Ashhar Hasan (13BCS-0015)

OpenGL Programs

Graphics Lab

Contents

[Main Driver Program 2](#_Toc467578023)

[Line 3](#_Toc467578024)

[Rectangle 5](#_Toc467578025)

[Triangle 7](#_Toc467578026)

[Circle 9](#_Toc467578027)

[Ellipse 12](#_Toc467578028)

[Cube 15](#_Toc467578029)

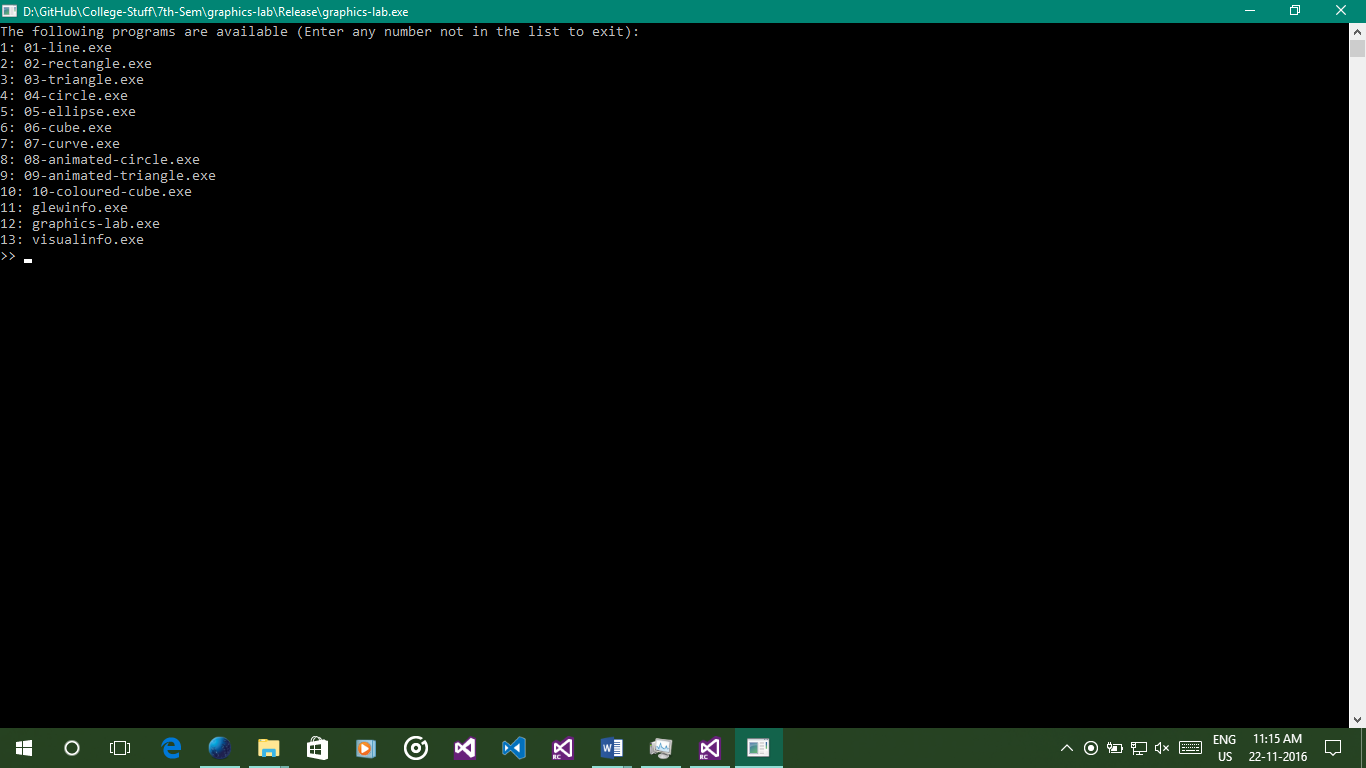
[Curve 17](#_Toc467578030)

[Animated Circle 20](#_Toc467578031)

[Animated Triangle 23](#_Toc467578032)

[Coloured Cube 25](#_Toc467578033)

# Main Driver Program



#include <iostream>

#include <string>

#include <vector>

#include <experimental/filesystem>

using namespace std;

bool recursiveFileGlob(string targetPath, vector<string> \*allMatchingFiles, string extension = ".exe")

{

using namespace experimental;

filesystem::directory\_iterator end\_itr;

for (filesystem::directory\_iterator program(targetPath); program != end\_itr; ++program)

{

// Skip if not a file

if (!filesystem::is\_regular\_file(program->status()))

recursiveFileGlob(program->path().string(), allMatchingFiles, extension);

if (program->path().extension() != extension)

continue;

// File matches, store it

allMatchingFiles->push\_back(program->path().string());

}

if (allMatchingFiles->size() > 0)

return true;

else

return false;

}

int main(int argc, char\*\* argv)

{

bool keepAlive = true;

const string basedir = "D:\\GitHub\\College-Stuff\\7th-Sem\\graphics-lab\\Release\\";

// Execute all available exe files from a menu.

while (keepAlive)

{

cout << "The following programs are available (Enter any number not in the list to exit): " << endl;

vector<string> programs;

if (recursiveFileGlob(basedir, &programs, ".exe"))

{

auto program = programs.begin();

for (int i = 0; program != programs.end(); ++program, ++i)

cout << i + 1 << ": " << program->substr(basedir.length()) << endl;

unsigned choice = 1;

cout << ">> ";

cin >> choice;

if (choice > programs.size() || choice < 1)

keepAlive = false;

else

system(string('"' + programs[choice - 1] + '"').c\_str());

}

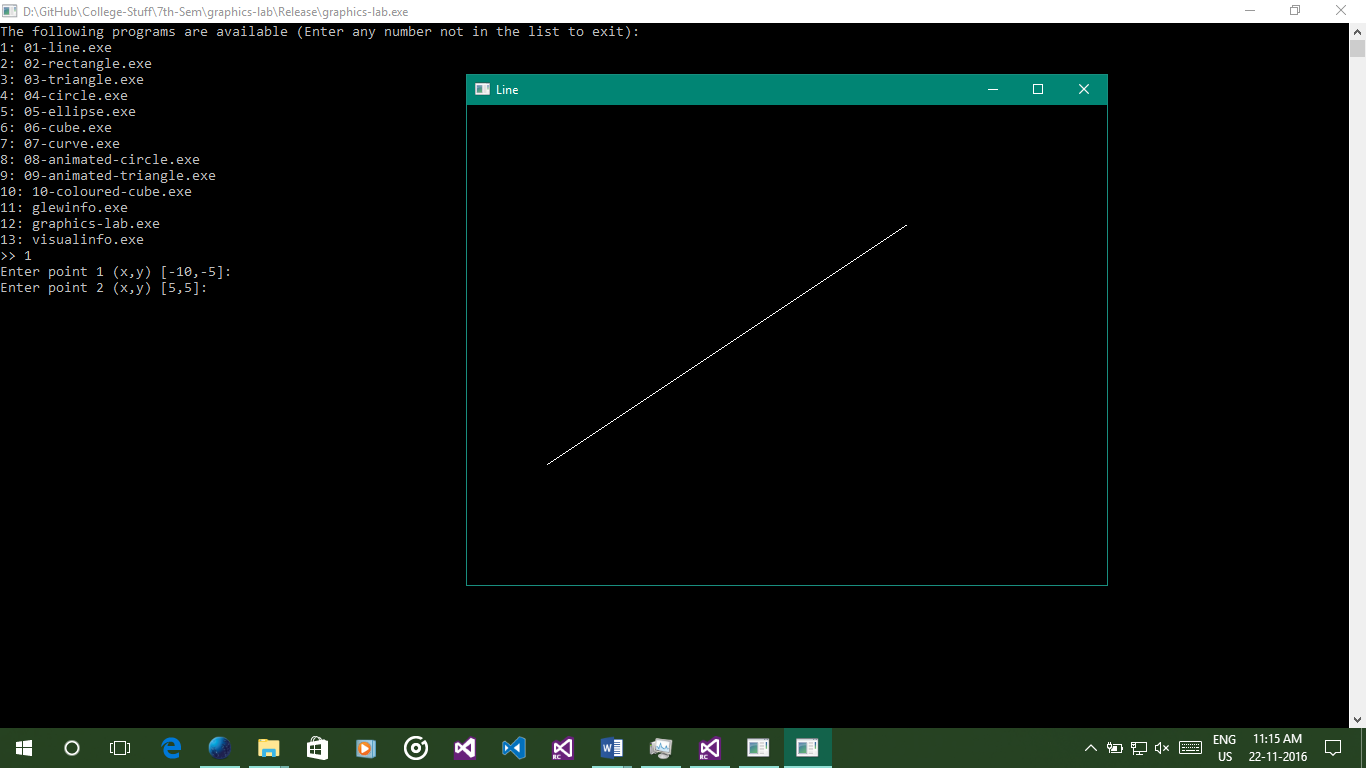
else

return -1;

