1

**EX:NO:1 (a)**

**AIM:**

**BRESENHAMS ALGORITHM FOR LINE**

2

To Implement the Bresenhams Algorithm for Line.

**ALGORITHM:**

**Step 1:** Input the two line Endpoints and the left endpoint in (x0, y0).

**Step 2:** Load (x0, y0) into the frame buffer that is plot the first point.

**Step 3:** Calculate constants ∆x, ∆y, 2∆y and 2∆y-2∆x, and obtain the starting

value for the decision parameter as P0=2∆y-∆x.

**Step 4:** At each xk along the line, Starting at k=0, perform the following test:

If Pk<0, the next point to plot is (xk+1,yk) and Pk+1=Pk+2∆y.

Otherwise the next point to plot is (xk+1,yk+1) and Pk+1=Pk+2∆y-2∆x.

**Step 5:** Repeat the above step ∆x times.

**PROGRAM:**

#include<stdio.h>

#include<math.h>

#include<graphics.h>

#include<conio.h>

#include<stdlib.h>

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

void main()

{

int x1,y1,x2,y2;

int driver = DETECT,gmode;

initgraph(&driver,&gmode," ");

printf("ENTER THE STARTING CO-ORDINATE\n");

scanf("%d%d",&x1,&y1);

printf("ENTER THE ENDING CO-ORDINATE\n");

scanf("%d%d",&x2,&y2);

bshm(x1,y1,x2,y2);

getch();

}

void bshm(int x1,int y1,int x2,int y2)

{

int x = x1,y = y1,dx,dy,s1,s2;

int length,dp,i,temp,swap = 0;

putpixel(x1,y1,RED);

dx = abs(x2 - x1);

2

dy = abs(y2 - y1);

if(x2 < x1)

s1 = -1;

else if(x2 > x1)

s1 = 1;

else

s1 = 0;

if(y2 < y1)

s2 = -1;

else if(y2 > y1)

s2 = 1;

else

s2 = 0;

dp = 2 \* dy - dx;

if(dy < dx)

{

temp = dx;

dx = dy;

dy = temp;

swap = 1;

}

for(i=1;i<=dx;i++)

{

if(dp < 0)

{

if(swap)

putpixel(x,y = -y+s2,RED);

else

putpixel(x = x+s1,y = y+s2,RED);

dp += 2 \* dy;

}

else

{

putpixel(x = x+s1,y = y+s2,YELLOW);

dp = dp+ 2\*dy - 2\*dx;

}

}

**OUTPUT:**

**IMPLEMENTATION OF BRESENHAM ALGORITHM FOR LINE**

Enter the starting co-ordinates : 100 100

Enter the ending co-ordinates : 200 200

3

3

**EX:NO:1 (b)**

**AIM:**

**BRESENHAMS ALGORITHM FOR ELLIPSE**

4

To Implement the Bresenhams Algorithm for Ellipse.

**ALGORITHM:**

**Step 1:**Input rx, ry and ellipse center (xc, yc) and obtain the first point on an ellipse

centered on the origin as

(x0, y0)= (0, ry).

**Step 2:** Calculate the initial value of the decision parameter in region 1 as

2-rx2ry+1/4 rx2.

P10=ry

**Step 3:** At each xk position in Region 1, starting at k=0, perform the following test,

If p1k<0, the next point along the ellipse centered on (0, 0) is

(xk+1, yk) and p1k+1=p1k+2ry2xk+1+ry2 .

Otherwise, the next point along the circle is (xk+1, yk-1) and

p1k+1=p1k+2ry2xk+1-2rx2yk+1+ry2 with

2xk+1=2ry2xk+2ry2

2ry

2rx2yk+1=2rx2yk-2rx2, and continue the Initial value of the decision

parameter in region 2 using the last point (x0,y0) calculated in region 1 as

p20=ry2(x0+1/2)2+rx2(y0-1)2-rx2ry2.

**Step 4:** At each yk position in region 2, starting at k=0, perform the following test,

If p2k>0, the next point along the ellipse centered on (0, 0) is

(xk ,yk-1) and p2k+1=p2k-2rx2yk+1+rx2.

Otherwise, the next point along the circle is (xk+1,yk-1) and

p2k+1=p2k+2ry2xk+1-2ry2yk+1+rx2 using the same incremental calculations for

x and y as in region 1.

