Department of Computer Science and Engineering

Shivanirudh S G, 185001146, Semester VII

26 July 2021

UCS1712 - Graphics and Multimedia Lab

Exercise 1: Study of Basic Output Primitives in C++ using OpenGL

Objective:

To create an output window using OPENGL and to draw the following basic output primitives: POINTS, LINES, LINE_STRIP, LINE_LOOP, TRIANGLES, QUADS, QUAD_STRIP, POLYGON.

Code:

Common files

```
#ifndef LOPENGL_H
#define LOPENGL_H

#include <GL/freeglut.h>
```

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <stdio.h>

#endif
```

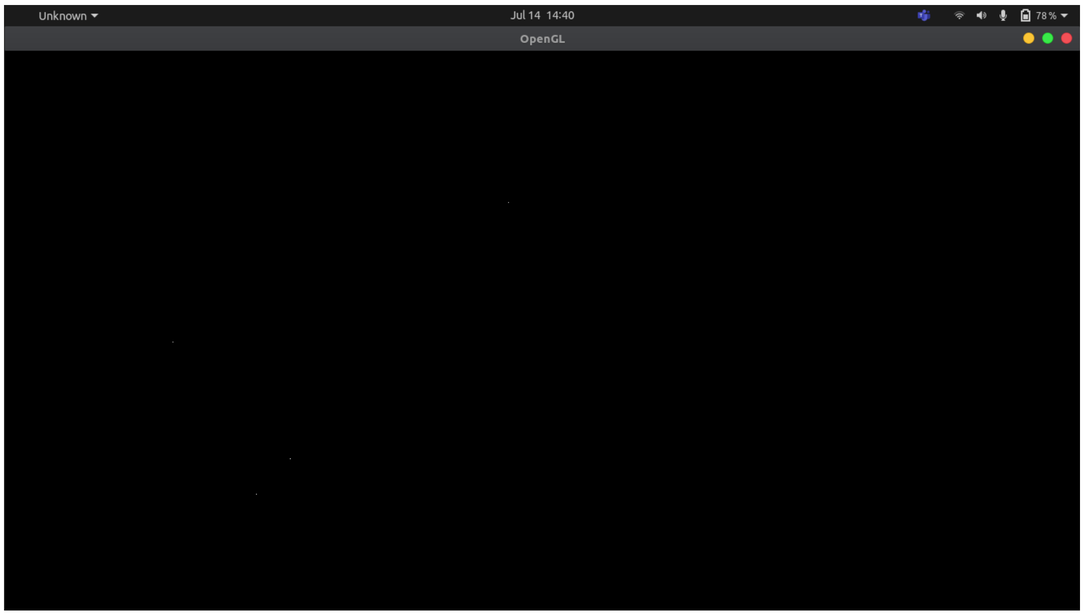
```
#ifndef LUTIL_H
#define LUTIL_H

#include "LOpenGL.h"
#include <stdio.h>

//Screen Constants
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_FPS = 60;

bool initGL();
/*
Pre Condition:
 -A valid OpenGL context
Post Condition:
 -Initializes matrices and clear color
 -Reports to console if there was an OpenGL error
 -Returns false if there was an error in initialization
Side Effects:
 -Projection matrix is set to identity matrix
 -Modelview matrix is set to identity matrix
 -Matrix mode is set to modelview
 -Clear color is set to black
*/

void update();
/*
Pre Condition:
 -None
Post Condition:
 -Does per frame logic
Side Effects:
 -None
*/

void render();
/*
Pre Condition:
 -A valid OpenGL context
 -Active modelview matrix
Post Condition:
 -Renders the scene
Side Effects:
 -Clears the color buffer
 -Swaps the front/back buffer
*/

#endif
```

```
#include "LUtil.h"

void runMainLoop( int val );
/*
Pre Condition:
 -Initialized freeGLUT
Post Condition:
 -Calls the main loop functions and sets itself to be called
   back in 1000 / SCREEN_FPS milliseconds
Side Effects:
 -Sets glutTimerFunc
*/

int main( int argc, char* args[] )
{
    //Initialize FreeGLUT
    glutInit( &argc, args );

    //Create OpenGL 2.1 context
    glutInitContextVersion( 2, 1 );

    //Create Singlele Buffered Window
    glutInitDisplayMode( GLUT_SINGLE|GLUT_RGB );
    glutInitWindowSize( SCREEN_WIDTH, SCREEN_HEIGHT );
    glutCreateWindow( "OpenGL" );

    //Do post window/context creation initialization
    if( !initGL() )
    {
        printf( "Unable to initialize graphics library!\n" );
        return 1;
    }