}

}

# Line



#include "GL/freeglut.h"

#include <set>

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

auto point1 = pair<double, double>(-10, -5);

auto point2 = pair<double, double>(5, 5);

void renderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Set black background with white foreground

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor4f(1.0, 1.0, 1.0, 1.0);

// Start rendering the line primitive

glBegin(GL\_LINES);

glVertex2d(point1.first, point1.second);

glVertex2d(point2.first, point2.second);

glEnd();

glutSwapBuffers();

}

void reshapeScene(GLsizei width, GLsizei height)

{

// Find new aspect ratio

if (height == 0)

height = 1;

double aspect = (GLdouble)width / (GLdouble)height;

// Set the viewport to cover the new window

glViewport(0, 0, width, height);

// Set the aspect ratio of the clipping area to match the viewport

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (width >= height)

{

// aspect >= 1, set the height from -1 to 1, with larger width

gluOrtho2D(-1.0 \* aspect, 1.0 \* aspect, -1.0, 1.0);

}

else

{

// aspect < 1, set the width to -1 to 1, with larger height

gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);

}

}

int main(int argc, char\*\* argv)

{

cout << "Enter point 1 (x,y) [" << point1.first << "," << point1.second << "]: ";

string input;

getline(cin, input);

stringstream ss(input);

char split;

ss >> point1.first >> split >> point1.second;

cout << "Enter point 2 (x,y) [" << point2.first << "," << point2.second << "]: ";

getline(cin, input);

ss = stringstream(input);

ss >> point2.first >> split >> point2.second;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowPosition(50, 50);

glutInitWindowSize(640, 480);

glutCreateWindow("Line");

glEnable(GL\_DEPTH\_TEST);

point1.first /= 10; point1.second /= 10;

point2.first /= 10; point2.second /= 10;

glutDisplayFunc(renderScene);

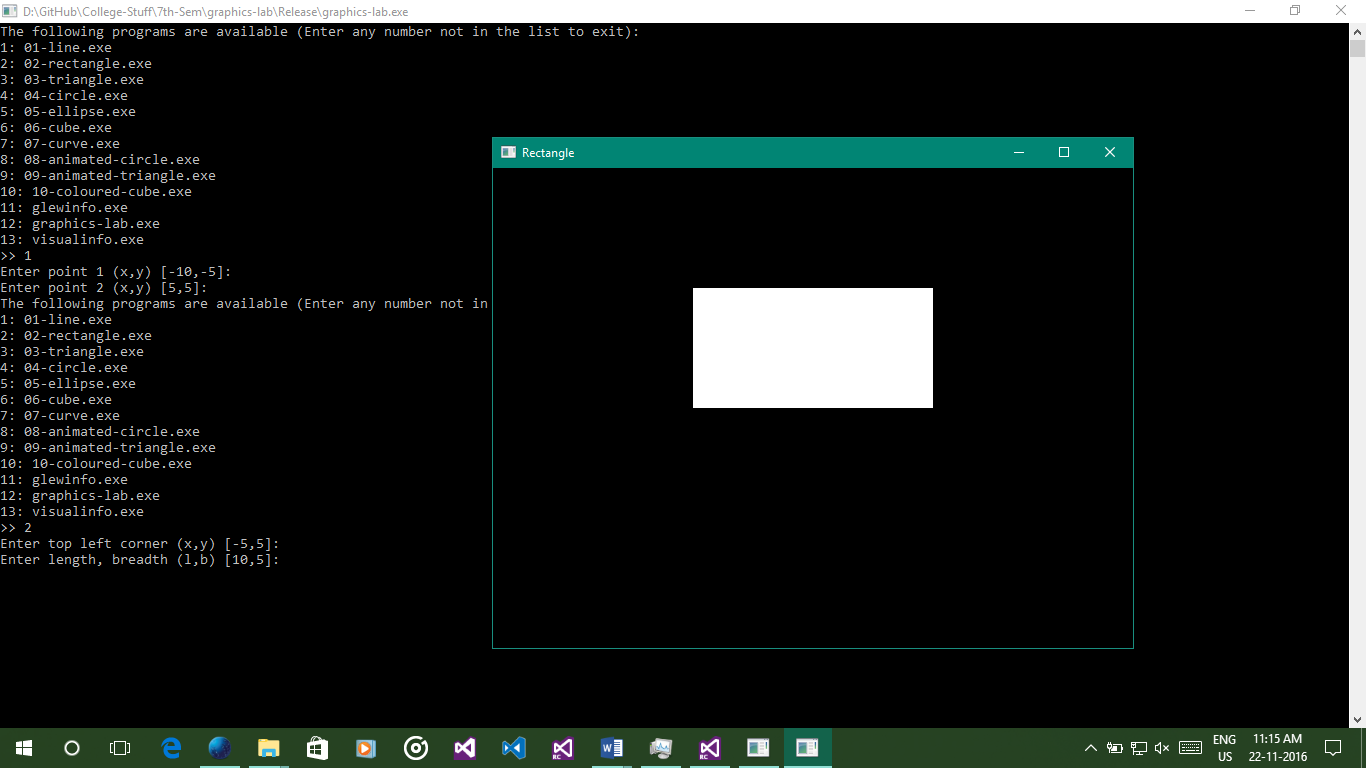
glutReshapeFunc(reshapeScene);

glutMainLoop();

return 0;

}

# Rectangle



#include "GL/freeglut.h"

#include <set>

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

auto topLeft = pair<double, double>(-5, 5);

double length = 10, breadth = 5;

void renderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Set black background with white foreground

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor4f(1.0, 1.0, 1.0, 1.0);

// Start rendering the quadrilateral primitive

glBegin(GL\_QUADS);

glVertex2d(topLeft.first, topLeft.second);

glVertex2d(topLeft.first + length, topLeft.second);

glVertex2d(topLeft.first + length, topLeft.second - breadth);

glVertex2d(topLeft.first, topLeft.second - breadth);

glEnd();

glutSwapBuffers();

}

void reshapeScene(GLsizei width, GLsizei height)

{

// Find new aspect ratio

if (height == 0)

height = 1;

GLdouble aspect = (GLdouble)width / (GLdouble)height;

// Set the viewport to cover the new window

glViewport(0, 0, width, height);

// Set the aspect ratio of the clipping area to match the viewport

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (width >= height)

{

// aspect >= 1, set the height from -1 to 1, with larger width

gluOrtho2D(-1.0 \* aspect, 1.0 \* aspect, -1.0, 1.0);

}

else

{

// aspect < 1, set the width to -1 to 1, with larger height

gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);

}

}

int main(int argc, char\*\* argv)

{

cout << "Enter top left corner (x,y) [" << topLeft.first << "," << topLeft.second << "]: ";

string input;

getline(cin, input);

stringstream ss(input);

char split;

ss >> topLeft.first >> split >> topLeft.second;

cout << "Enter length, breadth (l,b) [" << length << "," << breadth << "]: ";

getline(cin, input);

ss = stringstream(input);

ss >> length >> split >> breadth;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowPosition(50, 50);

glutInitWindowSize(640, 480);

glutCreateWindow("Rectangle");

glEnable(GL\_DEPTH\_TEST);

topLeft.first /= 10; topLeft.second /= 10;

length /= 10; breadth /= 10;

glutDisplayFunc(renderScene);