**Step 5:** Determine symmetry points in the other three quadrants.

**Step 6:** Move each calculated pixel position (x, y) on to the elliptical path centered

on (xc, yc) and plot the co-ordinate values:

x=c+xc

y=y+yc

**Step 7:** Repeat the steps for region 1 until 2ry2x>2rx2y.

**PROGRAM:**

#include<graphics.h>

#include<stdio.h>

#include<math.h>

#include<conio.h>

void ellipsemidpoint(int xc,int yc,int rx,int ry)

{

void ellipseplotpoints(int,int,int,int);

float rx2 = rx \* rx;

float ry2 = ry \* ry;

float tworx2 = 2 \* rx2;

float twory2 = 2 \* ry2;

4

float p;

float x = 0;

float y = ry;

float px = 0;

float py = tworx2 \* y;

ellipseplotpoints(xc,yc,x,y);

p = ry2 - rx2 \* ry + (0.25 \* rx2);

while(px < py)

{

x++;

px += twory2;

if(p >= 0)

{

y = y-1;

py = py - tworx2;

}

if(p < 0)

p += ry2 + px;

else

p += ry2 + px - py;

ellipseplotpoints(xc,yc,x,y);

}

p = ry2 \* (x + 0.5) \* (x + 0.5) + rx2 \* (y-1)\*(y-1) - rx2 \* ry2;

while(y > 0)

{

y--;

py -= tworx2;

if(p <= 0)

{

x++;

px += twory2;

}

if(p > 0)

p = p+rx2-py;

else

p = p+rx2-py+px;

ellipseplotpoints(xc,yc,x,y);

}

}

void ellipseplotpoints(int xc,int yc,int x,int y)

{

putpixel(xc+x,yc+y,RED);

putpixel(xc-x,yc+y,GREEN);

putpixel(xc+x,yc-y,BLUE);

putpixel(xc-x,yc-y,YELLOW);

}

5

5

void main()

{

int xc,yc,rx,ry;

int gdriver = DETECT,gmode;

initgraph(&gdriver,&gmode," ");

printf("\nENTER THE CENTER COORDINATES:\n");

scanf("%d%d",&xc,&yc);

printf("\nENTER THE MAJOR AND MINOR AXES:\n");

scanf("%d%d",&rx,&ry);

ellipsemidpoint(xc,yc,rx,ry);

getch();

}

**OUTPUT:**

**IMPLEMENTATION OF BRESENHAM ALGORITHM FOR ELLIPSE**

Enter the Center Co-ordinates: 200 200

Enter the Major and Minor axes : 80 50

6

6

**EX:NO:1 (c)**

**AIM:**

**BRESENHAMS ALGORITHM FOR CIRCLE**

7

To Implement the Bresenhams Algorithm for Circle.

**ALGORITHM:**

**Step 1:** Input the radius r and circle center(xc, yc) and obtain the first point on the

circumference of a circle centered on the origin as, (x0,y0)=(o, r).

**Step 2:** Calculate the Initial value of the decision parameter as,

P0=5/4-r.

**Step 3:** At each xk position, starting at k=0, perform the following test,

If pk<0, the next point along the circle centered on (0, 0) is (xk+1, yk) and

pk+1=pk+2xk+1+1.

Otherwise the next point along the circle is (xk+1, yk-1) and

pk+1=pk+2xk+1-2yk+1.

Where,

2xk+1=2xk+2

2yk+1=2yk-2.

**Step 4:** Determine symmetry points in the other seven octants.

**Step 5:** Move each calculated pixel position (x, y) onto the circular path centered on

(xc, yc) and plot the co-ordinate values:

x=x+xc

y=y+yc

**Step 6:** Repeat step 3 through 5 until x≥y.

**PROGRAM:**

#include<graphics.h>

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int xc,yc,r;

void symm(int x,int y)

{

putpixel(x+xc,y+yc,GREEN);

putpixel(-x+xc,-y+yc,GREEN);

putpixel(x+xc,-y+yc,GREEN);

putpixel(-x+xc,y+yc,GREEN);

putpixel(y+xc,x+yc,RED);

putpixel(-y+xc,-x+yc,RED);

putpixel(-y+xc,x+yc,RED);

putpixel(y+xc,-x+yc,RED);

}

void bhsmcir(int r)

{

7

int x=0,y=r,dp;

dp = 1-r;

while(x < y)

{

if(dp < 0)

{

dp += 2 \* x + 1;

symm(++x,y);

}

else

{

dp += 2 \* (x - y) + 1;

symm(++x,--y);

}

}

}

void main()

{

int gdriver = DETECT,gmode;

clrscr();

initgraph(&gdriver,&gmode," ");

printf("ENTER THE CENTER COORDINATES: \n");

scanf("%d%d",&xc,&yc);

printf("ENTER RADIUS:\n");

scanf("%d",&r);

bhsmcir(r);

getch();

}

**OUTPUT:**

**IMPLEMENTATION OF BRESENHAM ALGORITHM FOR CIRCLE**

**DRAWING.**

Enter the centre co-ordinates : 250 250

Enter the radius : 100

8

8

**EX: NO: 2 PERFORMING 2D TRANSFORMATIONS**

**AIM:**

To write a C program to perform 2D Transformations such as translation, scaling

and rotation.

**ALGORITHM:**

**Step 1:** Initialize the graphics system.

**Step 2:** Input the number to vertices and their corresponding co-ordinates.

**Step 3:** Draw the polygon using the vertices and input type of transformation on the

Polygon.