    //Set rendering function
    glutDisplayFunc( render );

    //Set main loop
    glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, 0 );

    //Start GLUT main loop
    glutMainLoop();

    return 0;
}

void runMainLoop( int val )
{
    //Frame logic
    update();
    render();

    //Run frame one more time
    glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, val );
}
```

POINTS:

```
void render()
{
    //Clear color buffer
     glClear(GL_COLOR_BUFFER_BIT);
     glBegin(GL_POINTS);
        glVertex2d(70,130);
        glVertex2d(100,230);
        glVertex2d(170,130);
        glVertex2d(300,350);
     glEnd();
     glFlush();

    //Update screen
    //glutSwapBuffers();
}
```

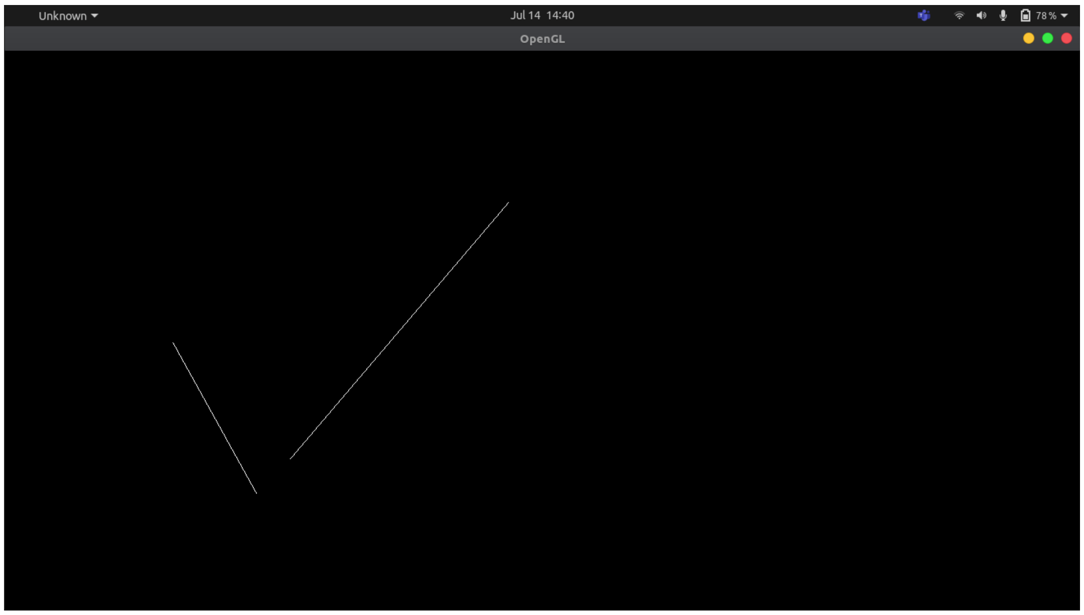
Output:



LINES:

```
void render()
{
    //Clear color buffer
     glClear(GL_COLOR_BUFFER_BIT);
     glBegin(GL_LINES);
        glVertex2d(70,130);
        glVertex2d(100,230);
        glVertex2d(170,130);
        glVertex2d(300,350);
     glEnd();
     glFlush();

    //Update screen
    //glutSwapBuffers();
}
```

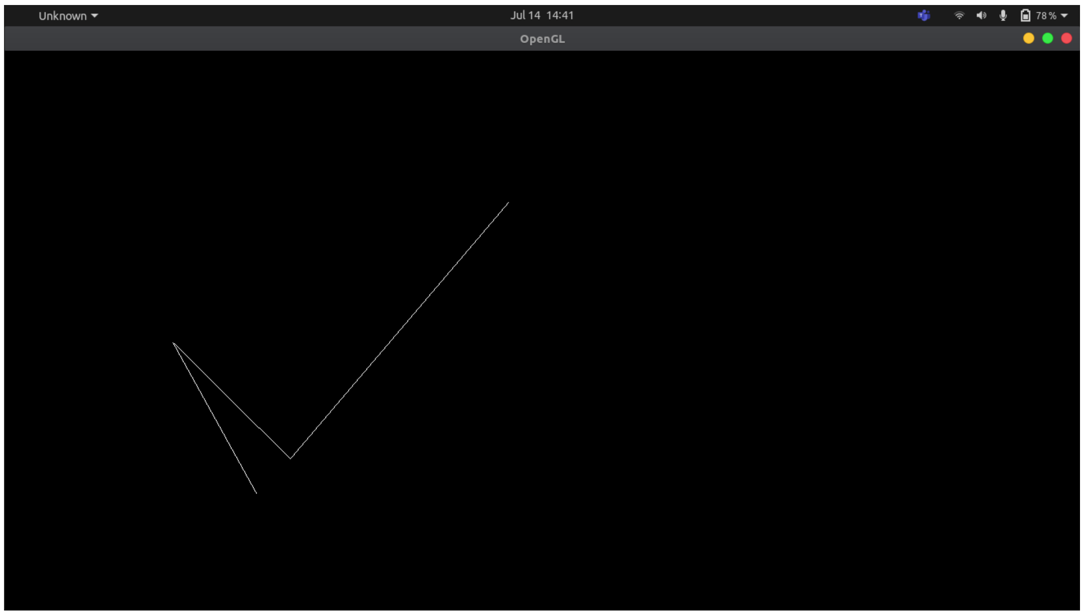
Output:



LINE_STRIP:

```
void render()
{
    //Clear color buffer
     glClear(GL_COLOR_BUFFER_BIT);
     glBegin(GL_LINE_STRIP);
        glVertex2d(70,130);
        glVertex2d(100,230);
        glVertex2d(170,130);
        glVertex2d(300,350);
     glEnd();
     glFlush();

    //Update screen
    //glutSwapBuffers();
}
```

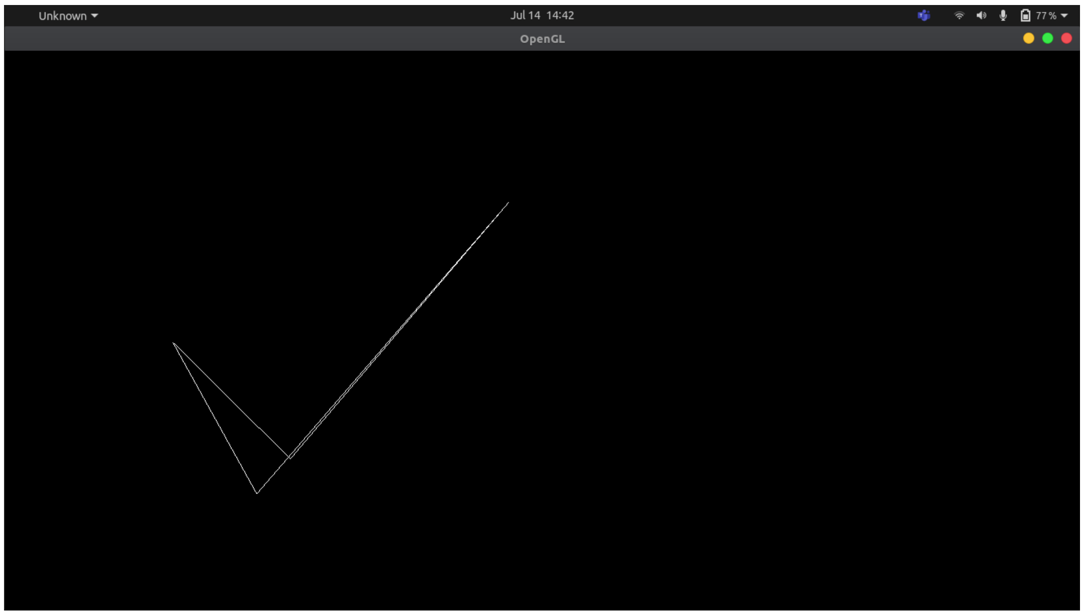
Output:



LINE_LOOP:

```
void render()
{
    //Clear color buffer
     glClear(GL_COLOR_BUFFER_BIT);
     glBegin(GL_LINE_LOOP);
        glVertex2d(70,130);
        glVertex2d(100,230);
        glVertex2d(170,130);
        glVertex2d(300,350);
     glEnd();
     glFlush();

