Basics:

#include<iostream>

#include<gl/glut.h>

void myinit() {

glClearColor(1, 1, 1, 0);

glColor3f(0, 0, 0);

glPointSize(10);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 500, 0, 500);

}

void myfunc() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POINTS);

glVertex2f(50, 50);

glVertex2f(100, 100);

glVertex2f(200, 200);

glEnd();

glBegin(GL\_LINE\_LOOP);

glVertex2f(300, 300);

glVertex2f(300, 400);

glVertex2f(400, 400);

glVertex2f(400, 300);

glEnd();

glFlush();

}

int main(int argc, char\* argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Practice");

glutDisplayFunc(myfunc);

myinit();

glutMainLoop();

return 1;

}

DDA:

#include<iostream>

#include<gl/glut.h>

#include<math.h>

using namespace std;

struct Point{

int x;

int y;

} p1, p2;

void myinit() {

glClearColor(1, 1, 1, 0);

glColor3f(0, 0, 0);

glPointSize(10);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 500, 0, 500);

}

int sign(float n){

if (n == 0) return 0;

else if (n < 0) return -1;

else return 1;

}

void line() {

glClear(GL\_COLOR\_BUFFER\_BIT);

//dda line

float x, y, dx, dy, x\_inc, y\_inc;

int steps;

dx = p2.x - p1.x;

dy = p2.y - p1.y;

if (abs(dx) >= abs(dy)) {

steps = abs(dx);

}

else {

steps = abs(dy);

}

x\_inc = dx / steps;

y\_inc = dy / steps;

x = p1.x + 0.5 \* sign(x\_inc);

y = p1.x + 0.5 \* sign(y\_inc);

int i = 0;

while (i <= steps) {

glBegin(GL\_POINTS);

cout << x << " " << y << "\n";

glVertex2f(x, y);

glEnd();

x += x\_inc;

y += y\_inc;

i++;

}

glFlush();

}

int main(int argc, char\* argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("DDA");

p1.x = 50;

p1.y = 100;

p2.x = 400;

p2.y = 300;

glutDisplayFunc(line);

myinit();

glutMainLoop();

return 1;

}

Bresenham:

#include<iostream>

#include<gl/glut.h>

#include<math.h>

using namespace std;

typedef struct {

float x;

float y;

}Point;

void myinit() {

glClearColor(1, 1, 1, 0);

glColor3f(0, 0, 0);

glPointSize(2);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 500, 0, 500);

}

void swap(float\* a, float\* b) {

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

void line() {

glClear(GL\_COLOR\_BUFFER\_BIT);

Point p1, p2;

int s1, s2, a, b, p;

float x, y;

bool isSwap = false;

p1.x = 250;

p1.y = 250;

p2.x = 50;

p2.y = 450;

float dx, dy;

dx = abs(p1.x - p2.x);

dy = abs(p1.y - p2.y);

if (p2.x > p1.x) {

s1 = 1;

}

else {

s1 = -1;

}

if (p2.y > p1.y) {

s2 = 1;

}

else {

s2 = -1;

}

if (dy > dx) {

swap(&dx, &dy);

isSwap = true;

}

p = 2 \* dy - dx;

a = 2 \* dy;

b = 2 \* (dy - dx);

x = p1.x;

y = p1.y;

glBegin(GL\_POINTS);

glVertex2f(x, y);

for (int i = 0; i <= dx; i++) {

if (p < 0) {

if (isSwap) {

y += s2;

}

else {

x += s1;

}

p += a;

}

else {

y += s2;

x += s1;

p += b;

}

glVertex2f(x, y);

}

glEnd();

glFlush();

}

int main(int argc, char\* argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Bresenham");

glutDisplayFunc(line);

myinit();

glutMainLoop();

return 1;

}

Circle:

#include<gl/glut.h>

#include<iostream>

#include<math.h>

using namespace std;

typedef struct {

float x;

float y;

float r;

}Circle;

void myinit() {

glClearColor(1, 1, 1, 0);

glColor3f(0, 0, 0);

glPointSize(2);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-250, 250, -250, 250);

}

void circle() {

glClear(GL\_COLOR\_BUFFER\_BIT);

Circle c;

c.x = 100;

c.y = 100;

c.r = 50;

float x, y, p;

x = 0;

y = c.r;

p = 1 - c.r;

glBegin(GL\_POINTS);

while (x <= y) {

x++;

if (p < 0) {

p += 2 \* x + 1;

}

else {

y--;

p += 2 \* x + 1 - 2 \* y;

}

glVertex2f(x+c.x, y+c.y);

glVertex2f(-x + c.x, y + c.y);

glVertex2f(x + c.x, -y + c.y);

glVertex2f(-x + c.x, -y + c.y);

glVertex2f(y + c.x, x + c.y);

glVertex2f(-y + c.x, x + c.y);

glVertex2f(y + c.x, -x + c.y);

glVertex2f(-y + c.x, -x + c.y);

}

glEnd();

glFlush();

}

int main(int argc, char\* argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Circle");

glutDisplayFunc(circle);

myinit();

glutMainLoop();

return 1;

}

Window to viewport:

#include<GL/glut.h>

#include<stdlib.h>

#include<iostream>

using namespace std;

int xw\_min, xw\_max, yw\_min, yw\_max, xv\_min, xv\_max, yv\_min, yv\_max;

int points[3][2];

void myInit() {

glClearColor(1.0, 1.0, 1.0, 0.0);

glPointSize(5);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 640.0, 0.0, 480.0);

}

void windowToViewport() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0f, 1.0f, 0.0f);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xw\_min, yw\_min);

glVertex2f(xw\_max, yw\_min);

glVertex2f(xw\_max, yw\_max);

glVertex2f(xw\_min, yw\_max);

glEnd();

glColor3f(0.0f, 0.0f, 1.0f);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xv\_min, yv\_min);

glVertex2f(xv\_max, yv\_min);

glVertex2f(xv\_max, yv\_max);

glVertex2f(xv\_min, yv\_max);

glEnd();

glColor3f(1.0f, 0.0f, 0.0f);

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 3; i++) {

glVertex2f(points[i][0], points[i][1]);

}

glEnd();

float sx = (xv\_max - xv\_min) / (xw\_max - xw\_min);

float sy = (yv\_max - yv\_min) / (yw\_max - yw\_min);

float xv, yv;

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 3; i++) {

xv = xv\_min + ((points[i][0] - xw\_min) \* sx);

yv = yv\_min + ((points[i][1] - yw\_min) \* sy);

glVertex2f(xv, yv);

}

glEnd();

glFlush();

}

int main(int argc, char\* argv[]) {

cout << "\nEnter xw\_min: ";

cin >> xw\_min;

cout << "\nEnter xw\_max: ";

cin >> xw\_max;

cout << "\nEnter yw\_min: ";

cin >> yw\_min;

cout << "\nEnter yw\_max: ";

cin >> yw\_max;

cout << "\nEnter xv\_min: ";

cin >> xv\_min;

cout << "\nEnter xv\_max: ";

cin >> xv\_max;

cout << "\nEnter yv\_min: ";

cin >> yv\_min;

cout << "\nEnter yv\_max: ";

cin >> yv\_max;

cout << "\nEnter point 1: ";

cin >> points[0][0]>>points[0][1];

cout << "\nEnter point 2: ";

cin >> points[1][0]>>points[1][1];

cout << "\nEnter point 3: ";

cin >> points[2][0]>>points[2][1];

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(640, 480);

glutCreateWindow("Window to Viewport Transformation");

glutDisplayFunc(windowToViewport);

myInit();

glutMainLoop();

return 1;

}

2D:

#include<iostream>

#include<gl/glut.h>

#include<math.h>

int n;

using namespace std;

typedef struct {

float x;

float y;

}Point;

void myinit() {

glClearColor(1, 1, 1, 0);

glColor3f(0, 0, 0);

glMatrixMode(GL\_PROJECTION);

glPointSize(5);

glLoadIdentity();

gluOrtho2D(-250, 250, -250, 250);

}

void translate(float T[][3], float tx, float ty) {

float temp[3][3] = { {1,0,tx},{0,1,ty},{0,0,1} };

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

T[i][j] = temp[i][j];

}

}

}

void multiply(float T[3][3], float P[3][10], float newP[3][10]) {

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < n; ++j) {

newP[i][j] = 0;

for (int k = 0; k < 3; ++k) {

newP[i][j] += T[i][k] \* P[k][j];

}

}

}

}

void multiplyT(float T1[3][3], float T2[3][3], float newT[3][3]) {

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

newT[i][j] = 0;

for (int k = 0; k < 3; ++k) {

newT[i][j] += T1[i][k] \* T2[k][j];

}

}

}

}

void plot(float P[3][10]) {

glBegin(GL\_LINE\_LOOP);

for (int j = 0; j < n; ++j) {

glVertex2f(P[0][j], P[1][j]);

}

glEnd();

for (int i = 0; i < n; ++i) {

for (int j = 0; j < 3; ++j) {

cout << P[j][i] << " ";

}

cout << "\n";

}

cout << "\n";

}

void scale(float T[3][3], float sx, float sy, Point fp) {

float temp[3][3] = { {sx,0,fp.x\*(1-sx)},{0,sy,fp.y\*(1-sy)},{0,0,1} };

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

T[i][j] = temp[i][j];

}

}

}

void rotate(float T[3][3], float a, Point fp) {

a = a \* 3.14 / 180;

float temp[3][3] = { {cos(a),-sin(a),fp.x\*(1-cos(a))+fp.y\*sin(a)},{sin(a),cos(a),fp.y\*(1-cos(a))-fp.x\*sin(a)},{0,0,1} };

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

T[i][j] = temp[i][j];

cout << T[i][j] << " ";

}

}

}

void reflectX(float T[3][3]) {

float temp[3][3] = { {1,0,0},{0,-1,0},{0,0,1} };

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

T[i][j] = temp[i][j];

}

}

}

void reflectY(float T[3][3]) {

float temp[3][3] = { {-1,0,0},{0,1,0},{0,0,1} };

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

T[i][j] = temp[i][j];

}

}

}

void reflectOrigin(float T[3][3]) {

float temp[3][3] = { {-1,0,0},{0,-1,0},{0,0,1} };

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

T[i][j] = temp[i][j];

}

}

}

void reflectXY(float T[3][3]) {

float temp[3][3] = { {0,1,0},{1,0,0},{0,0,1} };

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

T[i][j] = temp[i][j];

}

}

}

void shearX(float T[3][3], float sh) {

float temp[3][3] = { {1,sh,0},{0,1,0},{0,0,1} };

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

T[i][j] = temp[i][j];

}

}

}

void shearY(float T[3][3], float sh) {

float temp[3][3] = { {1,0,0},{sh,1,0},{0,0,1} };

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

T[i][j] = temp[i][j];

}

}

}

void transform() {

glClear(GL\_COLOR\_BUFFER\_BIT);

Point fp;

float T[3][3];

float P[3][10], newP[3][10];

n = 3;

P[0][0] = 0;

P[1][0] = 0;

P[2][0] = 1;

P[0][1] = 50;

P[1][1] = 50;

P[2][1] = 1;

P[0][2] = 100;

P[1][2] = 0;

P[2][2] = 1;

fp.x = 100;

fp.y = 100;

//translate(T, 50,50);

//scale(T, 2, 3, fp);

//rotate(T, 45, fp);

//reflectX(T);

//reflectY(T);

//reflectOrigin(T);

//reflectXY(T);

//shearX(T, 1);

shearY(T, 1);

multiply(T, P,newP);

plot(P);

glColor3f(1, 0, 0);

plot(newP);

glFlush();

}

void composite() {

glClear(GL\_COLOR\_BUFFER\_BIT);

Point fp;

float T1[3][3], T2[3][3], T[3][3];

float P[3][10], newP[3][10];

n = 3;

P[0][0] = 0;

P[1][0] = 0;

P[2][0] = 1;

P[0][1] = 50;

P[1][1] = 50;

P[2][1] = 1;

P[0][2] = 100;

P[1][2] = 0;

P[2][2] = 1;

fp.x = 50;

fp.y = 50;

rotate(T1, 45,fp);

translate(T2, 50, 50);

multiplyT(T1, T2, T);

multiply(T, P, newP);

plot(P);

glColor3f(1, 0, 0);

plot(newP);

glFlush();

}

void animate(int state) {

glClear(GL\_COLOR\_BUFFER\_BIT);

Point fp;

float T[3][3];

float P[3][10], newP[3][10];

n = 2;

P[0][0] = 0;

P[1][0] = 0;

P[2][0] = 1;

P[0][1] = 100;

P[1][1] = 100;

P[2][1] = 1;

fp.x = 0;

fp.y = 0;

rotate(T, state, fp);

//translate(T, state, 0);

multiply(T, P, newP);

glColor3f(1, 0, 0);

plot(newP);

glFlush();

glutTimerFunc(1000, animate, (state + 2) % 360);

}

void demo() {

glutTimerFunc(1000, animate, 0);

}

int main(int argc, char\* argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("2D transforms");

glutDisplayFunc(demo);

myinit();

glutMainLoop();

return 1;

}

Line clipping:

#include<iostream>

#include<gl/glut.h>

using namespace std;

typedef struct {

float x;

float y;

}Point;

float xmin, ymin, xmax, ymax;

void myinit() {

glClearColor(1, 1, 1, 0);

glColor3f(0, 0, 0);

glPointSize(2);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 500, 0, 500);

}

void findRegionCode(Point p, int rc[4]) {

//rc - TBRL

if (p.y > ymax)

rc[0] = 1;

else

rc[0] = 0;

if (p.y < ymin)

rc[1] = 1;

else

rc[1] = 0;

if (p.x > xmax)

rc[2] = 1;

else

rc[2] = 0;

if (p.x < xmin)

rc[3] = 1;

else

rc[3] = 0;

}

bool trivialAccept(int rc0[4], int rc1[4]) {

int res[4];

for (int i = 0; i < 4; ++i) {

res[i] = rc0[i] || rc1[i];

if (res[i] != 0) {

return false;

}

}

return true;

}

bool trivialReject(int rc0[4], int rc1[4]) {

int res[4];

for (int i = 0; i < 4; ++i) {

res[i] = rc0[i] && rc1[i];

if (res[i] == 1) {

return true;

}

}

return false;

}

void CohenPoint(Point \*p, int rc[4], float m ) {

float xClip, yClip;

float x = p->x, y = p->y;

if (rc[0] == 1) {

xClip = x + (ymax - y) / m;

if (xClip <= xmax && xClip >= xmin) {

x = xClip;

y = ymax;

}

}

if (rc[1] == 1) {

xClip = x + (ymin - y) / m;

if (xClip <= xmax && xClip >= xmin) {

x = xClip;

y = ymin;

}

}

if (rc[2] == 1) {

yClip = y + (xmax - x) \* m;

if (yClip <= ymax && yClip >= ymin) {

y = yClip;

x = xmax;

}

}

if (rc[3] == 1) {

yClip = y + (xmin - x) \* m;

if (yClip <= ymax && yClip >= ymin) {

y = yClip;

x = xmin;

}

}

p->x = x;

p->y = y;

}

void clip() {

Point p1, p2;

p1.x = 50;

p1.y = 50;

p2.x = 350;

p2.y = 450;

int rc1[4], rc2[4];

float m = (p2.y - p1.y) / (p2.x - p1.x);

findRegionCode(p1, rc1);

findRegionCode(p2, rc2);

if (trivialAccept(rc1, rc2)) {

cout << "\nLine inside";

glColor3f(0, 1, 1);

glBegin(GL\_LINE);

glVertex2f(p1.x, p1.y);

glVertex2f(p2.x, p2.y);

glEnd();

}

else if (trivialReject(rc1, rc2)) {

cout << "\nLine outside";

}

else {

cout << "\nLine to be clipped";

CohenPoint(&p1,rc1,m);

CohenPoint(&p2, rc2,m);

glColor3f(1, 0.5, 0);

glBegin(GL\_LINES);

glVertex2f(p1.x, p1.y);

glVertex2f(p2.x, p2.y);

glEnd();

}

}

void lineClipping() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0f, 0.0f, 0.0f);

xmin = 200;

ymin = 200;

xmax = 400;

ymax = 400;

glRectf(xmin, ymin, xmax, ymax);

clip();

glFlush();

}

int main(int argc, char\* argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Line clipping");

glutDisplayFunc(lineClipping);

myinit();

glutMainLoop();

return 1;

}

Projections:

#include<iostream>

#include<gl/glut.h>

bool keyState[265];

int xangle, yangle, zangle;

void init() {

glClearColor(1, 1, 1, 0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

//glOrtho(-2, 2, -2, 2, 1, 100);

gluPerspective(60, 1, 1, 100);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glEnable(GL\_DEPTH\_TEST);

for (int i = 0; i < 256; ++i) {

keyState[i] = false;

}

}

void draw() {

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

glColor3f(1, 0, 0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(0, 0, 7, 0, 0, 0, 0, 1, 0);

glRotatef(xangle, 1, 0, 0);

glRotatef(yangle, 0, 1, 0);

glRotatef(zangle, 0, 0, 1);

glutWireCube(1);

glutSwapBuffers();

}

void keyOperations() {

if (keyState['w']) xangle += 45;

else if (keyState['s']) xangle -= 45;

else if (keyState['a']) yangle -= 45;

else if (keyState['d']) yangle += 45;

else if (keyState['q']) zangle += 45;

else if (keyState['e']) zangle -= 45;

xangle %= 360;

yangle %= 360;

zangle %= 360;

draw();

}

void keyPressed(unsigned char key, int x, int y) {

keyState[key] = true;

keyOperations();

}

void keyUp(unsigned char key, int x, int y) {

keyState[key] = false;

}

int main(int argc, char\* argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Ortho prjection");

glutDisplayFunc(draw);

init();

glutKeyboardFunc(keyPressed);

glutKeyboardUpFunc(keyUp);

glutMainLoop();

return 1;

}

3D animation:

#include<gl/glut.h>

#include<iostream>

void myinit() {

glClearColor(1, 1, 1, 0);

//glColor3f(0.0, 0.0, 0.0);

glShadeModel(GL\_SMOOTH);

GLfloat light\_diffuse[] = { 1,1,1,1 };

GLfloat light\_position[] = { 0,0,1,0 };

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60, 1, 1, 100);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glEnable(GL\_DEPTH\_TEST);

}

void drawCube(int state) {

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

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(0, 0, 7, 0, 0, 0, 0, 1, 0);

GLfloat cube\_color[] = { 1, 0.5, 0.0, 0.0 };

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, cube\_color);

glRotatef(state, 1, 1, 0);

glutSolidCube(1);

glutSwapBuffers();

glutTimerFunc(1000 / 60, drawCube, (state+1)%360);

}

void cubeDemo() {

glutTimerFunc(1000 / 60, drawCube, 0);

}

int main(int argc, char\* argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("3d animation");

glutDisplayFunc(cubeDemo);

myinit();

glutMainLoop();

return 1;

}