**Step 4: For Translation:**

o Input the translating co-ordinates tx and ty.

o Call translate () which calculates,

x1=x1+tx

y1=y1+ty

x2=x2+tx

y2=y2+ty

o Draw the polygon using the new co-ordinates.

**Step 5: For Scaling:**

 Input the scaling factors sx,sy and the reference co-ordinates xf,yf.

 Call scale() which calculates

x'=x. sx+x’(1-sx)

y’=y. sy+y’(1-sy)

 Draw the polygon using the new co-ordinates.

**Step 6: For Rotation:**

 Input the Rotating Angle and the reference co-ordinates xr,yr.

 Call rotate() which calculates

x'=xr+(x-xr)cosθ-(y-yr)sinθ

y’=yr+(x-xr)sinθ-(y-yr)cosθ

 Draw the polygon using the new co-ordinates.

**Step 7:** Repeat the steps until the user needs to exit.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

#define PI 3.14

float cord[10][3];

int n;

void myclrscr()

{

9

9

int gdriver = DETECT,gmode;

initgraph(&gdriver,&gmode,"");

}

void getpoints()

{

int i;

printf("Enter the no of defining vertices:");

scanf("%d",&n);

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

{

printf("Enter co-ordinates %d:",i+1);

scanf("%f%f",&cord[i][0],&cord[i][1]);

cord[i][2] = 1;

}

}

void drawpolygon()

{

int i;

for(i=0;i<n-1;i++)

line(cord[i][0],cord[i][1],cord[i+1][0],cord[i+1][1]);

line(cord[0][0],cord[0][1],cord[i][0],cord[i][1]);

}

void mul(float mat1[][3],float m1,int n,float mat2[][3],int m2,float ans[][3])

{

int i,j,k;

float sum;

for(i=0;i<m1;i++)

for(j=0;j<m2;j++)

{

sum = 0;

for(k=0;k<n;k++)

sum = sum+mat1[i][k]\*mat2[k][j];

ans[i][j] = sum;

}

}

void translate(int tx,int ty,int draw)

{

int i;

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

tmat[2][0] = tx;

tmat[2][1] = ty;

mul(cord,n,3,tmat,3,cord);

if(draw)

drawpolygon();

}

void scale(float sx,float sy,int xf,int yf)

10

10

{

int i;

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

tmat[0][0]=sx;

tmat[1][1]=sy;

tmat[2][0]=xf-sx\*xf;

tmat[2][1]=yf-sy\*yf;

drawpolygon();

}

void rotate(float angle,int xf,int yf)

{

int i;

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

angle \*=PI/180;

translate(-xf,-yf,0);

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

tMat[0][1]=sin(angle);

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

tMat[1][1]=cos(angle);

mul(cord,n,3,tMat,3,cord);

translate(xf,yf,0);

drawpolygon();

}

void main()

{

int choice;

float xf,yf,sx,sy,tx,ty;

float r;

myclrscr();

getpoints();

drawpolygon();

getch();

myclrscr();

while(1)

{

printf(“1. TRANSLATION\n”);

printf(“2. Scaling\n”);

printf(“3. Rotation\n”);

printf(“4. Exit\n”);

scanf(“ENTER UR CHOICE:\n”);

switch(choice)

{

case 1:

printf(“ Enter the Translating Co-Ordinates:\n”);

scanf(“%f%f”,&tx,&ty);

11

11

myclrscr();

translate(tx,ty,1);

break;

case 2:

printf(“ Enter the Scaling Factors:\n”);

scanf(“%f%f”,&sx,&sy);

printf(“ Enter the Reference Co-Ordinates:\n);

scanf(“%f%f”,&xf,&yf);

myclrscr();

scale(sx,sy,xf,yf);

break;

case 3:

printf(“ Enter the Rotating Angle:\n”);

scanf(“%f”,&r);

printf(“Enter the Reference Co-Ordinates:\n”);

scanf(“%f%f”,&xf,&yf);

myclrscr();

rotate(r,xy,yf);

break;

case 4:

myclrscr();

exit(0);

getch();

}

}

}

**OUTPUT:**

**2D TRANSFORMATION**

Enter the no of defining vertices : 3

Enter coordinate 1: 200 200

Enter coordinate 2: 100 300

Enter coordinate 3: 300 300

12

12

1- Translation

2- Scaling

3- Rotation

0- Exit

Enter your choice : 1

Enter the translating co-ordinates : 40 60

Enter your choice : 2

Enter the Scaling factors : 1.2 1.2

Enter the refrence co-ordinates : 20 50

Enter your choice : 3

Enter the Rotating angle : 15

Enter the refrence co-ordinates : 20 30

13

13

14

**EX: NO: 3a IMPLEMENTATION OF COHEN-SUTHERLAND 2D CLIPPING**

**AIM:**

To write a C program to implement Cohen-Sutherland 2D Clipping.