    //Update screen
    //glutSwapBuffers();
}
```

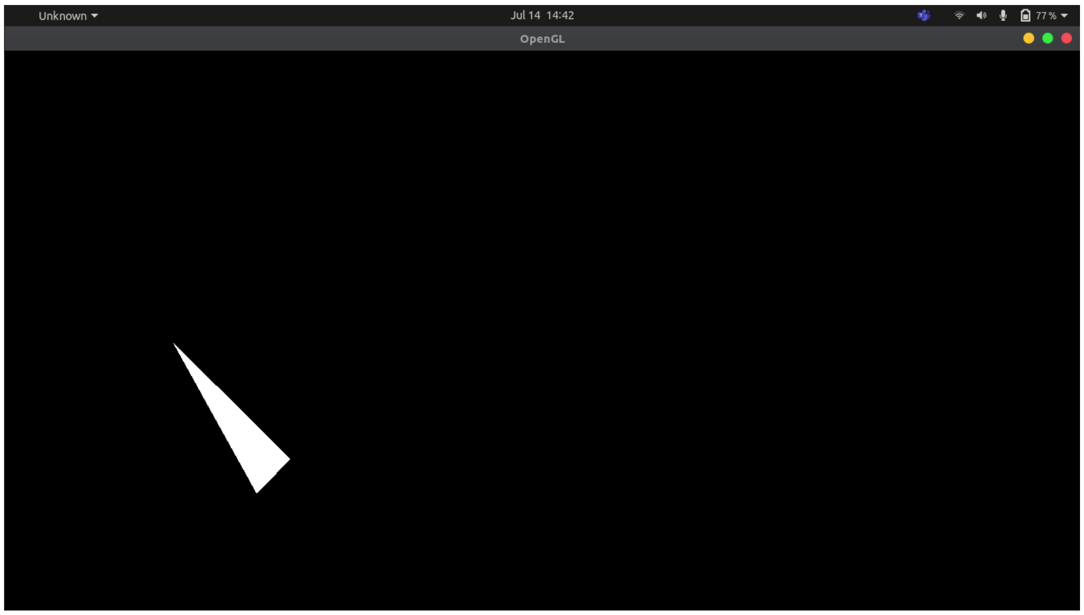
Output:



TRIANGLES:

```
void render()
{
    //Clear color buffer
     glClear(GL_COLOR_BUFFER_BIT);
     glBegin(GL_TRIANGLES);
        glVertex2d(70,130);
        glVertex2d(100,230);
        glVertex2d(170,130);
        glVertex2d(300,350);
     glEnd();
     glFlush();

    //Update screen
    //glutSwapBuffers();
}
```

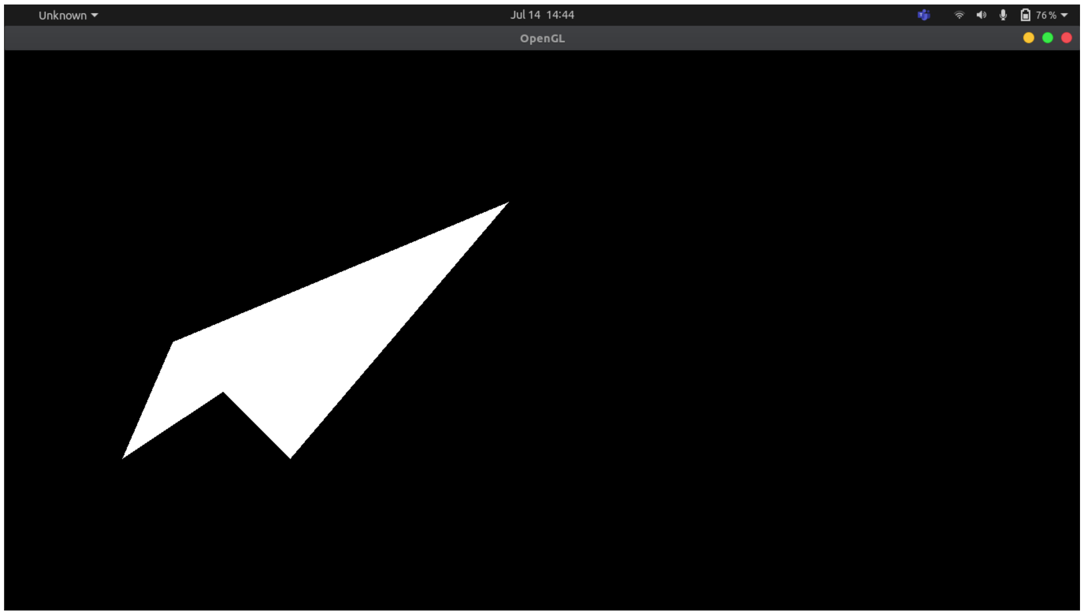
Output:



QUADS:

```
void render()
{
    //Clear color buffer
     glClear(GL_COLOR_BUFFER_BIT);
     glBegin(GL_QUADS);
        glVertex2d(70,130);
        glVertex2d(100,230);
        glVertex2d(170,130);
        glVertex2d(300,350);
     glEnd();
     glFlush();