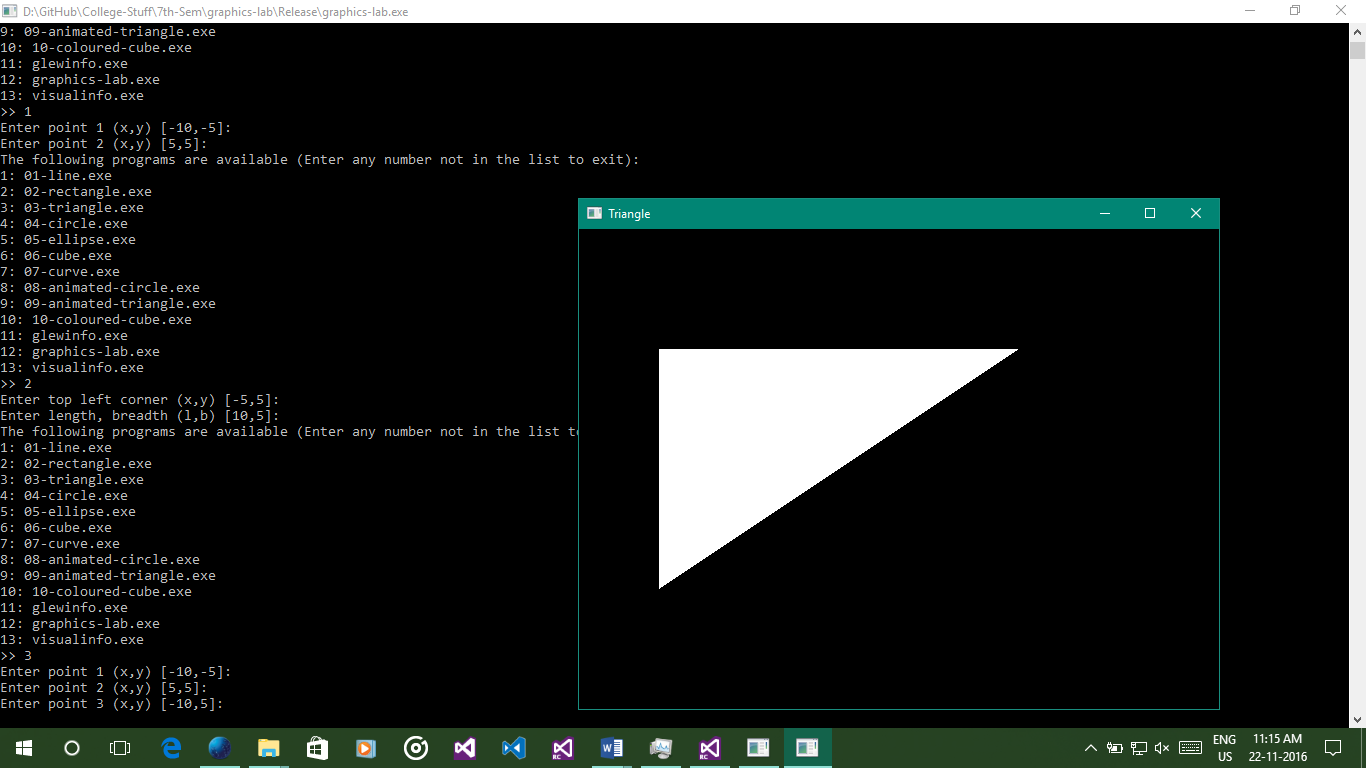
glutReshapeFunc(reshapeScene);

glutMainLoop();

return 0;

}

# Triangle



#include "GL/freeglut.h"

#include <set>

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

auto point1 = pair<double, double>(-10, -5);

auto point2 = pair<double, double>(5, 5);

auto point3 = pair<double, double>(-10, 5);

void renderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Set black background with white foreground

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor4f(1.0, 1.0, 1.0, 1.0);

// Start rendering the triangle primitive

glBegin(GL\_TRIANGLES);

glVertex2d(point1.first, point1.second);

glVertex2d(point2.first, point2.second);

glVertex2d(point3.first, point3.second);

glEnd();

glutSwapBuffers();

}

void reshapeScene(GLsizei width, GLsizei height)

{

// Find new aspect ratio

if (height == 0)

height = 1;

GLdouble aspect = (GLdouble)width / (GLdouble)height;

// Set the viewport to cover the new window

glViewport(0, 0, width, height);

// Set the aspect ratio of the clipping area to match the viewport

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (width >= height)

{

// aspect >= 1, set the height from -1 to 1, with larger width

gluOrtho2D(-1.0 \* aspect, 1.0 \* aspect, -1.0, 1.0);

}

else

{

// aspect < 1, set the width to -1 to 1, with larger height

gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);

}

}

int main(int argc, char\*\* argv)

{

cout << "Enter point 1 (x,y) [" << point1.first << "," << point1.second << "]: ";

string input;

getline(cin, input);

stringstream ss(input);

char split;

ss >> point1.first >> split >> point1.second;

cout << "Enter point 2 (x,y) [" << point2.first << "," << point2.second << "]: ";

getline(cin, input);

ss = stringstream(input);

ss >> point2.first >> split >> point2.second;

cout << "Enter point 3 (x,y) [" << point3.first << "," << point3.second << "]: ";

getline(cin, input);

ss = stringstream(input);

ss >> point3.first >> split >> point3.second;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowPosition(50, 50);

glutInitWindowSize(640, 480);

glutCreateWindow("Triangle");

glEnable(GL\_DEPTH\_TEST);

point1.first /= 10; point1.second /= 10;

point2.first /= 10; point2.second /= 10;

point3.first /= 10; point3.second /= 10;

glutDisplayFunc(renderScene);

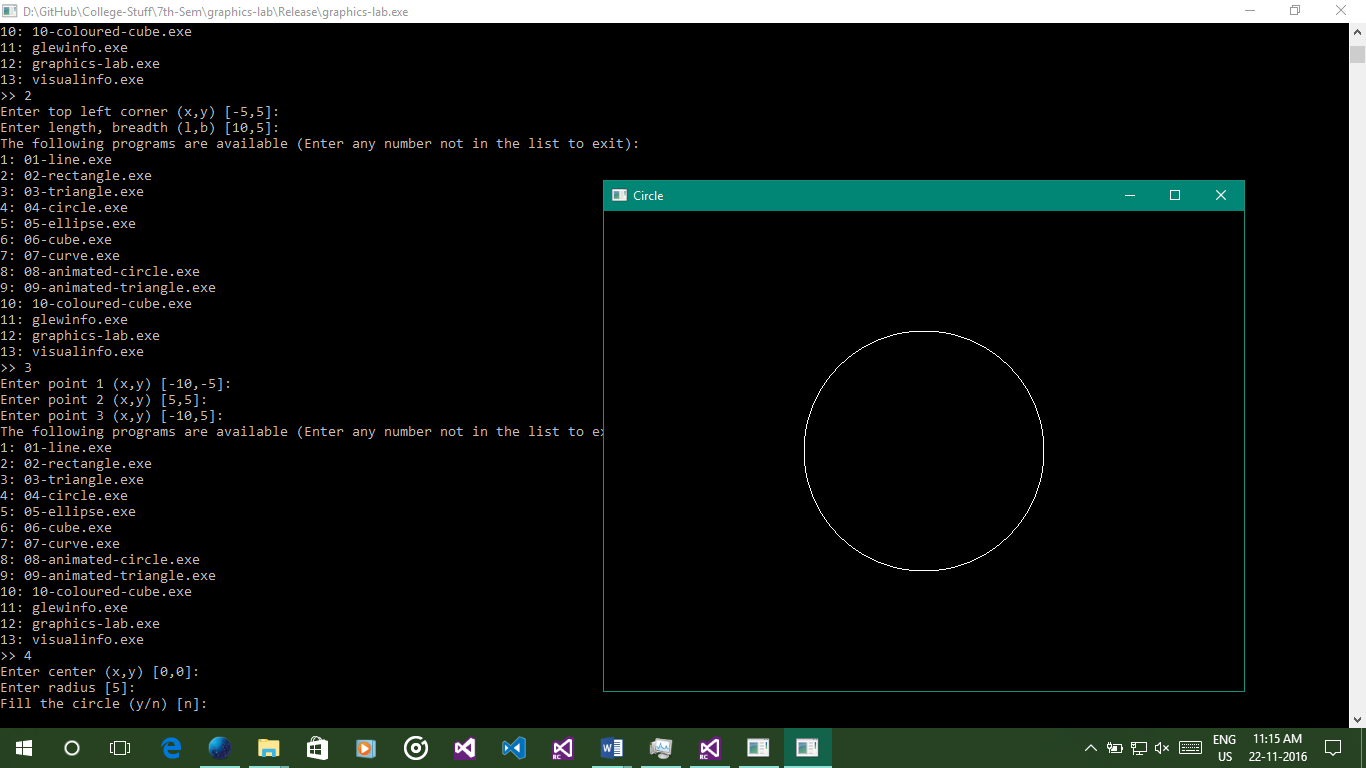
glutReshapeFunc(reshapeScene);

glutMainLoop();

return 0;