**ALGORITHM:**

**Step 1:** Input the endpoints p1 and p2.

**Step 2:** Input the window co-ordinates and draw the rectangular window on the screen.

**Step 3:** Check the bottom endpoint of the line p1 against the left, right and bottom

boundaries of the window and find whether it is below the clip window.

**Step 4:** If p1 is below the clip window, find the point of intersection of the line with the

bottom boundary of the window as p1’.

**Step 5:** Discard the line section from p1 tp p1’.

**Step 6:** Check whether the point p2 is above the clip window, (ie) outside the clip

window.

**Step 7:** If p2 is above the clip window andto the left/right of the window, find the point

of intersection of the line with the left/right boundary of the window as p2’.

**Step 8:** Find the final point of intersection of the line with the top boundary of the clip

window as p2’’.

**Step 9:** Discard the line sections from p2 to p2’ and p2’ to p2’’.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

typedef unsigned int outcode;

enum{TOP = 0x1,BOTTOM = 0x2,RIGHT = 0x4,LEFT = 0x8};

void lineclip(x0,y0,x1,y1,xwmin,ywmin,xwmax,ywmax)

float x0,y0,x1,y1,xwmin,ywmin,xwmax,ywmax;

{

int gd,gm;

outcode code0,code1,codeout;

int accept = 0,done = 0;

code0 = calcode(x0,y0,xwmin,ywmin,xwmax,ywmax);

code1 = calcode(x1,y1,xwmin,ywmin,xwmax,ywmax);

do

{

if(!(code0 | code1))

{

accept = 1;

done = 1;

}

14

else if(code0 & code1)

done = 1;

else

{

float x,y;

codeout = code0 ? code0 : code1;

if(codeout & TOP)

{

x = x0 + (x1 - x0) \* (ywmax - y0)/(y1 - y0);

y = ywmax;

}

else if(codeout & BOTTOM)

{

x = x0 + (x1 - x0) \* (ywmin - y0)/(y1 - y0);

y = ywmin;

}

else if(codeout & RIGHT)

{

y = y0 + (y1 - y0) \* (xwmax - x0)/(x1 - x0);

x = xwmax;

}

else

{

y = y0 + (y1 - y0) \* (xwmin - x0)/(x1 - x0);

x = xwmin;

}

if(codeout == code0)

{

x0 = x;y0 = y;

code0 = calcode(x0,y0,xwmin,ywmin,xwmax,ywmax);

}

else

{

x1 = x; y1 = y;

code0 = calcode(x1,y1,xwmin,ywmin,xwmax,ywmax);

}

}

}while(done == 0);

if(accept)

line(x0,y0,x1,y1);

rectangle(xwmin,ywmin,xwmax,ywmax);

getch();

}

int calcode(x,y,xwmin,ywmin,xwmax,ywmax)

float x,y,xwmin,ywmin,xwmax,ywmax;

15

15

{

int code = 0;

if(y > ywmax)

code |= TOP;

else if(y < ywmin)

code |= BOTTOM;

else if(x > xwmax)

code |= RIGHT;

else if(x < xwmin)

code |= LEFT;

return(code);

}

void main()

{

float x2,y2,x1,y1,xwmin,ywmin,xwmax,ywmax;

int gd,gm;

clrscr();

detectgraph(&gd,&gm);

initgraph(&gd,&gm," ");

printf("ENTER THE COORDINATES OF THE LINE:\n");

scanf("%f%f",&x1,&y1);

scanf("%f%f",&x2,&y2);

printf("ENTER THE COORDINATES OF THE WINDOW:\n");

scanf("%f%f",&xwmin,&ywmin);

scanf("%f%f",&xwmax,&ywmax);

printf("\n\nBEFORE CLIPPING:\n");

line(x1,y1,x2,y2);

rectangle(xwmin,ywmin,xwmax,ywmax);

getch();

cleardevice();

printf("\n\nAFTER CLIPPING:\n");

lineclip(x1,y1,x2,y2,xwmin,ywmin,xwmax,ywmax);

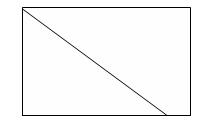
getch();

closegraph();

}

16

16



**OUTPUT:**

**2D LINE CLIPPING**

Enter the co-ordinates of Line:

X1 Y1 : 280 300

X2 Y2 : 400 420

Enter the co-ordinates of window:

xwmin , ywmin :300 320

xwmax , ywmax : 380 390

**BEFORE CLIPPING**

**AFTER CLIPPING**

17

17

18

**EX: NO: 3b IMPLEMENTATION OF WINDOW-TO-VIEWPORT MAPPING**

**AIM:**

To write a C program to implement Window to View port Mapping.