    //Update screen
    //glutSwapBuffers();
}
```

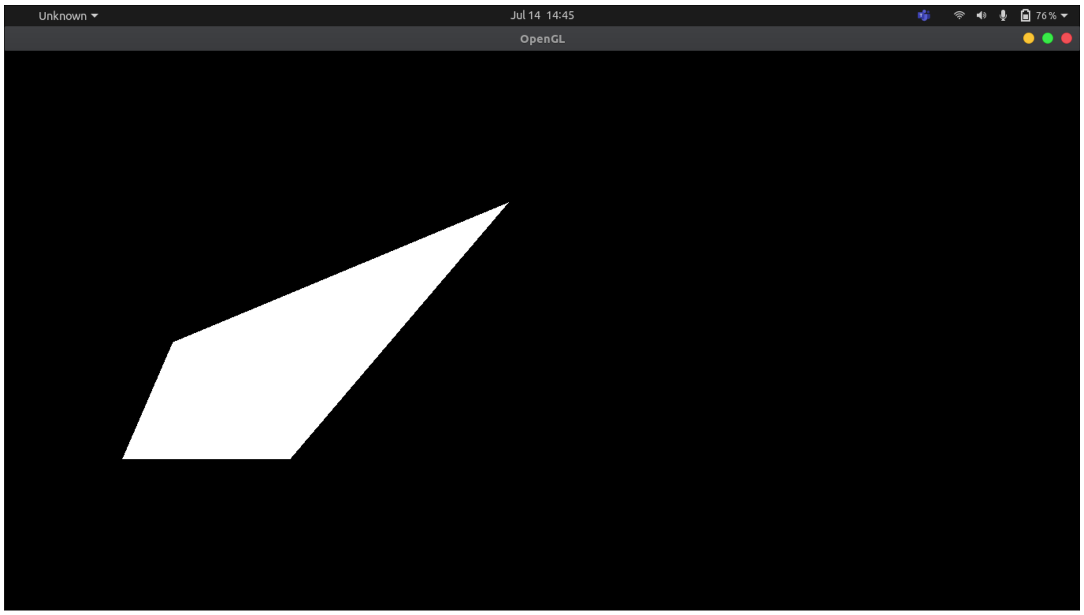
Output:



QUAD_STRIP:

```
void render()
{
    //Clear color buffer
     glClear(GL_COLOR_BUFFER_BIT);
     glBegin(GL_LINE_STRIP);
        glVertex2d(70,130);
        glVertex2d(100,230);
        glVertex2d(170,130);
        glVertex2d(300,350);
     glEnd();
     glFlush();

    //Update screen
    //glutSwapBuffers();
}
```

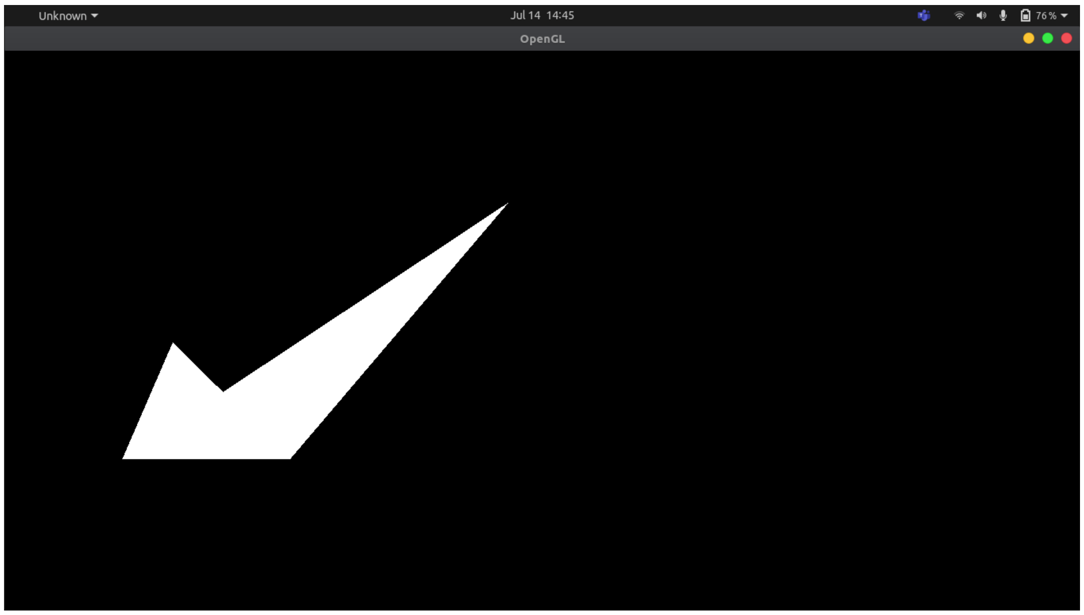
Output:



POLYGON:

```
void render()
{
    //Clear color buffer
     glClear(GL_COLOR_BUFFER_BIT);
     glBegin(GL_LINE_LOOP);
        glVertex2d(70,130);
        glVertex2d(100,230);
        glVertex2d(170,130);
        glVertex2d(300,350);
     glEnd();
     glFlush();

    //Update screen
    //glutSwapBuffers();
}
```

Output:



Objective:

To create an output window and draw a checkerboard using OpenGL.

Code:

```

#ifndef LOPENGL_H
#define LOPENGL_H

#include <GL/freeglut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <stdio.h>

#endif
```

```
#ifndef LTEXTURE_H
#define LTEXTURE_H

#include "LOpenGL.h"
#include<stdio.h>

class LTexture
{
    public:
        //Constructor
        LTexture();

        //Destructor
        ~LTexture();

        //Creates texture from pixels
        bool loadTextureFromPixels32( GLuint* pixels, GLuint
    width, GLuint height );

        //Delete Texture
        void freeTexture();

        void render( GLfloat x, GLfloat y );

        GLuint getTextureID();

        GLuint textureWidth();

        GLuint textureHeight();

    private:
        //Texture name
        GLuint mTextureID;

        //Texture dimensions
        GLuint mTextureWidth;
        GLuint mTextureHeight;
};

#endif
```

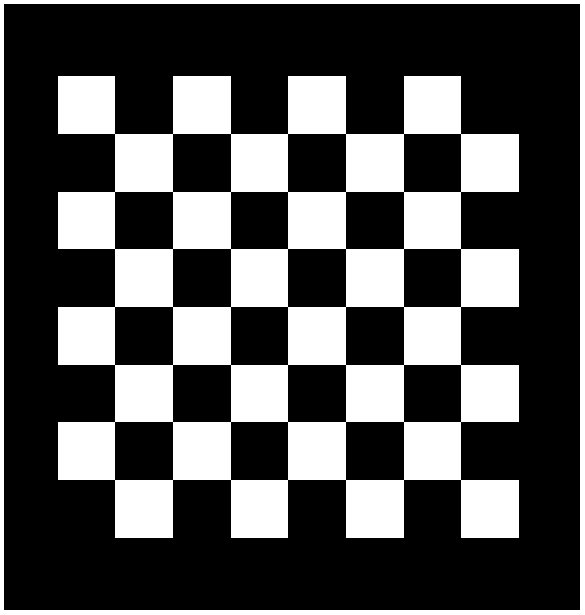
```

#include "LTexture.h"

LTexture::LTexture(){
    //Initialize texture ID
    mTextureID = 0;

    //Initialize texture dimensions
    mTextureWidth = 0;
    mTextureHeight = 0;
}

LTexture::~LTexture(){
    //Free texture data if needed
    freeTexture();
}

bool LTexture::loadTextureFromPixels32( GLuint* pixels,
   GLuint width, GLuint height ){
    //Free texture if it exists
    freeTexture();

    //Get texture dimensions
    mTextureWidth = width;
    mTextureHeight = height;

    //Generate texture ID
    glGenTextures( 1, &mTextureID );

    //Bind texture ID
    glBindTexture( GL_TEXTURE_2D, mTextureID );

    //Generate texture
    glTexImage2D( GL_TEXTURE_2D, 0, GL_RGBA, width, height,
   0, GL_RGBA, GL_UNSIGNED_BYTE, pixels );

    //Set texture parameters
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
   GL_LINEAR );
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
   GL_LINEAR );

    //Unbind texture
    glBindTexture( GL_TEXTURE_2D, NULL );

    //Check for error
    GLenum error = glGetError();
    if( error != GL_NO_ERROR ){
        printf( "Error loading texture from %p pixels! %s\n",
    pixels, gluErrorString( error ) );
        return false;
    }

    return true;
}

void LTexture::freeTexture(){
    //Delete texture
    if( mTextureID != 0 ){
        glDeleteTextures( 1, &mTextureID );
        mTextureID = 0;
    }

    mTextureWidth = 0;
    mTextureHeight = 0;
}

void LTexture::render( GLfloat x, GLfloat y ){
    //If the texture exists
    if( mTextureID != 0 ){
        //Remove any previous transformations
        glLoadIdentity();

        //Move to rendering point
        glTranslatef( x, y, 0.f );

        //Set texture ID
        glBindTexture( GL_TEXTURE_2D, mTextureID );