}

# Circle



#include "GL/freeglut.h"

#include <set>

#include <iostream>

#include <string>

#include <sstream>

#define \_USE\_MATH\_DEFINES

#include <math.h>

using namespace std;

auto center = pair<double, double>(0, 0);

double radius = 5.0;

bool filled = false;

void drawFilledCircle()

{

glBegin(GL\_TRIANGLE\_FAN);

glVertex2d(center.first, center.second);

for (double angle = 1.0; angle < 361.0; angle += 0.2)

{

double x = center.first + sin(angle) \* radius;

double y = center.second + cos(angle) \* radius;

glVertex2d(x, y);

}

glEnd();

}

void drawEmptyCircle()

{

// Start rendering the circle

glBegin(GL\_LINE\_LOOP);

const int loops = 300;

for (int i = 0; i <= loops; i++)

{

double angle = 2 \* M\_PI \* i / loops;

double x = radius \* cos(angle);

double y = radius \* sin(angle);

glVertex2d(x + center.first, y + center.second);

}

glEnd();

}

void renderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Set black background with white foreground

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor4f(1.0, 1.0, 1.0, 1.0);

filled ? drawFilledCircle() : drawEmptyCircle();

glutSwapBuffers();

}

void reshapeScene(GLsizei width, GLsizei height)

{

// Find new aspect ratio

if (height == 0)

height = 1;

GLdouble aspect = (GLdouble)width / (GLdouble)height;

// Set the viewport to cover the new window

glViewport(0, 0, width, height);

// Set the aspect ratio of the clipping area to match the viewport

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (width >= height)

{

// aspect >= 1, set the height from -1 to 1, with larger width

gluOrtho2D(-1.0 \* aspect, 1.0 \* aspect, -1.0, 1.0);

}

else

{

// aspect < 1, set the width to -1 to 1, with larger height

gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);

}

}

int main(int argc, char\*\* argv)

{

cout << "Enter center (x,y) [" << center.first << "," << center.second << "]: ";

string input;

getline(cin, input);

stringstream ss(input);

char split;

ss >> center.first >> split >> center.second;

cout << "Enter radius [" << radius << "]: ";

getline(cin, input);

ss = stringstream(input);

ss >> radius;

cout << "Fill the circle (y/n) [" << (filled ? "y" : "n") << "]: ";

getline(cin, input);

ss = stringstream(input);

char temp = filled ? 'y' : 'n';

ss >> temp;

if (tolower(temp) == 'y')

filled = true;

else

filled = false;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowPosition(50, 50);

glutInitWindowSize(640, 480);

glutCreateWindow("Circle");

glEnable(GL\_DEPTH\_TEST);

center.first /= 10; center.second /= 10;

radius /= 10;

glutDisplayFunc(renderScene);

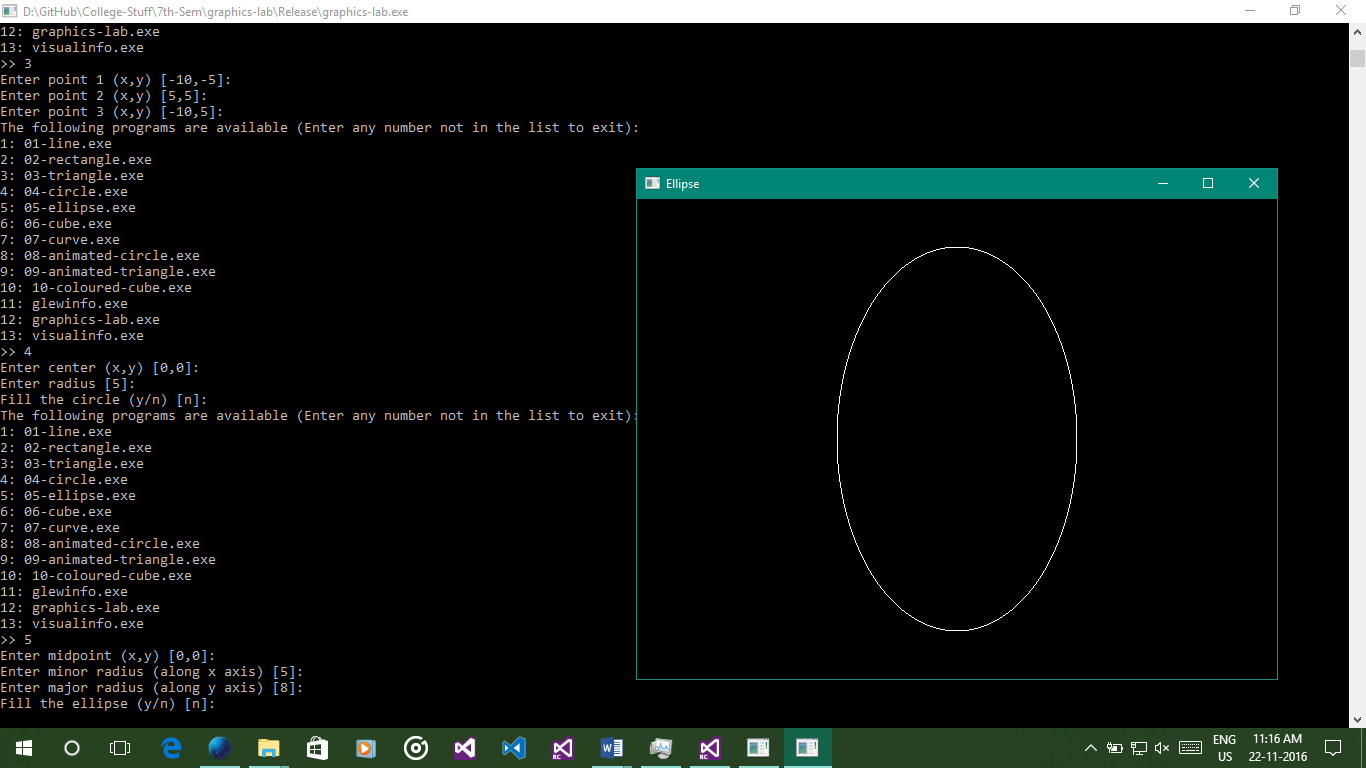
glutReshapeFunc(reshapeScene);

glutMainLoop();

return 0;

}

# Ellipse



#include "GL/freeglut.h"

#include <set>

#include <iostream>

#include <string>

#include <sstream>

#define \_USE\_MATH\_DEFINES

#include <math.h>

using namespace std;

auto center = pair<double, double>(0, 0);

double minorRadius = 5.0, majorRadius = 8.0;

bool filled = false;

void drawFilledEllipse()

{

glBegin(GL\_TRIANGLE\_FAN);

glVertex2d(center.first, center.second);

for (double angle = 1.0; angle < 361.0; angle += 0.2)

{

double x = center.first + sin(angle) \* minorRadius;

double y = center.second + cos(angle) \* majorRadius;

glVertex2d(x, y);

}

glEnd();

}

void drawEmptyEllipse()

{

glBegin(GL\_LINE\_LOOP);

const int loops = 300;

for (int i = 0; i <= loops; i++)

{

double angle = 2 \* M\_PI \* i / loops;

double x = minorRadius \* cos(angle);

double y = majorRadius \* sin(angle);

glVertex2d(x + center.first, y + center.second);

}

glEnd();

}

void renderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Set black background with white foreground

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor4f(1.0, 1.0, 1.0, 1.0);

filled ? drawFilledEllipse() : drawEmptyEllipse();

glutSwapBuffers();

}

void reshapeScene(GLsizei width, GLsizei height)

{

// Find new aspect ratio

if (height == 0)

height = 1;

GLdouble aspect = (GLdouble)width / (GLdouble)height;

// Set the viewport to cover the new window

glViewport(0, 0, width, height);

// Set the aspect ratio of the clipping area to match the viewport

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (width >= height)

{

// aspect >= 1, set the height from -1 to 1, with larger width

gluOrtho2D(-1.0 \* aspect, 1.0 \* aspect, -1.0, 1.0);

}

else

{

// aspect < 1, set the width to -1 to 1, with larger height

gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);

}

}

int main(int argc, char\*\* argv)

{

cout << "Enter midpoint (x,y) [" << center.first << "," << center.second << "]: ";

string input;

getline(cin, input);

stringstream ss(input);

char split;

ss >> center.first >> split >> center.second;

cout << "Enter minor radius (along x axis) [" << minorRadius << "]: ";

getline(cin, input);

ss = stringstream(input);

ss >> minorRadius;

cout << "Enter major radius (along y axis) [" << majorRadius << "]: ";

getline(cin, input);

ss = stringstream(input);

ss >> majorRadius;

cout << "Fill the ellipse (y/n) [" << (filled ? "y" : "n") << "]: ";

getline(cin, input);

ss = stringstream(input);

char temp = filled ? 'y' : 'n';

ss >> temp;

if (tolower(temp) == 'y')

filled = true;

else

filled = false;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowPosition(50, 50);

glutInitWindowSize(640, 480);

glutCreateWindow("Ellipse");

glEnable(GL\_DEPTH\_TEST);

center.first /= 10; center.second /= 10;

minorRadius /= 10; majorRadius /= 10;

glutDisplayFunc(renderScene);

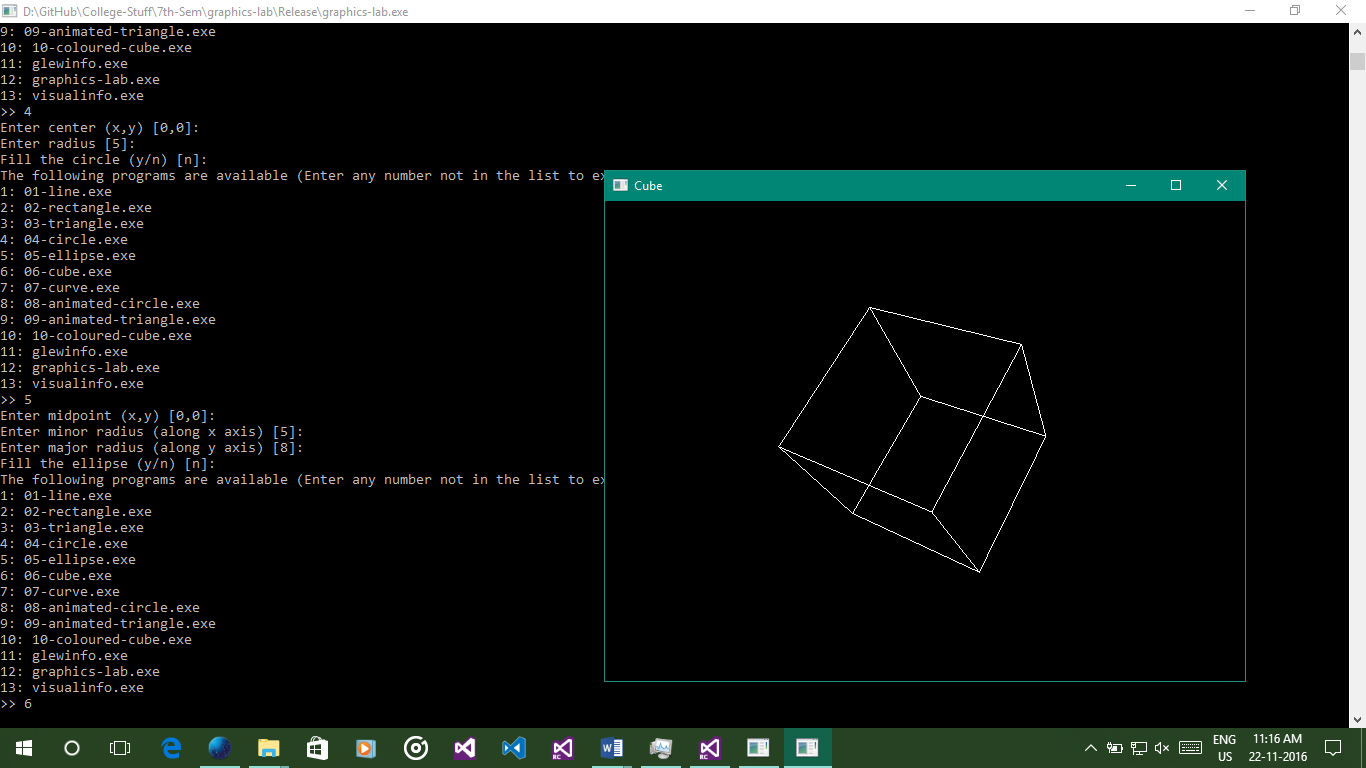
glutReshapeFunc(reshapeScene);

glutMainLoop();

return 0;

}

# Cube



#include "GL/freeglut.h"

#include <set>

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

double angleCube = 0.0;

int refreshMills = 15;

void renderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Set black background with white foreground

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor4f(1.0, 1.0, 1.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslated(0.0, 0.0, -7.0);

glRotated(angleCube, 1.0, 1.0, 1.0);

// Start rendering the quadrilateral primitive

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

glBegin(GL\_QUADS);

// Top face (y = 1.0)

glVertex3d(1.0, 1.0, -1.0);

glVertex3d(-1.0, 1.0, -1.0);

glVertex3d(-1.0, 1.0, 1.0);

glVertex3d(1.0, 1.0, 1.0);

// Bottom face (y = -1.0)

glVertex3d(1.0, -1.0, 1.0);

glVertex3d(-1.0, -1.0, 1.0);

glVertex3d(-1.0, -1.0, -1.0);

glVertex3d(1.0, -1.0, -1.0);

// Front face (z = 1.0)

glVertex3d(1.0, 1.0, 1.0);

glVertex3d(-1.0, 1.0, 1.0);

glVertex3d(-1.0, -1.0, 1.0);

glVertex3d(1.0, -1.0, 1.0);

// Back face (z = -1.0)

glVertex3d(1.0, -1.0, -1.0);

glVertex3d(-1.0, -1.0, -1.0);

glVertex3d(-1.0, 1.0, -1.0);

glVertex3d(1.0, 1.0, -1.0);

// Left face (x = -1.0)

glVertex3d(-1.0, 1.0, 1.0);

glVertex3d(-1.0, 1.0, -1.0);

glVertex3d(-1.0, -1.0, -1.0);

glVertex3d(-1.0, -1.0, 1.0);

// Right face (x = 1.0)

glVertex3d(1.0, 1.0, -1.0);

glVertex3d(1.0, 1.0, 1.0);

glVertex3d(1.0, -1.0, 1.0);

glVertex3d(1.0, -1.0, -1.0);

glEnd();

glutSwapBuffers();

angleCube -= 0.15;

}

void timer(int value)

{

glutPostRedisplay();

glutTimerFunc(refreshMills, timer, 0);

}

void reshapeScene(GLsizei width, GLsizei height)

{

// Find new aspect ratio

if (height == 0)

height = 1;

GLdouble aspect = (GLdouble)width / (GLdouble)height;

// Set the viewport to cover the new window

glViewport(0, 0, width, height);

// Set the aspect ratio of the clipping area to match the viewport

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0f, aspect, 0.1f, 100.0f);

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE);

glutInitWindowPosition(50, 50);

glutInitWindowSize(640, 480);

glutCreateWindow("Cube");

glutDisplayFunc(renderScene);

glutReshapeFunc(reshapeScene);

glClearColor(0.0, 0.0, 0.0, 1.0);

glClearDepth(1.0);

glEnable(GL\_DEPTH\_TEST);

glDepthFunc(GL\_LEQUAL);

glShadeModel(GL\_SMOOTH);

glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST);

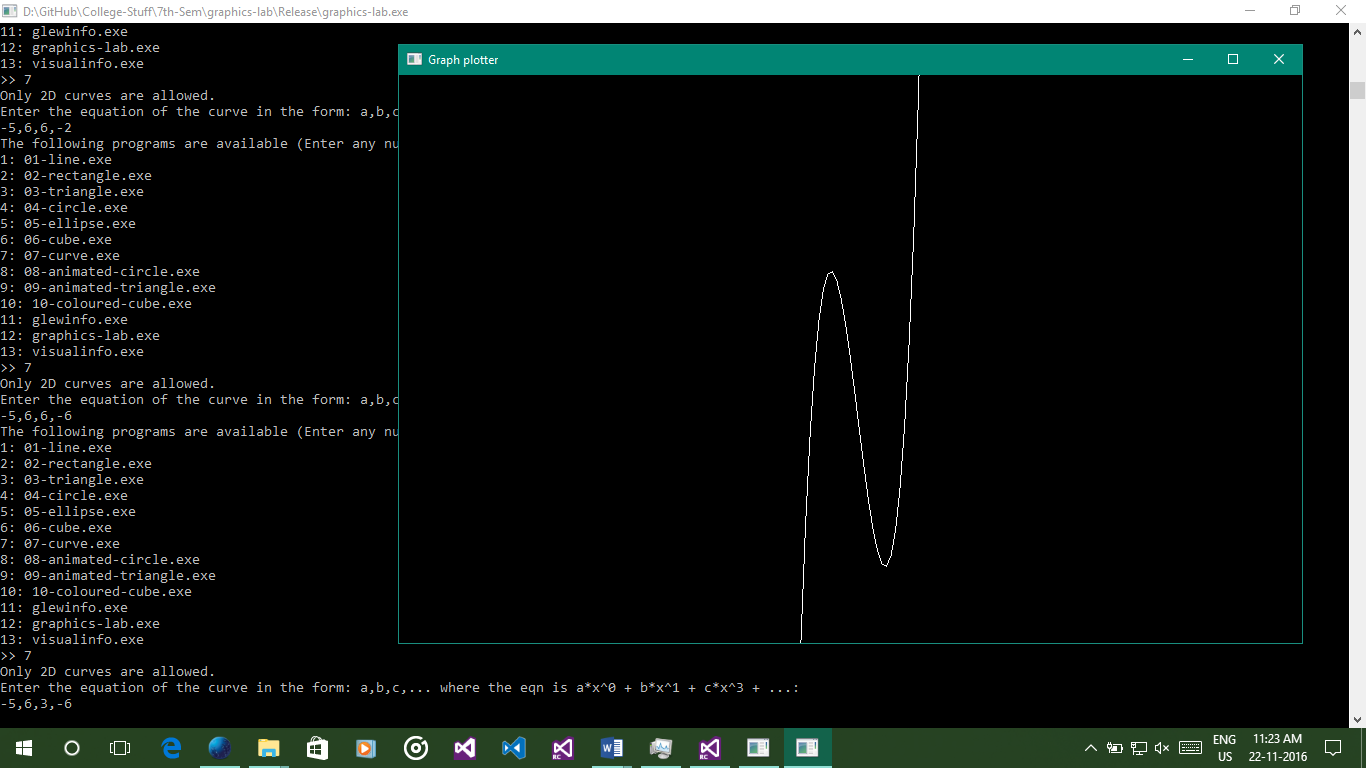
glutTimerFunc(0, timer, 0);

glutMainLoop();

return 0;

}

# Curve



#include <iostream>

#include <string>

#include <sstream>

#include <vector>

#include <math.h>

#include <GL/freeglut.h>

using namespace std;

auto y = double(0.0);

vector<double> p;

/\* Sample func itself \*/

double func(float x)

{

y = 0.0;

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

y += p[i] \* pow(x, i);

return y;

}

/\* Function plotting func \*/

void draw(double(\*func)(float), float x1, float x2, float y1, float y2, int N)

{

float x, dx = 1.0 / N;

glPushMatrix(); /\* GL\_MODELVIEW is default \*/

glScalef(1.0 / (x2 - x1), 1.0 / (y2 - y1), 1.0);

glTranslatef(-x1, -y1, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_LINE\_STRIP);

for (x = x1; x < x2; x += dx)

{

glVertex2f(x, func(x));

}

glEnd();

glPopMatrix();

};

/\* Redrawing func \*/

void redraw(void)

{

glClearColor(0, 0, 0, 0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

draw(func, -10, 10, 0, 50, 50);

glutSwapBuffers();

};

/\* Idle proc. Redisplays, if called. \*/

void idle(void)

{

glutPostRedisplay();

};

/\* Key press processing \*/

void key(unsigned char c, int x, int y)

{

if (c == 27) exit(0);

};

/\* Window reashape \*/

void reshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0, 1, 0, 1, -1, 1);

glMatrixMode(GL\_MODELVIEW);

};

/\* Main function \*/

int main(int argc, char \*\*argv)

{

cout << "Only 2D curves are allowed." << endl;

cout << "Enter the equation of the curve in the form: a,b,c,... where the eqn is a\*x^0 + b\*x^1 + c\*x^3 + ...: " << endl;

string curve;

getline(cin, curve);

stringstream ss(curve);

while (!ss.eof())

{

double temp;

char split;

ss >> temp >> split;

p.push\_back(temp);

}

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE);

glutCreateWindow("Graph plotter");

/\* Register GLUT callbacks. \*/

glutDisplayFunc(redraw);

glutKeyboardFunc(key);

glutReshapeFunc(reshape);

glutIdleFunc(idle);

/\* Init the GL state \*/

glLineWidth(1.0);

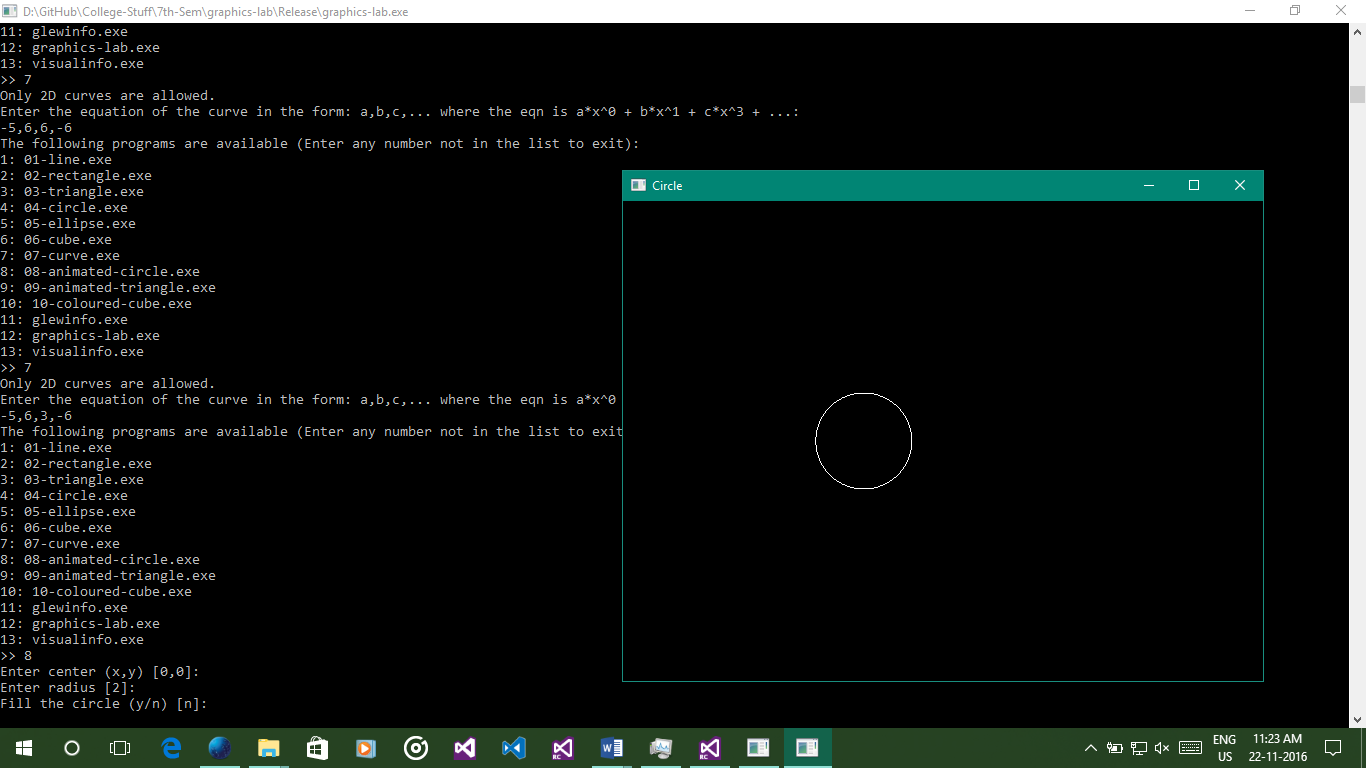
/\* Main loop \*/

glutMainLoop();

return 0;

}

# Animated Circle



#include "GL/freeglut.h"

#include <set>

#include <iostream>

#include <string>

#include <sstream>

#define \_USE\_MATH\_DEFINES

#include <math.h>

using namespace std;

auto center = pair<double, double>(0, 0);

double radius = 2.0;

bool filled = false;

int refreshMills = 10;

double circlePos = -10.0;

double delta = 0.01;

int direction = 1;

void drawFilledCircle()

{

glBegin(GL\_TRIANGLE\_FAN);

glVertex2d(center.first + circlePos, center.second);

for (double angle = 1.0; angle < 361.0; angle += 0.2)

{

double x = center.first + sin(angle) \* radius;

double y = center.second + cos(angle) \* radius;

glVertex2d(x + circlePos, y);

}

circlePos += direction \* delta;

glEnd();

}

void drawEmptyCircle()

{

// Start rendering the circle

glBegin(GL\_LINE\_LOOP);

const int loops = 300;

for (int i = 0; i <= loops; i++)

{

double angle = 2 \* M\_PI \* i / loops;

double x = radius \* cos(angle);

double y = radius \* sin(angle);

glVertex2d(x + center.first + circlePos, y + center.second);

}

circlePos += direction \* delta;

glEnd();

}

void renderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Set black background with white foreground

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor4f(1.0, 1.0, 1.0, 1.0);

if (direction == 1 && circlePos > 1.0)

direction = -1;

else if (direction == -1 && circlePos < -1.0)

direction = 1;

filled ? drawFilledCircle() : drawEmptyCircle();

glutSwapBuffers();

}

void timer(int value)

{

glutPostRedisplay();

glutTimerFunc(refreshMills, timer, 0);

}

void reshapeScene(GLsizei width, GLsizei height)

{

// Find new aspect ratio

if (height == 0)

height = 1;

GLdouble aspect = (GLdouble)width / (GLdouble)height;

// Set the viewport to cover the new window

glViewport(0, 0, width, height);

// Set the aspect ratio of the clipping area to match the viewport

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (width >= height)

{

// aspect >= 1, set the height from -1 to 1, with larger width

gluOrtho2D(-1.0 \* aspect, 1.0 \* aspect, -1.0, 1.0);

}

else

{

// aspect < 1, set the width to -1 to 1, with larger height

gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);

}

}

int main(int argc, char\*\* argv)

{

cout << "Enter center (x,y) [" << center.first << "," << center.second << "]: ";

string input;

getline(cin, input);

stringstream ss(input);

char split;

ss >> center.first >> split >> center.second;

cout << "Enter radius [" << radius << "]: ";

getline(cin, input);

ss = stringstream(input);

ss >> radius;

cout << "Fill the circle (y/n) [" << (filled ? "y" : "n") << "]: ";

getline(cin, input);

ss = stringstream(input);

char temp = filled ? 'y' : 'n';

ss >> temp;

if (tolower(temp) == 'y')

filled = true;

else

filled = false;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowPosition(50, 50);

glutInitWindowSize(640, 480);

glutCreateWindow("Circle");

glEnable(GL\_DEPTH\_TEST);

center.first /= 10; center.second /= 10;

radius /= 10;

circlePos /= 10;

glutDisplayFunc(renderScene);

glutReshapeFunc(reshapeScene);

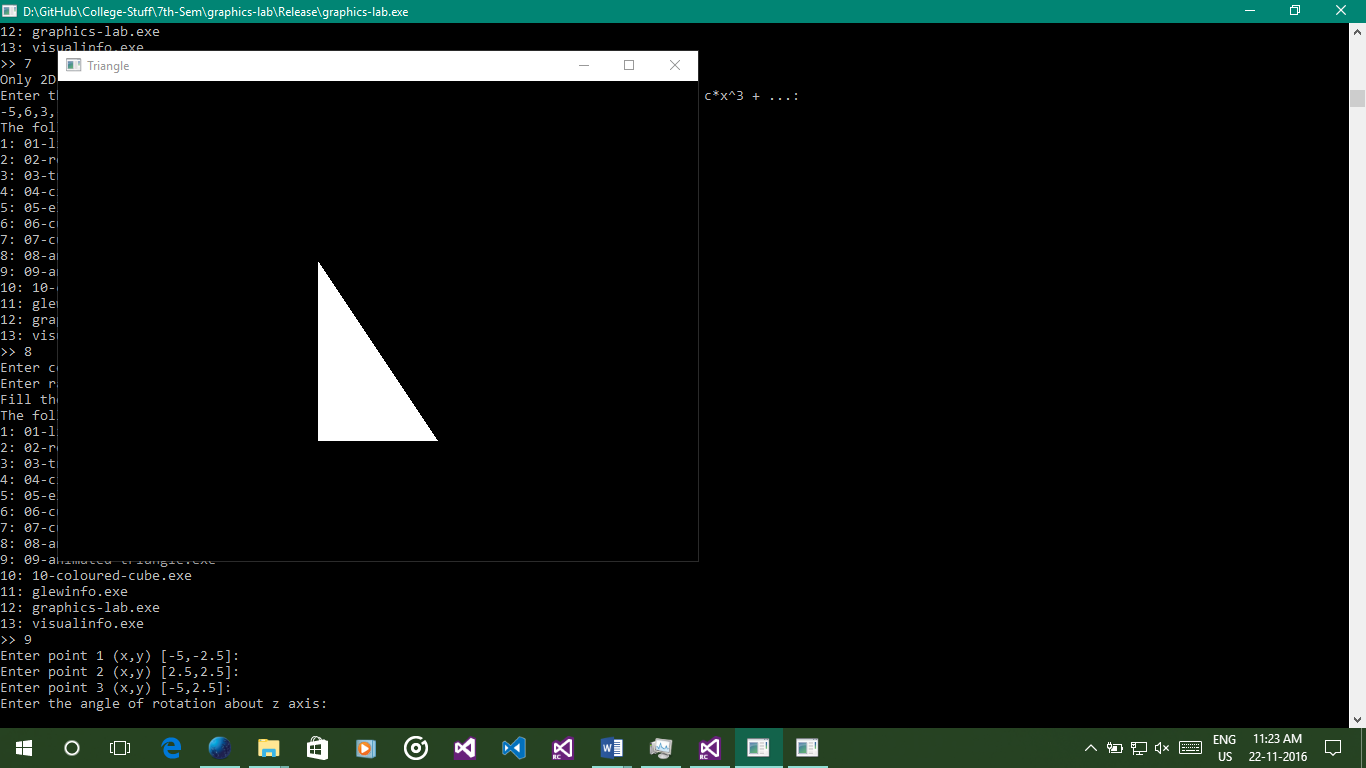
glutTimerFunc(0, timer, 0);

glutMainLoop();

return 0;

}

# Animated Triangle



#include "GL/freeglut.h"

#include <set>

#include <iostream>

#include <string>

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <sstream>

using namespace std;

auto point1 = pair<double, double>(-5, -2.5);

auto point2 = pair<double, double>(2.5, 2.5);

auto point3 = pair<double, double>(-5, 2.5);

double angle = 90;

int refreshMills = 3000;

void renderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Set black background with white foreground

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor4f(1.0, 1.0, 1.0, 1.0);

// Start rendering the triangle primitive

glBegin(GL\_TRIANGLES);

glVertex2d(point1.first, point1.second);

glVertex2d(point2.first, point2.second);

glVertex2d(point3.first, point3.second);

glEnd();

glutSwapBuffers();

glRotated(angle, 0, 0, 1);

}

void timer(int value)

{

glutPostRedisplay();

glutTimerFunc(refreshMills, timer, 0);

}

void reshapeScene(GLsizei width, GLsizei height)

{

// Find new aspect ratio

if (height == 0)

height = 1;

GLdouble aspect = (GLdouble)width / (GLdouble)height;

// Set the viewport to cover the new window

glViewport(0, 0, width, height);

// Set the aspect ratio of the clipping area to match the viewport

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (width >= height)

{

// aspect >= 1, set the height from -1 to 1, with larger width

gluOrtho2D(-1.0 \* aspect, 1.0 \* aspect, -1.0, 1.0);

}

else

{

// aspect < 1, set the width to -1 to 1, with larger height

gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);

}

}

int main(int argc, char\*\* argv)

{

cout << "Enter point 1 (x,y) [" << point1.first << "," << point1.second << "]: ";

string input;

getline(cin, input);

stringstream ss(input);

char split;

ss >> point1.first >> split >> point1.second;

cout << "Enter point 2 (x,y) [" << point2.first << "," << point2.second << "]: ";

getline(cin, input);

ss = stringstream(input);

ss >> point2.first >> split >> point2.second;

cout << "Enter point 3 (x,y) [" << point3.first << "," << point3.second << "]: ";

getline(cin, input);

ss = stringstream(input);

ss >> point3.first >> split >> point3.second;

cout << "Enter the angle of rotation about z axis: ";

getline(cin, input);

ss = stringstream(input);

ss >> angle;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowPosition(50, 50);

glutInitWindowSize(640, 480);

glutCreateWindow("Triangle");

glEnable(GL\_DEPTH\_TEST);

point1.first /= 10; point1.second /= 10;

point2.first /= 10; point2.second /= 10;

point3.first /= 10; point3.second /= 10;

glutDisplayFunc(renderScene);

glutReshapeFunc(reshapeScene);

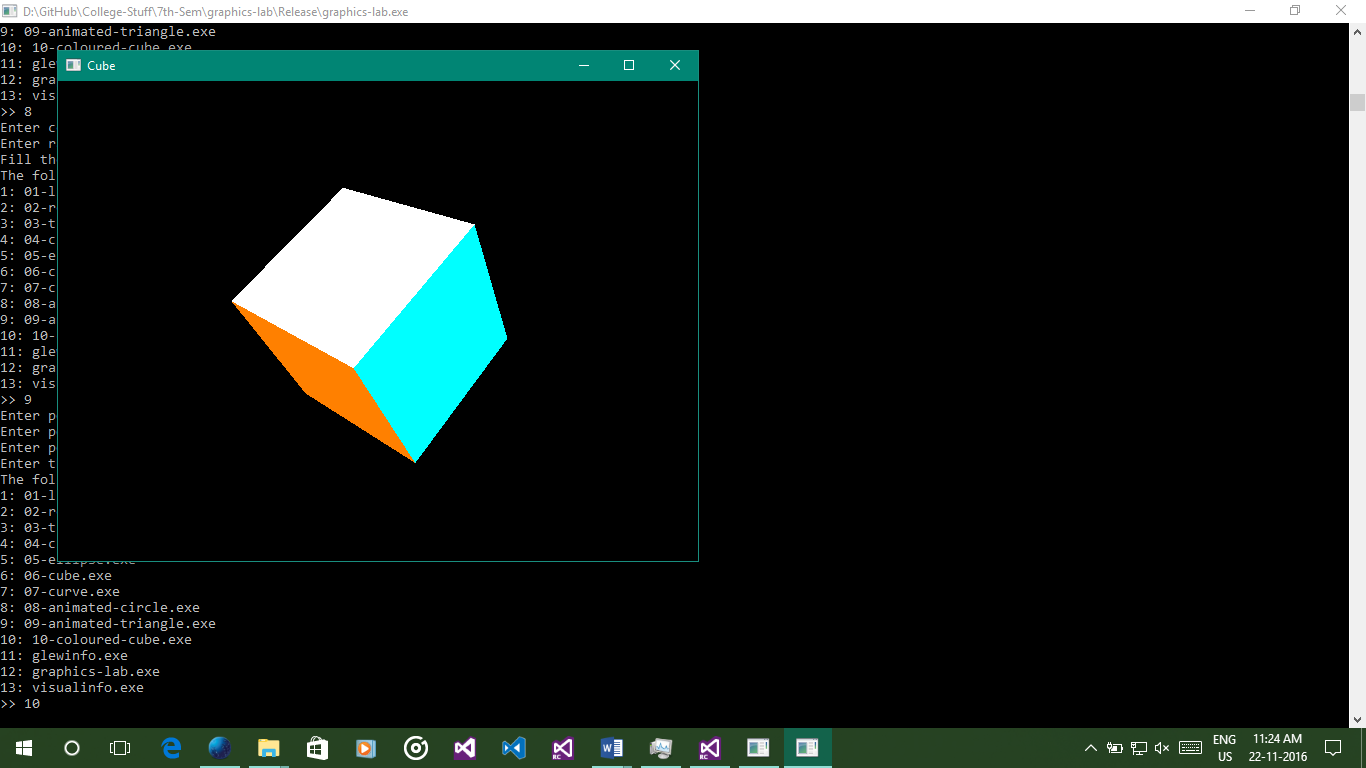
glutTimerFunc(0, timer, 0);

glutMainLoop();

return 0;

}

# Coloured Cube



#include "GL/freeglut.h"

#include <set>

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

double angleCube = 0.0;

int refreshMills = 15;

void renderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslated(0.0, 0.0, -7.0);

glRotated(angleCube, 1.0, 1.0, 1.0);

// Start rendering the quadrilateral primitive

glBegin(GL\_QUADS);

// Top face (y = 1.0)

glColor3d(0.0, 1.0, 1.0);

glVertex3d(1.0, 1.0, -1.0);

glVertex3d(-1.0, 1.0, -1.0);

glVertex3d(-1.0, 1.0, 1.0);

glVertex3d(1.0, 1.0, 1.0);

// Bottom face (y = -1.0)

glColor3d(1.0, 1.0, 0.0);

glVertex3d(1.0, -1.0, 1.0);

glVertex3d(-1.0, -1.0, 1.0);

glVertex3d(-1.0, -1.0, -1.0);

glVertex3d(1.0, -1.0, -1.0);

// Front face (z = 1.0)

glColor3d(1.0, 0.0, 1.0);

glVertex3d(1.0, 1.0, 1.0);

glVertex3d(-1.0, 1.0, 1.0);

glVertex3d(-1.0, -1.0, 1.0);

glVertex3d(1.0, -1.0, 1.0);

// Back face (z = -1.0)

glColor3d(1.0, 0.5, 0.0);

glVertex3d(1.0, -1.0, -1.0);

glVertex3d(-1.0, -1.0, -1.0);

glVertex3d(-1.0, 1.0, -1.0);

glVertex3d(1.0, 1.0, -1.0);

// Left face (x = -1.0)

glColor3d(0.0, 0.0, 1.0);

glVertex3d(-1.0, 1.0, 1.0);

glVertex3d(-1.0, 1.0, -1.0);

glVertex3d(-1.0, -1.0, -1.0);

glVertex3d(-1.0, -1.0, 1.0);

// Right face (x = 1.0)

glColor3d(1.0, 1.0, 1.0);

glVertex3d(1.0, 1.0, -1.0);

glVertex3d(1.0, 1.0, 1.0);

glVertex3d(1.0, -1.0, 1.0);

glVertex3d(1.0, -1.0, -1.0);

glEnd();

glutSwapBuffers();

angleCube -= 0.15;

}

void timer(int value)

{

glutPostRedisplay();

glutTimerFunc(refreshMills, timer, 0);

}

void reshapeScene(GLsizei width, GLsizei height)

{

// Find new aspect ratio

if (height == 0)

height = 1;

GLdouble aspect = (GLdouble)width / (GLdouble)height;

// Set the viewport to cover the new window

glViewport(0, 0, width, height);

// Set the aspect ratio of the clipping area to match the viewport

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0f, aspect, 0.1f, 100.0f);

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE);

glutInitWindowPosition(50, 50);

glutInitWindowSize(640, 480);

glutCreateWindow("Cube");

glutDisplayFunc(renderScene);

glutReshapeFunc(reshapeScene);

glClearColor(0.0, 0.0, 0.0, 1.0);

glClearDepth(1.0);

glEnable(GL\_DEPTH\_TEST);

glDepthFunc(GL\_LEQUAL);

glShadeModel(GL\_SMOOTH);

glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST);

glutTimerFunc(0, timer, 0);

glutMainLoop();

return 0;

}