**ALGORITHM:**

**Step 1:** Initialize the graphics system.

**Step 2:** Assign the co-ordinates of the window to xwmin,xwmax,ywmin,ywmax.

**Step 3:** Assign the co-ordinates of the view port to xvmin,xvmax,yvmin,yvmax.

**Step 4:** Draw the window using the window co-ordinates.

**Step 5:** Draw the view port using the viewport co-ordinates.

**Step 6:** Choose the co-ordinates of a rectangle to be drawn inside the window and to be

mapped onto the viewport.

**Step 7:** Draw the rectangle inside the drawn window using its co-ordinates.

**Step 8:** Calculate the scaling factors sx,sy as,

Sx= (xvmax-xvmin)/(xwmax-xwmin)

Sy=(yvmax-yvmin)/(ywmax-ywmin)

**Step 9:** To map a point at position (xw,yw) in the window into position (xv,yv) in the

associated viewport, maintain the same relative placement in the viewport as in

the window using the expressions

xv= xvmin+(xw-xwmin).sx

yv= yvmin+(yw-ywmin).sy

**Step 10:** Obtain the co-ordinates of the rectangle which is to be mapped in the window

using the above two expressions.

**Step 11:** Draw the relative rectangle in the viewport using the mapped co-ordinates.

**Step 12:** Close the graphics system.

**PROGRAM:**

#include<graphics.h>

#include<stdio.h>

#include<conio.h>

#include<math.h>

void wind(int,int,int,int);

void draw(int,int,int,int,int,int,int,int);

void main()

{

int xwmin,xwmax,ywmin,ywmax,xvmin,xvmax,yvmin,yvmax;

int x1,x2,x3,x4,y1,y2,y3,y4;

double sx,sy;

int device,mode,a[8];

xwmin = 100; xvmin = 350;

ywmin = 200; yvmin = 225;

xwmax = 200; ywmax = 300;

18

xvmax = 400; yvmax = 275;

x1 = 140; x2 = 180; x3 = 180; x4 = 140;

y1 = 240; y2 = 240; y3 = 280; y4 = 280;

sx = float(xvmax - xvmin)/float(xwmax - xwmin);

sy = float(yvmax - yvmin)/float(ywmax - ywmin);

device = DETECT;

initgraph(&device,&mode," ");

settextstyle(3,0,2);

outtextxy(100,50,"WINDOW TO VIEWPORT TRANSFORMATION");

settextstyle(3,0,2);

outtextxy(100,125,"WINDOW");

settextstyle(2,0,3);

outtextxy(100,150,"=========");

settextstyle(3,0,2);

outtextxy(325,125,"VIEWPORT");

settextstyle(2,0,3);

outtextxy(325,150,"==========");

setlinestyle(DOTTED\_LINE,1,1);

draw(x1,y1,x2,y2,x3,y3,x4,y4);

int vx1 = sx \* (x1 - xwmin) + xvmin;

int vx2 = sx \* (x2 - xwmin) + xvmin;

int vx3 = sx \* (x3 - xwmin) + xvmin;

int vx4 = sx \* (x4 - xwmin) + xvmin;

int vy1 = sy \* (y1 - ywmin) + yvmin;

int vy2 = sy \* (y2 - ywmin) + yvmin;

int vy3 = sy \* (y3 - ywmin) + yvmin;

int vy4 = sy \* (y4 - ywmin) + yvmin;

draw(vx1,vy1,vx2,vy2,vx3,vy3,vx4,vy4);

getch();

closegraph();

}

void wind(int xwmin,int xwmax,int ywmin,int ywmax)

{

line(xwmin,ywmax,xwmax,ywmax);

line(xwmin,ywmin,xwmax,ywmin);

line(xwmin,ywmin,xwmin,ywmax);

line(xwmax,ywmin,xwmax,ywmax);

}

void draw(int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4)

{

line(x1,y1,x2,y2);

line(x1,y1,x4,y4);

line(x3,y3,x2,y2);

line(x3,y3,x4,y4);

}

19

19



**OUTPUT:**

**WINDOW TO VIEWPORT TRANSFORMATIONS**

WINDOW VIEWPORT

20

20

**EX: NO: 4 PERFORMING 3D TRANSFORMATIONS**

**AIM:**

21

To write a C++ program to perform 3D Transformations such as translation, scaling,

and rotation.

**ALGORITHM:**

**Step 1:** Initialize the graphics system.

**Step 2:** Input the number of vertices and their corresponding co-ordinates.

**Step 3:** Draw the polygon ie the 3D object and get the type of transformation to be

performed.

**Step 4:** For Scaling,

 Input the scaling factors along each axis sx,sy,sz.

 Calculate the new co-ordinates x1=sx.x, y1=y.sy, z1=z.sz

 Draw the polygon using the new co-ordinates.

**Step 5:** For Translation,

 Input the translation factors tx,ty,tz.