        //Render textured quad
        glBegin( GL_QUADS );
            glTexCoord2f( 0.f, 0.f ); glVertex2f(
   0.f,           0.f );
            glTexCoord2f( 1.f, 0.f ); glVertex2f(
   mTextureWidth,           0.f );
            glTexCoord2f( 1.f, 1.f ); glVertex2f(
   mTextureWidth, mTextureHeight );
            glTexCoord2f( 0.f, 1.f ); glVertex2f(
   0.f, mTextureHeight );
        glEnd();
    }
}

GLuint LTexture::getTextureID(){
    return mTextureID;
}

GLuint LTexture::textureWidth(){
    return mTextureWidth;
}

GLuint LTexture::textureHeight(){
    return mTextureHeight;
}
```

```

#ifndef LUTIL_H
#define LUTIL_H

#include "LOpenGL.h"
#include <stdio.h>

//Screen Constants
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_FPS = 60;

bool initGL();

bool loadMedia();

void update();

void render();

#endif
```

```

#include "LUtil.h"
#include "LTexture.h"

//Checkerboard texture
LTexture gCheckerBoardTexture;

bool initGL(){
    //Initialize Projection Matrix
    glViewport(0.f, 0.f, SCREEN_WIDTH, SCREEN_HEIGHT);
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluOrtho2D(0.0,640.0,0.0,480.0);

    //Initialize Modelview Matrix
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();

    //Initialize clear color
    glClearColor( 0.f, 0.f, 0.f, 1.f );

    //Enable texturing
    glEnable( GL_TEXTURE_2D );

    //Check for error
    GLenum error = glGetError();
    if( error != GL_NO_ERROR )
    {
        printf( "Error initializing OpenGL! %s\n",
    gluErrorString( error ) );
        return false;
    }

    return true;
}

bool loadMedia(){
    //Checkerboard pixels
    const int CHECKERBOARD_WIDTH = 128;
    const int CHECKERBOARD_HEIGHT = 128;
    const int CHECKERBOARD_PIXEL_COUNT = CHECKERBOARD_WIDTH * CHECKERBOARD_HEIGHT;
    GLuint checkerBoard[ CHECKERBOARD_PIXEL_COUNT ];

    //Go through pixels
    for( int i = 0; i < CHECKERBOARD_PIXEL_COUNT; ++i )
    {
        //Get the individual color components
        GLubyte* colors = (GLubyte*)&checkerBoard[ i ];

        //If the 5th bit of the x and y offsets of the pixel do not match
        if( i / 128 & 16 ^ i % 128 & 16 )
        {
            //Set pixel to white
            colors[ 0 ] = 0xFF;
            colors[ 1 ] = 0xFF;
            colors[ 2 ] = 0xFF;
            colors[ 3 ] = 0xFF;
        }
        else
        {
            //Set pixel to red
            colors[ 0 ] = 0x00;
            colors[ 1 ] = 0x00;
            colors[ 2 ] = 0x00;
            colors[ 3 ] = 0xFF;
        }
    }

    //Load texture
    if( !gCheckerBoardTexture.loadTextureFromPixels32( checkerBoard, CHECKERBOARD_WIDTH, CHECKERBOARD_HEIGHT ) )
    {
        printf( "Unable to load checkerboard texture!\n" );
        return false;
    }

    return true;
}

void update(){

}

void render(){
    //Clear color buffer
     glClear(GL_COLOR_BUFFER_BIT);

    //Calculate centered offsets
    GLfloat x = ( SCREEN_WIDTH - gCheckerBoardTexture.
   textureWidth() ) / 2.f;
    GLfloat y = ( SCREEN_HEIGHT - gCheckerBoardTexture.
   textureHeight() ) / 2.f;

    //Render checkerboard texture
    gCheckerBoardTexture.render( x, y );

    //Update screen
    glutSwapBuffers();
}
```

```

#include "LUtil.h"

void runMainLoop( int val );


int main( int argc, char* args[] )
{
    //Initialize FreeGLUT
    glutInit( &argc, args );

    //Create OpenGL 2.1 context
    glutInitContextVersion( 2, 1 );

    //Create Singlele Buffered Window
    glutInitDisplayMode( GLUT_DOUBLE );
    glutInitWindowSize( SCREEN_WIDTH, SCREEN_HEIGHT );
    glutCreateWindow( "OpenGL" );

    //Do post window/context creation initialization
    if( !initGL() ){
        printf( "Unable to initialize graphics library!\n" );
        return 1;
    }

    if(!loadMedia()){
        printf("Unable to load media\n");
        return 2;
    }

