatef(0.0, 0.0, -4.0);
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
    glEnd();

    glTranslatef(1.0, 0.0, 0.0);
    glRotatef(angle, 0.0, 1.0, 0.0);
    glBindTexture(GL_TEXTURE_2D, texture[1]);
    GLUquadric* quadricObj = gluNewQuadric();  // Create a quadric surface object
    gluQuadricTexture(quadricObj, GL_TRUE);    // Set texture mode to true
    gluSphere(quadricObj, 0.8, 80, 80);
    gluDeleteQuadric(quadricObj);  // object must be deleted or it will be created every call of the
    glLoadIdentity();              // function and memory taken!!!
    glFlush();
}

void myReshape(GLsizei w, GLsizei h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, 1.0 * (GLfloat)w / (GLfloat)h, 0.0, 300.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(500, 500);
    if (!glutCreateWindow("Texture Mapping"))
        exit(0);
    myinit();
    glutKeyboardFunc(keyboard_input);
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutMainLoop();
}
```


Texture Mapping