 Calculate the new co-ordinates x1=x.tx,y1=y.ty,z1=z.tz

 Draw the polygon 3D object using the new co-ordinates.

**Step 6:** For Rotation,

Rotation about X-Axis

 Input the rotating angle.

 Calculate the co-ordinates

y’= ycosθ-zsinθ

z’= ysinθ+zcosθ and x’=x.

Rotation about Y-Axis

 Input the rotating angle.

 Calculate the co-ordinates

z’= zcosθ-xsinθ

x’= zsinθ+xcosθ and y’=y.

Rotation about Z-Axis

 Input the rotating angle.

 Calculate the co-ordinates

x’= xcosθ-ysinθ

y’= xsinθ+ycosθ and z’=z.

**Step 7:** Exit the execution by drawing the 3D object.

**PROGRAM:**

**3D TRANSFORMATION USING C++**

#include<iostream.h>

#include<dos.h>

#include<stdio.h>

#include<conio.h>

#include<math.h>

21

#include<process.h>

#include<graphics.h>

int gd = DETECT, gm;

double x1,x2,y1,y2;

void draw\_cube(double edge[20][3])

{

initgraph(&gd,&gm,"..\bgi"); int i;

clearviewport();

for(i=0;i<19;i++)

{

x1 = edge[i][0]+edge[i][2]\*(cos(2.3562));

y1 = edge[i][1]-edge[i][2]\*(sin(2.3562));

x2 = edge[i+1][0]+edge[i+1][2]\*(cos(2.3562));

y2 = edge[i+1][1]-edge[i+1][2]\*(sin(2.3562));

line(x1+320,240-y1,x2+320,240-y2);

}

line(320,240,320,25);

line(320,240,550,240);

line(320,240,150,410);

getch();

closegraph();

}

void scale(double edge[20][3])

{

double a,b,c; int i;

cout<<"Enter the scaling Factors : =";

cin>>a>>b>>c;

initgraph(&gd,&gm, "..\bgi");

clearviewport();

for(i=0;i<20;i++)

{

edge[i][0] = edge[i][0] \* a;

edge[i][1] = edge[i][1] \* b;

edge[i][2] = edge[i][2] \* c;

}

draw\_cube(edge);

closegraph();

}

void translate(double edge[20][3])

{

int a,b,c; int i;

cout<<"Enter the Translation Factors : ="; cin>>a>>b>>c;

initgraph(&gd,&gm, "..\bgi");

clearviewport();

for(i=0;i<20;i++)

{

22

22

edge[i][0] += a;

edge[i][0] += b;

edge[i][0] += c;

}

draw\_cube(edge);

closegraph();

}

void rotate(double edge[20][3])

{

int ch,i; double temp,theta,temp1;

clrscr();

cout<<"-=[Rotation About]=-";

cout<<"1:==>X-Axis";

cout<<"2:==>Y-Axis";

cout<<"3:==>Z-Axis";

cout<<"Enter UR choice";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter the angle :=";

cin>>theta;

theta = (theta\*3.14)/180;

for(i=0;i<20;i++)

{

edge[i][0] = edge[i][0];

temp = edge[i][1];

temp1 = edge[i][2];

edge[i][1]=temp\*cos(theta)-temp1\*sin(theta);

edge[i][2]=temp\*sin(theta)+temp1\*cos(theta);

}

draw\_cube(edge);

break;

case 2:

cout<<"Enter the Angle :=";

cin>>theta;

theta=(theta\*3.14)/180;

for(i=0;i<20;i++)

{

edge[i][1]=edge[i][1];

temp = edge[i][0];

temp1= edge[i][2];

edge[i][0]=temp\*cos(theta)+temp1\*sin(theta);

edge[i][2]=-temp\*sin(theta)+temp1\*cos(theta);

}

draw\_cube(edge); break;

23

23

case 3:

cout<<"Enter the Angle :="; cin>>theta;

theta=(theta\*3.14)/180;

for(i=0;i<20;i++)

{

edge[i][2]=edge[i][2];

temp = edge[i][0];

temp1= edge[i][1];

edge[i][0]=temp\*cos(theta)-temp1\*sin(theta);

edge[i][1]=temp\*sin(theta)+temp1\*cos(theta);

}

draw\_cube(edge);

break;

}

}

void main()

{

int choice;

double edge[20][3]={100,0,0,100,100,0,0,100,0,0,100,100,0,0,100,

0,0,0,100,0,0,100,0,100,100,75,100,75,100,100,

100,100,75,100,100,0,100,100,75,100,75,100,

75,100,100,0,100,100,0,100,0,0,0,0,0,0,100,100,0,100};

while(1)

{

clrscr();

cout<<"\n\t\t3D TRANSFORMATIONS\n";

cout<<"\n\n1:==>Draw cube\n";

cout<<"\n2:==>SCALING\n";

cout<<"\n3:==>ROTATION\n\n";

cout<<"\n4:==>TRANSLATION\n\n";

cout<<"\n5:==>EXIT\n\n";

cout<<"Enter UR choice";

cin>>choice;

switch(choice)

{

case 1:

draw\_cube(edge);break;

case 2:

scale(edge);break;

case 3:

rotate(edge);break;

case 4:

translate(edge);break;

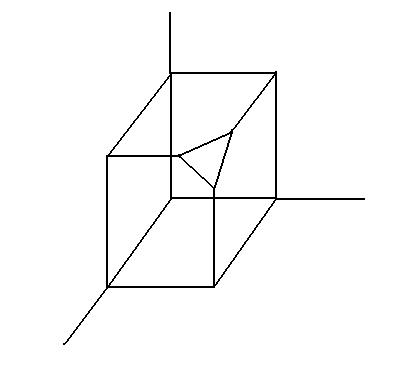
case 5:

exit(0);

default:

24

24



cout<<"Press the valid key!";

getch();break;

}

closegraph();

}

}

**OUTPUT:**

**3D TRANSFORMATIONS**

**CUBE**

1: Draw Cube

2: Scaling

3: Rotation

4: Translation

5: Exit

Enter Your Choice: 1

**3D TRANSFORMATIONS**

**ROTATION**

1: Draw Cube

2: Scaling

3: Rotation

4: Translation

5: Exit

Enter Your Choice: 3

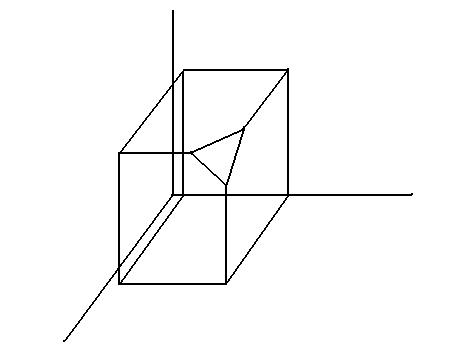
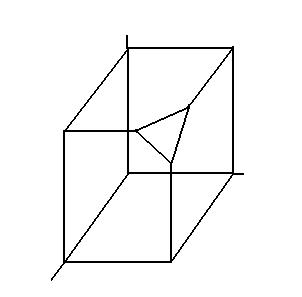
[Rotation About ] 1 : X-Axis 2: Y-Axis 3: Z-Axis

Enter Your Choice: 1

Enter The Angle: 15

25

25



**3D TRANSFORMATIONS**

**SCALING**

1: Draw Cube

2: Scaling

3: Rotation

4: Translation

5: Exit

Enter Your Choice: 2

Enter The Scaling Factors: 2 2 2

**3D TRANSFORMATIONS**

**TRANSLATION**

1: Draw Cube

2: Scaling

3: Rotation

4: Translation

5: Exit

Enter Your Choice: 4

Enter The Translation Factors:6 6 6

26

26

**EX: NO: 5 VISUALIZING PROJECTIONS OF 3D IMAGES**

**AIM:**

To write a C++ program to visualize the projections of 3D images.

**ALGORITHM:**

**Step 1:** Initialize the graphics system.

27

**Step 2:** Choose the co-ordinates (x, y, and z) for all edges of the cube and store them in a

2D Array.

**Step 3:** Draw the cube using its co-ordinates.

**Step 4:** Input the axis along which projection is to be done,(ie) x or y or z axis.

**Step 5:** Enter the reference value about which the perspective projection is to be viewed.

**Step 6:** Calculate the projection points using the co-ordinates and the reference value.

**Step 7:** Display the perspective projection of the 3D image along the axis.

**Step 8:** Exit the execution.

**PROGRAM:**

#include<iostream.h>

#include<math.h>

#include<conio.h>

#include<graphics.h>

#include<process.h>

int gd = DETECT,gm;

double x1,x2,y1,y2;

void drawcube(double edge[20][3])

{

initgraph(&gd,&gm," ");

int i;

clearviewport();

for(i=0;i<19;i++)

{

x1 = edge[i][0] + edge[i][2] \* cos(2.3562);

y1 = edge[i][1] - edge[i][2] \* sin(2.3562);

x2 = edge[i+1][0] + edge[i+1][2] \* cos(2.3562);

y2 = edge[i+1][1] - edge[i+1][2] \* sin(2.3562);

line(x1+320,240-y1,x2+320,240-y2);

}

line(320,240,320,25);

line(320,240,550,240);

line(320,240,150,410);

getch();

closegraph();

27

}

void perspect(double edge[20][3])

{

int ch,i;

double p,q,r;

clrscr();

cout<<"PERSPECTIVE PROJECTION ABOUT...\n";

cout<<"1. X-AXIS\n2. Y-AXIS\n3. Z-AXIS\n";

cout<<"ENTER UR CHOICE:\n";

cin>>ch;

switch(ch)

{

case 1:

cout<<"ENTER P:\n";

cin>>p;

for(i=0;i<20;i++)

