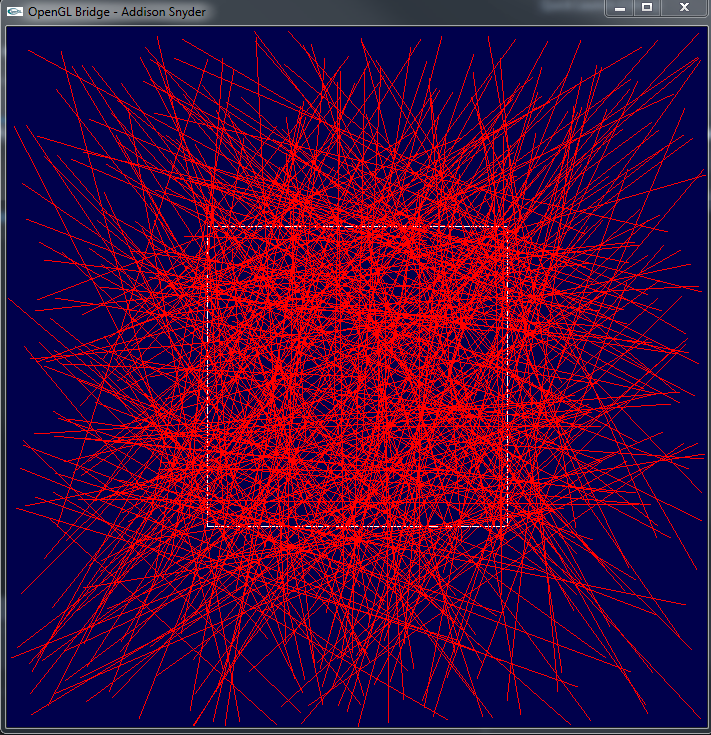
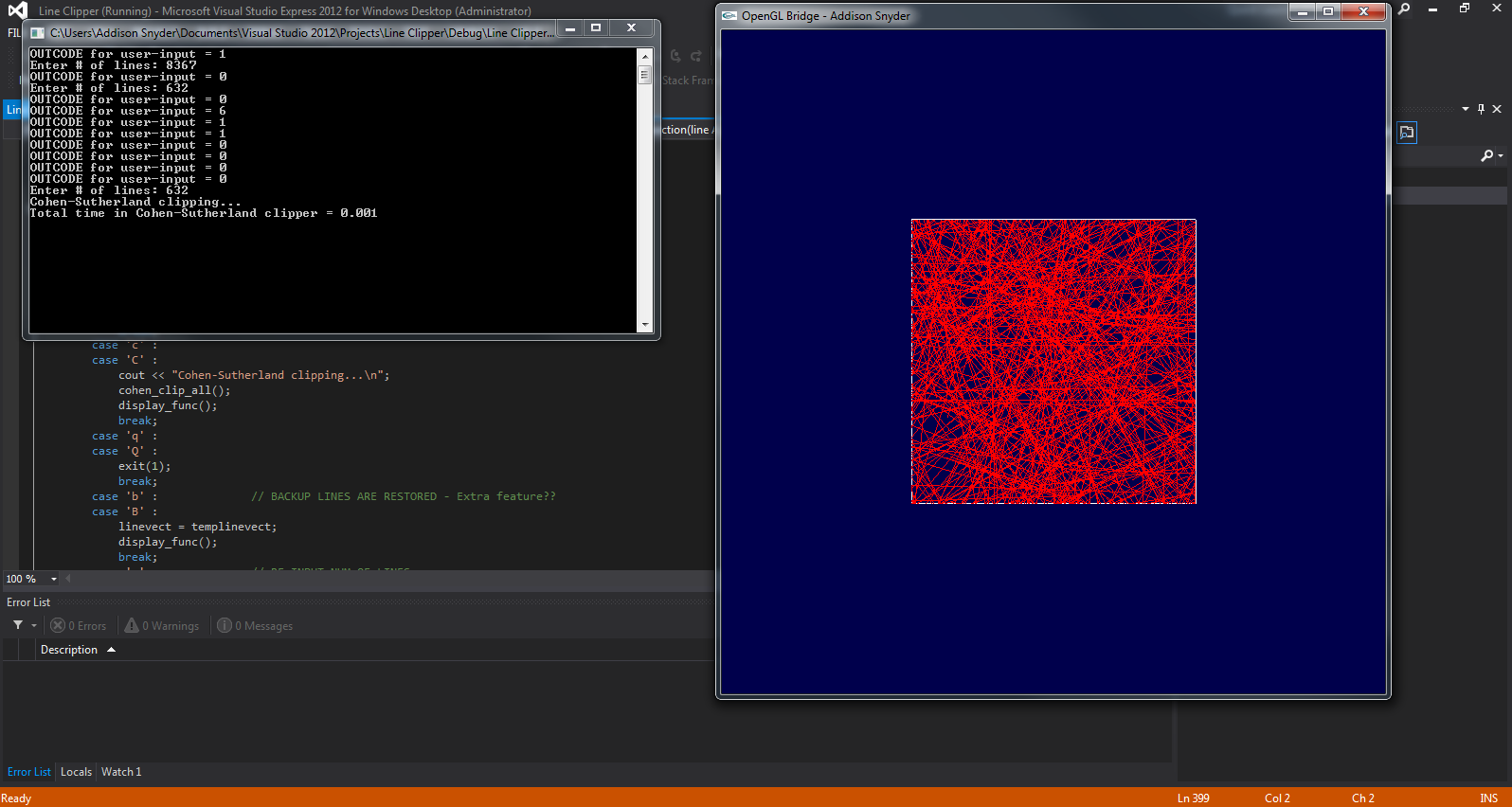


Long story short, I had completely forgotten or failed to see that this assignment was due that night... I just had enough time to get all of the objectives completed with about 5 minutes left on the clock after coding for 5 hours. Definitely not very good code, but it got the job done and performed well enough. There are many areas where better layout could have been used, such as variable declaration. Also, bracket use was somewhat inconsistent (missing brackets for single statement if/for). Still, this is my favorite piece of code because it creates a nice visual result, and helped me learn to think on my feet for code. NOT the right way to do things, but still rewarding to pull it off…





// Press 'b' to backup to an earlier version of the lines

// Addison Snyder - Program 3 - Clipping Algorithms

// See function init. section for descriptions

#include "stdafx.h"

#include <stdlib.h>

#include <glut.h>

#include <iostream>

#include <vector>

#include <stdio.h>

#include <time.h>

#include <math.h>

#include "GL/cs\_graphics\_setup.h"

#define WINDOW\_XS 700

#define WINDOW\_YS 700

#define WINDOW\_NAME "OpenGL Bridge - Addison Snyder"

char\* msg = "Press 'r' and enter number of lines in command prompt!";

using namespace std;

typedef struct pt

{

GLint x1, y1;

}pt;

typedef struct line

{

GLint x1, y1, x2, y2;

}line;

bool generated = false;

bool change = false;

int num\_lines = 10;

const int TOP = 500, BOTTOM = 200, RIGHT = 500, LEFT = 200;

line Top = {0, 500, 700, 500};

line Bottom = {0, 200, 700, 200};

line Right = {500, 0, 500, 700};

line Left = {200, 0, 200, 700};

const int INSIDE = 0; // 0000

const int LEFT\_CODE = 1; // 0001

const int RIGHT\_CODE = 2; // 0010

const int BOTTOM\_CODE = 4; // 0100

const int TOP\_CODE = 8; // 1000

vector<line> linevect;

vector<line> templinevect;

clock\_t start;

double duration;

line mouseline;

int mouse = 0;

pt mouse1;

int mousecode;

bool input = false;

void display\_func(void); // Basic display

void keyboard\_func(unsigned char c, int x, int y); // keyboard input handling

void mouse\_func(int button, int state, int x, int y); // mouse handling

void output(int x, int y, float r, float g, float b, void \*font, char \*string); // crude text output

void generate\_lines(); // randomly generates lines and places them into 'linevect'

void draw\_lines(); // draws all lines in linevect

line simple\_clip(line A); // simple clips ONE line

pt get\_intersection(line A, line edge); // calculates intersection of two lines for simple clip algorithm

bool inter\_valid(pt in); // returns false if there is no intersection between two lines

void simple\_clip\_all(); // simple clips all linevect lines..

bool out(line in); // returns false if line is completlely outside screen

void cohen\_clip\_all(); // Cohen-Sutherland clipping of ALL lines

line cohen\_clip(line A); // same with only ONE line

int get\_code(pt in); // generates OUTCODE for a point

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

{

srand (time(NULL));

glutInit(&argc, argv);

init\_setup(WINDOW\_XS, WINDOW\_YS, WINDOW\_NAME);

//cin >> num\_lines;

generate\_lines();

glutDisplayFunc(display\_func); // call back for display event

glutKeyboardFunc(keyboard\_func); // call back for keyboard event

glutMouseFunc(mouse\_func); // call back for mouse event

glutMainLoop();

return 1;

} // end of main()

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void display\_func(void)

{

glClearColor(0.0, 0.0, 0.3, 1.0); // background color (white)

glClear(GL\_COLOR\_BUFFER\_BIT); // clearing the buffer not to keep the color

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_LINES); //Outline Screen

glVertex2f(200, 200);

glVertex2f(500, 200);

glVertex2f(500, 500);

glVertex2f(500, 200);

glVertex2f(500, 500);

glVertex2f(200, 500);

glVertex2f(200, 200);

glVertex2f(200, 500);

glEnd();

if(!input)

output(10, 10, 1,1,1, GLUT\_BITMAP\_HELVETICA\_18, msg);

if(generated)

{

draw\_lines();

}

glFlush();

} // end of display\_func()

void keyboard\_func(unsigned char c, int x, int y)

{

switch(c){

case 's' :

case 'S' :

cout << "Simple clipping...\n";

simple\_clip\_all();

display\_func();

break;

case 'c' :

case 'C' :

cout << "Cohen-Sutherland clipping...\n";

cohen\_clip\_all();

display\_func();

break;

case 'q' :

case 'Q' :

exit(1);

break;

case 'b' : // BACKUP LINES ARE RESTORED - Extra feature??

case 'B' :

linevect = templinevect;

display\_func();

break;

case 'r' : // RE-INPUT NUM OF LINES

case 'R' :

cout << "Enter # of lines: ";

cin >> num\_lines;

generate\_lines();

input = true;

display\_func();

break;

} // end of switch

} // end of keyboard\_func()

void mouse\_func(int button, int state, int x, int y)

{

if(button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)

{

mouse++;

if(mouse%2 == 1) // Every two points, the line is diplayed

{

mouse1.x1 = x;

mouse1.y1 = y;

mousecode = get\_code(mouse1);

cout << "OUTCODE for user-input = " << mousecode << "\n";

mouseline.x1 = x;

mouseline.y1 = 700-y;

}

else

{

mouse1.x1 = x;

mouse1.y1 = y;

mousecode = get\_code(mouse1);

cout << "OUTCODE for user-input = " << mousecode << "\n";

mouseline.x2 = x;

mouseline.y2 = 700-y;

linevect.push\_back(mouseline); // This line is UNCLIPPED, but can be using clipping methods

display\_func();

}

glutPostRedisplay();

}

} // end of mouse\_func()

void output(int x, int y, float r, float g, float b, void \*font, char \*string)

{

glColor3f( r, g, b );

glRasterPos2f(x, y);

int len, i;

len = (int)strlen(string);

for (i = 0; i < len; i++)

glutBitmapCharacter(font, string[i]);

}

void generate\_lines(){

linevect.clear();

line in;

for(int i = 0; i < num\_lines; i++)

{

in.x1 = rand() % 700;

in.y1 = rand() % 700;

in.x2 = rand() % 700;

in.y2 = rand() % 700;

linevect.push\_back(in);

}

generated = true;

return;

}

void draw\_lines(){

glColor3f(1.0, 0, 0);

glBegin(GL\_LINES);

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

{

if(!out(linevect[i])){

glVertex2f(linevect[i].x1, linevect[i].y1);

glVertex2f(linevect[i].x2, linevect[i].y2);

}

}

glEnd();

}

int get\_code(pt in){

int code = INSIDE;

if (in.x1 < 200) // to the left of clip window

code |= LEFT\_CODE;

else if (in.x1 > 500) // to the right of clip window

code |= RIGHT\_CODE;

if (in.y1 < 200) // below the clip window

code |= BOTTOM\_CODE;

else if (in.y1 > 500) // above the clip window

code |= TOP\_CODE;

return code;

}

line cohen\_clip(line in){ // Code adapted from wikipedia to fit my program.

pt A = {in.x1, in.y1};

pt B = {in.x2, in.y2};

int code1 = get\_code(A);

int code2 = get\_code(B);

bool done = false;

double x, y;

while (1) {

if (!(code1 | code2)) { // Bitwise OR is 0. Trivially accept and get out of loop

done = true;

break;

} else if (code1 & code2) { // Bitwise AND is not 0. Trivially reject and get out of loop

break;

} else {

// failed both tests, so calculate the line segment to clip

// from an outside point to an intersection with clip edge

// At least one endpoint is outside the clip rectangle; pick it.

int outcodeOut = code1 ? code1 : code2;

// Now find the intersection point;

// use formulas y = y0 + slope \* (x - x0), x = x0 + (1 / slope) \* (y - y0)

if (outcodeOut & TOP\_CODE) { // point is above the clip rectangle

x = A.x1 + (B.x1 - A.x1) \* (500 - A.y1) / (B.y1 - A.y1);

y = 500;

} else if (outcodeOut & BOTTOM\_CODE) { // point is below the clip rectangle

x = A.x1 + (B.x1 - A.x1) \* (200 - A.y1) / (B.y1 - A.y1);

y = 200;

} else if (outcodeOut & RIGHT\_CODE) { // point is to the right of clip rectangle

y = A.y1 + (B.y1 - A.y1) \* (500 - A.x1) / (B.x1 - A.x1);

x = 500;

} else if (outcodeOut & LEFT\_CODE) { // point is to the left of clip rectangle

y = A.y1 + (B.y1 - A.y1) \* (200 - A.x1) / (B.x1 - A.x1);

x = 200;

}

// Now we move outside point to intersection point to clip

// and get ready for next pass.

if (outcodeOut == code1) {

A.x1 = x;

A.y1 = y;

code1 = get\_code(A);

} else {

B.x1 = x;

B.y1 = y;

code2 = get\_code(B);

}

}

}

if (done) {

line out = {A.x1,A.y1,B.x1,B.y1};

return out;

}

line out = {A.x1,A.y1,B.x1,B.y1};

return out;

}

void cohen\_clip\_all(){

templinevect = linevect;

start = clock();

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

{

linevect[i] = cohen\_clip(linevect[i]);

}

cout << "Total time in Cohen-Sutherland clipper = " << (clock() - start) / (double)CLOCKS\_PER\_SEC << "\n";

}

void simple\_clip\_all(){

templinevect = linevect;

start = clock();

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

{

linevect[i] = simple\_clip(linevect[i]);

}

cout << "Total time in simple clipper = " << (clock() - start) / (double)CLOCKS\_PER\_SEC << "\n";

}

line simple\_clip(line A){

//cout << "Line IN = {" << A.x1 << ", " << A.y1 << ", " << A.x2 << ", " << A.y2 << "}\n";

pt change;

if(inter\_valid(get\_intersection(A, Left)))

{

change = get\_intersection(A, Left);

if(A.x1 < A.x2)

{

A.x1 = change.x1;

A.y1 = change.y1;

}

else

{

A.x2 = change.x1;

A.y2 = change.y1;

}

}

if(inter\_valid(get\_intersection(A, Right)))

{

change = get\_intersection(A, Right);

if(A.x1 > A.x2)

{

A.x1 = change.x1;

A.y1 = change.y1;

}

else

{

A.x2 = change.x1;

A.y2 = change.y1;

}

}

if(inter\_valid(get\_intersection(A, Top)))

{

change = get\_intersection(A, Top);

if(A.y1 > A.y2)

{

A.x1 = change.x1;

A.y1 = change.y1;

}

else

{

A.x2 = change.x1;

A.y2 = change.y1;

}

}

if(inter\_valid(get\_intersection(A, Bottom)))

{

change = get\_intersection(A, Bottom);

if(A.y1 < A.y2)

{

A.x1 = change.x1;

A.y1 = change.y1;

}

else

{

A.x2 = change.x1;

A.y2 = change.y1;

}

}

//cout << "Line OUT = {" << A.x1 << ", " << A.y1 << ", " << A.x2 << ", " << A.y2 << "}\n";

return A;

}

bool inter\_valid(pt in){

if(in.x1 == -1 && in.y1 == -1)

return false;

else

return true;

}

bool out(line in){

if((in.x1 < 200 && in.x2 < 200) || (in.x1 > 500 && in.x2 > 500) || (in.y1 < 200 && in.y2 < 200) || (in.y1 > 500 && in.y2 > 500))

return true;

else

return false;

}

pt get\_intersection(line A, line edge) // code modified from http://flassari.is/2008/11/line-line-intersection-in-cplusplus/

{

pt out = {-1,-1};

float x1 = A.x1, x2 = A.x2, x3 = edge.x1, x4 = edge.x2;

float y1 = A.y1, y2 = A.y2, y3 = edge.y1, y4 = edge.y2;

float d = (x1 - x2) \* (y3 - y4) - (y1 - y2) \* (x3 - x4);

// If d is zero, there is no intersection

if (d == 0) return out;

// Get the x and y

float pre = (x1\*y2 - y1\*x2), post = (x3\*y4 - y3\*x4);

float x = ( pre \* (x3 - x4) - (x1 - x2) \* post ) / d;

float y = ( pre \* (y3 - y4) - (y1 - y2) \* post ) / d;

// Check if the x and y coordinates are within both lines

if ( x < min(x1, x2) || x > max(x1, x2) || x < min(x3, x4) || x > max(x3, x4) ) return out;

if ( y < min(y1, y2) || y > max(y1, y2) || y < min(y3, y4) || y > max(y3, y4) ) return out;

// Return the point of intersection

out.x1 = x;

out.y1 = y;

return out;

}