    //Set rendering function
    glutDisplayFunc( render );

    //Set main loop
    glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, 0 );

    //Start GLUT main loop
    glutMainLoop();

    return 0;
}

void runMainLoop( int val )
{
    //Frame logic
    update();
    render();

    //Run frame one more time
    glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, val );
}
```

Output:



Objective:

To create an output window and draw a house using POINTS,LINES,TRAINGLES and QUADS/POLYGON.

Code:

```

#ifndef LOPENGL_H
#define LOPENGL_H

#include <GL/freeglut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <stdio.h>

#endif
```

```

#ifndef LUTIL_H
#define LUTIL_H

#include "LOpenGL.h"
#include <stdio.h>

//Screen Constants
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_FPS = 60;

bool initGL();

void update();

void render();

void building();

void roof();

void door();

void window();

void chimney();

void line(int x1, int y1, int x2, int y2);

void lineloop(int x1, int y1, int x2, int y2);

void triangle(int x1, int y1, int x2, int y2, int x3, int y3)
    ;

void quad(int x1, int y1, int x2, int y2);


#endif
```

```

#include "LUtil.h"

bool initGL()
{
    //Initialize Projection Matrix
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluOrtho2D(0.0,640.0,0.0,480.0);

    //Initialize Modelview Matrix
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();

    //Initialize clear color
    glClearColor( 0.f, 0.f, 0.f, 1.f );

    //Check for error
    GLenum error = glGetError();
    if( error != GL_NO_ERROR )
    {
        printf( "Error initializing OpenGL! %s\n",
    gluErrorString( error ) );
        return false;
    }

    return true;
}
```

```

void update()
{

}

void render()
{
    //Clear color buffer
    glClear(GL_COLOR_BUFFER_BIT);

    building();
    roof();
    door();
    window();
    chimney();

    glFlush();

    //Update screen
    //glutSwapBuffers();
}


void line(int x1, int y1, int x2, int y2) {

    glBegin(GL_LINES);

    glVertex2d(x1,y1);
    glVertex2d(x2,y2);

    glEnd();
}

void lineloop(int x1, int y1, int x2, int y2) {

    glBegin(GL_LINE_LOOP);

    glVertex2d(x1,y1);
    glVertex2d(x2,y1);
    glVertex2d(x2,y2);
    glVertex2d(x1,y2);

    glEnd();
}
```

```

void triangle(int x1, int y1, int x2, int y2, int x3, int y3)
    {

    glBegin(GL_TRIANGLES);

    glVertex2d(x1,y1);
    glVertex2d(x2,y2);
    glVertex2d(x3,y3);

    glEnd();
}

void quad(int x1, int y1, int x2, int y2) {

    glBegin(GL_QUADS);

    glVertex2d(x1,y1);
    glVertex2d(x2,y1);
    glVertex2d(x2,y2);
    glVertex2d(x1,y2);

    glEnd();
}

void building() {
    lineloop(250,100, 330,250);

    lineloop(330,100, 530,250);
}

void roof() {
    triangle(250,250, 330,250, 290,300);
    line(290,300, 490,300);
    line(490,300, 530,250);

}

void door(){
    quad(280,100, 283,160);
    quad(280,158, 300,160);
    quad(297,100, 300,160);
}

void window(){
    quad(418,155, 420,167);
    quad(418,165, 442,167);
    quad(418,153, 442,155);
    quad(440,155, 442,167);
    line(430,155, 430,165);
    line(420,160, 440,160);
}

void chimney(){
    quad(500,260, 515,310);
}
```

```

#include "LUtil.h"

void runMainLoop( int val );


int main( int argc, char* args[] )
{
    //Initialize FreeGLUT
    glutInit( &argc, args );

    //Create OpenGL 2.1 context
    glutInitContextVersion( 2, 1 );

    //Create Singlele Buffered Window
    glutInitDisplayMode( GLUT_SINGLE|GLUT_RGB );
    glutInitWindowSize( SCREEN_WIDTH, SCREEN_HEIGHT );
    glutCreateWindow( "OpenGL" );

    //Do post window/context creation initialization
    if( !initGL() )
    {
        printf( "Unable to initialize graphics library!\n" );
        return 1;
    }

    //Set rendering function
    glutDisplayFunc( render );

    //Set main loop
    glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, 0 );

    //Start GLUT main loop
    glutMainLoop();

    return 0;
}

void runMainLoop( int val )
{
    //Frame logic
    update();
    render();

    //Run frame one more time
    glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, val );
}
```

Output: