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

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

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

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

}

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 transforms:

#include<iostream>

#include<gl/glut.h>

using namespace std;

int n;

float points[3][10], new\_points[3][10];

void myinit() {

glClearColor(1, 1, 1, 0);

glColor3f(0, 0, 0);

glPointSize(2);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

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

}

void multiply(float a[3][3], float b[3][10], float c[3][10]) {

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

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

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

c[i][j] += a[i][k] \* b[k][j];

}

}

}

}

void display(float points[3][10]) {

glBegin(GL\_LINE\_LOOP);

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

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

}

glEnd();

}

void translate() {

glClear(GL\_COLOR\_BUFFER\_BIT);

float tx = 50, ty = 50;

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

multiply

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

cout << new\_points[0][i] << " " << new\_points[1][i] << "\n";

}

display(points);

glColor3f(1, 0, 0);

display(new\_points);

glFlush();

}

void scale() {

glClear(GL\_COLOR\_BUFFER\_BIT);

float sx = 2, sy = 2;

float xf, yf;

//scale on origin

//xf = yf = 0;

//scale on fixed point

xf = 200;

yf = 100;

float T[3][3] = { {sx,0,xf\*(1-sx)}, {0,sy,yf\*(1-sy)}, {0,0,1} };

multiply(T, points, new\_points);

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

cout << new\_points[0][i] << " " << new\_points[1][i] << "\n";

}

display(points);

glColor3f(1, 0, 0);

display(new\_points);

glFlush();

}

void rotate() {

glClear(GL\_COLOR\_BUFFER\_BIT);

float theta, xf, yf;

theta = 45;

//convert degree to radian

theta \*= 3.14 / 180;

//rotate wrt origin

//xf = yf = 0;

//rotate wrt fixed point

xf = 150;

yf = 50;

float T[3][3] = {

{cos(theta), -1\*sin(theta), xf\*(1-cos(theta)) + yf\*(sin(theta)) },

{sin(theta), cos(theta), yf \* (1 - cos(theta)) - xf \* (sin(theta))},

{0,0,1}

};

multiply(T, points, new\_points);

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

cout << new\_points[0][i] << " " << new\_points[1][i] << "\n";

}

display(points);

glColor3f(1, 0, 0);

display(new\_points);

glFlush();

}

void reflectX() {

glClear(GL\_COLOR\_BUFFER\_BIT);

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

multiply(T, points, new\_points);

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

cout << new\_points[0][i] << " " << new\_points[1][i] << "\n";

}

display(points);

glColor3f(1, 0, 0);

display(new\_points);

glFlush();

}

void reflectY() {

glClear(GL\_COLOR\_BUFFER\_BIT);

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

multiply(T, points, new\_points);

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

cout << new\_points[0][i] << " " << new\_points[1][i] << "\n";

}

display(points);

glColor3f(1, 0, 0);

display(new\_points);

glFlush();

}

void reflectOrigin() {

glClear(GL\_COLOR\_BUFFER\_BIT);

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

multiply(T, points, new\_points);

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

cout << new\_points[0][i] << " " << new\_points[1][i] << "\n";

}

display(points);

glColor3f(1, 0, 0);

display(new\_points);

glFlush();

}

void reflectXeqY() {

glClear(GL\_COLOR\_BUFFER\_BIT);

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

multiply(T, points, new\_points);

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

cout << new\_points[0][i] << " " << new\_points[1][i] << "\n";

}

display(points);

glColor3f(1, 0, 0);

display(new\_points);

glFlush();

}

void shearX() {

float sh;

sh = 1;

glClear(GL\_COLOR\_BUFFER\_BIT);

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

multiply(T, points, new\_points);

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

cout << new\_points[0][i] << " " << new\_points[1][i] << "\n";

}

display(points);

glColor3f(1, 0, 0);

display(new\_points);

glFlush();

}

void shearY() {

float sh;

sh = 1;

glClear(GL\_COLOR\_BUFFER\_BIT);

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

multiply(T, points, new\_points);

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

cout << new\_points[0][i] << " " << new\_points[1][i] << "\n";

}

display(points);

glColor3f(1, 0, 0);

display(new\_points);

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("2D transforms");

n = 4;

points[0][0] = 50;

points[1][0] = 50;

points[2][0] = 1;

points[0][1] = 150;

points[1][1] = 50;

points[2][1] = 1;

points[0][2] = 150;

points[1][2] = 150;

points[2][2] = 1;

points[0][3] = 50;

points[1][3] = 150;

points[2][3] = 1;

//glutDisplayFunc(translate);

//glutDisplayFunc(scale);

//glutDisplayFunc(rotate);

//glutDisplayFunc(reflectY);

//glutDisplayFunc(reflectOrigin);

//glutDisplayFunc(reflectXeqY);

//glutDisplayFunc(shearX);

glutDisplayFunc(shearY);

myinit();

glutMainLoop();

return 1;

}

2D composite:

#include<iostream>

#include<gl/glut.h>

using namespace std;

int n;

float points[3][10], new\_points[3][10];

void myinit() {

glClearColor(1, 1, 1, 0);

glColor3f(0, 0, 0);

glPointSize(2);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

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

}

void multiply(float a[3][3], float b[3][10], float c[3][10]) {

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

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

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

c[i][j] += a[i][k] \* b[k][j];

}

}

}

}

void mulT(float a[3][3], float b[3][3], float c[3][3]) {

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

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

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

c[i][j] += a[i][k] \* b[k][j];

}

}

}

}

void display(float points[3][10]) {

glBegin(GL\_LINE\_LOOP);

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

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

}

glEnd();

}

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

float tx = 50, ty = 50;

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 scale(float T[3][3]) {

glClear(GL\_COLOR\_BUFFER\_BIT);

float sx = 2, sy = 2;

float xf, yf;

//scale on origin

//xf = yf = 0;

//scale on fixed point

xf = 200;

yf = 100;

float temp[3][3] = { {sx,0,xf \* (1 - sx)}, {0,sy,yf \* (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]) {

glClear(GL\_COLOR\_BUFFER\_BIT);

float theta, xf, yf;

theta = 45;

//convert degree to radian

theta \*= 3.14 / 180;

//rotate wrt origin

//xf = yf = 0;

//rotate wrt fixed point

xf = 150;

yf = 50;

float temp[3][3] = {

{cos(theta), -1 \* sin(theta), xf \* (1 - cos(theta)) + yf \* (sin(theta)) },

{sin(theta), cos(theta), yf \* (1 - cos(theta)) - xf \* (sin(theta))},

{0,0,1}

};

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

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

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

}

}

}

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

glClear(GL\_COLOR\_BUFFER\_BIT);

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]) {

glClear(GL\_COLOR\_BUFFER\_BIT);

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]) {

glClear(GL\_COLOR\_BUFFER\_BIT);

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 reflectXeqY(float T[3][3]) {

glClear(GL\_COLOR\_BUFFER\_BIT);

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;

sh = 1;

glClear(GL\_COLOR\_BUFFER\_BIT);

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;

sh = 1;

glClear(GL\_COLOR\_BUFFER\_BIT);

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 composite() {

glClear(GL\_COLOR\_BUFFER\_BIT);

float T1[3][3], T2[3][3], T[3][3] = { { 0,0,0 } ,{ 0,0,0 } ,{ 0,0,0 } };

rotate(T1);

reflectY(T2);

mulT(T1, T2,T);

multiply(T, points, new\_points);

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

cout << new\_points[0][i] << " " << new\_points[1][i] << "\n";

}

display(points);

glColor3f(1, 0, 0);

display(new\_points);

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("2D transforms");

n = 4;

points[0][0] = 50;

points[1][0] = 50;

points[2][0] = 1;

points[0][1] = 150;

points[1][1] = 50;

points[2][1] = 1;

points[0][2] = 150;

points[1][2] = 150;

points[2][2] = 1;

points[0][3] = 50;

points[1][3] = 150;

points[2][3] = 1;

glutDisplayFunc(composite);

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;

}

3D transforms:

#include<GL/glut.h>

#include<bits/stdc++.h>

using namespace std;

#define PI 3.14159265358979323846264338327950288419716939937510582

typedef struct Point{

double x, y, z, h;

}Point;

typedef struct Face{

Point v[4];

}Face;

typedef struct Cuboid{

Point v[8];

Face faces[6];

}Cuboid;

int assignList[6][4] = {{0, 1, 3, 2}, {0, 4, 5, 1}, {0, 4, 6, 2},

{4, 5, 7, 6}, {2, 6, 7, 3}, {1, 5, 7, 3}

};

float colors[6][3] = {{0, 1, 0}, {1, 0, 0}, {0, 0, 1},

{1, 1, 0}, {0, 1, 1}, {1, 0, 1}

};

double tMat[4][4];

bool tflag = false;

Cuboid cuboid, tcuboid;

Cuboid initCuboid(){

Cuboid cuboid;

double po[8][3] = {{-25, 25, 0}, {25, 25, 0}, {-25, -25, 0}, {25,

-25, 0},

{-25, 25, 50}, {25, 25, 50}, {-25, -25, 50}, {25,

-25, 50}

};

for(int i=0;i<8;i++){

cuboid.v[i].x = po[i][0];

cuboid.v[i].y = po[i][1];

cuboid.v[i].z = po[i][2];

cuboid.v[i].h = 1;

}

for(int i=0;i<6;i++){

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

cuboid.faces[i].v[j] = cuboid.v[assignList[i][j]];

}

}

return cuboid;

}

void myInit() {

glClearColor(1.0,1.0,1.0,0.0);

glColor3f(0, 0, 0);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

glLoadIdentity();

glOrtho(-200, 200, -200, 200, -200, 200);

glEnable(GL\_DEPTH\_TEST);

}

void disp() {

glRotatef(30, 1, 0, 0);

glRotatef(30, 0, 1, 0);

}

void transformCuboid() {

tflag=true;

for(int i=0;i<8;i++){

cout<<cuboid.v[i].x<<" "<<cuboid.v[i].y<<"

"<<cuboid.v[i].z<<"\n";

}

cout<<"\n\n";

for(int p=0;p<8;p++){

double pnt[4][1], pnt1[4][1];

pnt[0][0] = cuboid.v[p].x;

pnt[1][0] = cuboid.v[p].y;

pnt[2][0] = cuboid.v[p].z;

pnt[3][0] = cuboid.v[p].h;

memset(pnt1, 0, sizeof(pnt1));

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

for(int j=0;j<1;j++){

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

pnt1[i][j] += tMat[i][k]\*pnt[k][j];

}

}

}

tcuboid.v[p].x = pnt1[0][0];

tcuboid.v[p].y = pnt1[1][0];

tcuboid.v[p].z = pnt1[2][0];

tcuboid.v[p].h = pnt1[3][0];

}

for(int i=0;i<6;i++){

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

tcuboid.faces[i].v[j] =

tcuboid.v[assignList[i][j]];

}

}

for(int i=0;i<8;i++){

cout<<tcuboid.v[i].x<<" "<<tcuboid.v[i].y<<"

"<<tcuboid.v[i].z<<"\n";

}

glutPostRedisplay();

}

void getTransformMatrix() {

memset(tMat, 0, sizeof(tMat));

tMat[0][0] = tMat[1][1] = tMat[2][2] = tMat[3][3] = 1;

int ch;

cout<<"Menu:\n\t1.Translation\n\t2.Rotation\n\t3.Scaling\n\tChoice:

";

cin>>ch;

switch(ch){

case 1:

cout<<"Enter translation parameters: ";

cin>>tMat[0][3]>>tMat[1][3]>>tMat[2][3];

break;

case 2:

cout<<"Enter degree of rotation: ";

double deg;

cin>>deg;

deg = deg\*PI/180;

tMat[0][0] = cos(deg);

tMat[0][1] = -sin(deg);

tMat[1][1] = tMat[0][0];

tMat[1][0] = -tMat[0][1];

break;

case 3:

cout<<"Enter scaling parameters: ";

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

break;

default: cout<<"Bruh\n";

}

transformCuboid();

}

void displayCuboid(Cuboid cuboid, double alpha = 0.6){

for(int i=0;i<6;i++){

glColor4f(colors[i][0],colors[i][1],colors[i][2], alpha);

glBegin(GL\_POLYGON);

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

glVertex3d(cuboid.faces[i].v[j].x,cuboid.faces[i].v[j].y,cuboid.faces[i].v

[j].z);

}

glEnd();

}

}

void myDisplay(){

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

glColor4f(0, 0, 0, 1);

glBegin(GL\_LINES);

glVertex3d(300, 0, 0);

glVertex3d(-300, 0, 0);

glEnd();

glBegin(GL\_LINES);

glVertex3d(0, 300, 0);

glVertex3d(0, -300, 0);

glEnd();

glBegin(GL\_LINES);

glVertex3d(0, 0, 300);

glVertex3d(0, 0, -300);

glEnd();

displayCuboid(cuboid);

if(tflag)displayCuboid(tcuboid, 1);

glFlush();

getTransformMatrix();

}

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

cuboid = initCuboid();

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(960, 960);

glutInitWindowPosition(0, 0);

glutCreateWindow("3D transformations");

myInit();

disp();

glutDisplayFunc(myDisplay);

glutMainLoop();

}

Projections:

#include<gl/glut.h>

#include<stdio.h>

bool\* keyStates = new bool[256];

int x\_angle = 0, y\_angle = 0, z\_angle = 0;

void drawObj() {

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

glColor3f(0.0, 0.0, 0.0);

// X, Y and Z axis

glBegin(GL\_LINES);

glVertex3d(-5, 0, 0);

glVertex3d(5, 0, 0);

glVertex3d(0, -5, 0);

glVertex3d(0, 5, 0);

glVertex3d(0, 0, 1);

glVertex3d(0, 0, 100);

glEnd();

glColor3f(1.0, 0.0, 0.0);

glLoadIdentity();

glTranslatef(0.0f, 0.0f, -5.0f);

glPushMatrix();

glRotatef(x\_angle, 1, 0, 0);

glRotatef(y\_angle, 0, 1, 0);

glRotatef(z\_angle, 0, 0, 1);

glutWireCube(1);

glPopMatrix();

glFlush();

}

void myinit() {

int WIDTH = 500, HEIGHT = 500, choice;

choice = 1; //parallel

choice = 2; //persp

glClearColor(1, 1, 1, 0);

glViewport(0, 0, WIDTH, HEIGHT);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (choice == 1) {

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

}

else {

gluPerspective(60, (GLfloat)WIDTH / (GLfloat)HEIGHT, 1, 100);

}

glMatrixMode(GL\_MODELVIEW);

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

keyStates[i] = false;

}

glEnable(GL\_DEPTH\_TEST);

}

void keyOperations(void) {

int ANGLE\_INC = 45;

if (keyStates['w']) { x\_angle += ANGLE\_INC; }

else if (keyStates['s']) { x\_angle -= ANGLE\_INC; }

else if (keyStates['a']) { y\_angle -= ANGLE\_INC; }

else if (keyStates['d']) { y\_angle += ANGLE\_INC; }

else if (keyStates[' ']) { z\_angle += ANGLE\_INC; }

x\_angle %= 360;

y\_angle %= 360;

z\_angle %= 360;

drawObj();

}

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

keyStates[key] = true;

keyOperations();

}

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

keyStates[key] = false;

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Projections");

glutDisplayFunc(drawObj);

myinit();

glutKeyboardFunc(keyPressed);

glutKeyboardUpFunc(keyUp);

glutMainLoop();

return 1;

}

3D object:

#include <GL/glut.h>

#include <GL/glu.h>

#include <stdlib.h>

#include <stdio.h>

int INC = 1;

void initialize(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);

glShadeModel(GL\_SMOOTH);

GLfloat light\_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };

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);

glEnable(GL\_DEPTH\_TEST);

}

// Helper function to load texture from a .bmp file

GLuint LoadTexture(const char\* filename)

{

GLuint texture;

int width, height;

unsigned char\* data;

FILE\* file;

file = fopen(filename, "rb");

if (file == NULL) return 0;

width = 474;

height = 395;

data = (unsigned char\*)malloc(width \* height \* 3);

//int size = fseek(file,);

fread(data, width \* height \* 3, 1, file);

fclose(file);

for (int i = 0; i < width \* height; ++i)

{

int index = i\*3;

unsigned char B, R;

B = data[index];

R = data[index + 2];

data[index] = R;

data[index + 2] = B;

}

glGenTextures(1, &texture);

glBindTexture(GL\_TEXTURE\_2D, texture);

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER,

GL\_LINEAR\_MIPMAP\_NEAREST);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

gluBuild2DMipmaps(GL\_TEXTURE\_2D, 3, width, height, GL\_RGB,

GL\_UNSIGNED\_BYTE, data);

free(data);

return texture;

}

void drawScene(int state)

{

if (state == 0)

INC = 1;

else if (state == 10)

INC = -1;

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

// "Silver" texture will be applied to the teapot alone

GLuint texture;

texture = LoadTexture("silver.bmp");

glBindTexture(GL\_TEXTURE\_2D, texture);

glLoadIdentity();

gluLookAt(0.0, 1.0, 7.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glMatrixMode(GL\_MODELVIEW);

// Cube

glPushMatrix();

GLfloat cube\_color[] = { 0.46, 0.26, 0.2, 1.0 };

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

glScalef(4, 1.5, 1.0);

glTranslatef(0.4, -1.0, 0.0);

glutSolidCube(1.0);

glPopMatrix();

// Teapot

glPushMatrix();

glEnable(GL\_TEXTURE\_2D);

GLfloat teapot\_color[] = { 0.7, 0.7, 0.7, 0.0 };

GLfloat mat\_shininess[] = { 100 };

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

glMaterialfv(GL\_FRONT, GL\_SHININESS, mat\_shininess);

glTranslatef(1.7, -0.2, 0.0);

glutSolidTeapot(0.7);

glDisable(GL\_TEXTURE\_2D);

glPopMatrix();

// Ramp

glPushMatrix();

GLfloat ramp\_color[] = { 0.6, 0.44, 0.39, 1.0 };

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

glRotatef(45, 0, 0, 1);

glTranslatef(-1.2, -0.2, 0);

glScalef(3.4, 0.2, 1.0);

glutSolidCube(1.0);

glPopMatrix();

// Sphere

glPushMatrix();

GLfloat ball\_color[] = { 0.59, 0.1, 0.55, 1.0 };

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

glRotatef(-0.1 \* state, 0, 0, 1);

glTranslatef(-2.5 - 0.25\*state, -2, 0);

glutSolidSphere(0.5, 10, 10);

glPopMatrix();

glutSwapBuffers();

glutTimerFunc(1000 / 60, drawScene, state + INC);

}

void reshape(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(75, 1, 1, 20);

glMatrixMode(GL\_MODELVIEW);

}

void sceneDemo() {

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

glutTimerFunc(1000 / 60, drawScene, 0);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("3D Scene");

initialize();

glutDisplayFunc(sceneDemo);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}