{

edge[i][0] = edge[i][0]/(p \* edge[i][0] + 1);

edge[i][1] = edge[i][1]/(p \* edge[i][0] + 1);

edge[i][2] = edge[i][2]/(p \* edge[i][0] + 1);

}

drawcube(edge);

break;

case 2:

cout<<"ENTER Q:\n";

cin>>q;

for(i=0;i<20;i++)

{

edge[i][0] = edge[i][0]/(q \* edge[i][1] + 1);

edge[i][1] = edge[i][1]/(q \* edge[i][1] + 1);

edge[i][2] = edge[i][2]/(q \* edge[i][1] + 1);

}

drawcube(edge);

break;

case 3:

cout<<"ENTER R:\n";

cin>>r;

for(i=0;i<20;i++)

{

edge[i][0] = edge[i][0]/(r \* edge[i][2] + 1);

edge[i][1] = edge[i][1]/(r \* edge[i][2] + 1);

edge[i][2] = edge[i][2]/(r \* edge[i][2] + 1);

}

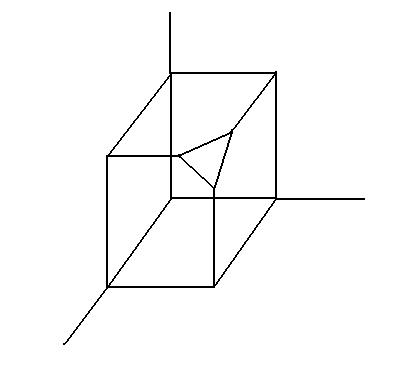
drawcube(edge);

break;

}

28

28



closegraph();

}

void main()

{

double edge[20][3] = {100,0,0,100,100,0,

0,100,0,0,100,100,

0,0,100,0,0,0,

100,0,0,100,0,100,

100,75,100,75,100,100,

100,100,75,100,100,0,

100,100,75,100,75,100,

75,100,100,0,100,100,

0,100,0,0,0,0,

0,0,100,100,0,100};

clrscr();

cout<<"\n\t3D PROJECTION\n";

getch();

drawcube(edge);

perspect(edge);

getch();

closegraph();

}

**OUTPUT:**

**3D PROJECTIONS**

**CUBE**

1: Draw Cube

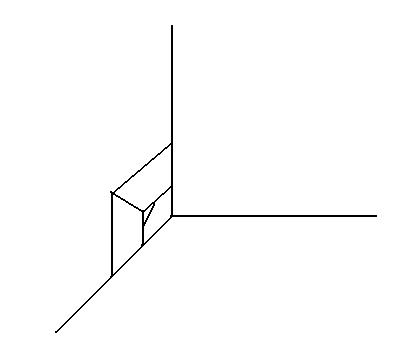
2: Perspective Projection

3: Exit

Enter Your Choice: 1

29

29



**3D PROJECTIONS**

**PERSPECTIVE PROJECTION**

1: Draw Cube

2: Perspective Projection

3: Exit

Enter Your Choice: 2

[Perspective Projection About ] 1:X-Axis 2: Y-Axis 3 : Z-Axis

Enter Your Choice: 1

Enter P: 20

30

30

**EX: NO: 6a CONVERION BETWEEN RGB TO HSV COLOR MODEL**

**AIM:**

To write a C program to convert RGB Color model into HSV Color model.

**ALGORITHM:**

**Step 1:** Start the program.

**Step 2:** Input the R, G and B values.

**Step 3:** Find the maximum and minimum of the R, G, and B values.

**Step 4:** Assign the maximum value to V.

**Step 5:** If the maximum value is not 0, calculate S as S= (max-min)/max.

**Step 6:** If the maximum value is 0, then assign the value 0 to S, and -1 to H.

**Step 7:** If R is the maximum of R, G, and B calculate H as H = (G-B)/ (max-min).

**Step 8:** If Value of G is maximum, calculate H as H= 2+ (B-R)/ (max-min).

**Step 9:** If B is maximum, Calculate H as H = 4+(R-G)/ (max-min).

**Step 10:** Multiply H by 60.

**Step 11:** If H is less than 0, add 360 to H.

**Step 12:** Divide H by 360.

**Step 13:** Print the values of H, S and V.

**Step 14:** Stop the program.

**PROGRAM:**

const no\_hue = -1;

#include<stdio.h>

#include<math.h>

#include<conio.h>

#define min(a,b) (a<b?a:b)

#define max(a,b) (a>b?a:b)

void rgbtohsv(float r,float g,float b)

{

float h,s,v;

float max1 = max(r,max(g,b)),min1 = min(r,min(g,b));

float delta = max1 - min1;

v = max1;

if(max1 != 0.0)

s = delta/max1;

else

s = 0.0;

if(s == 0.0)

h = no\_hue;

else

{

if(r == max1)

h = (g-b)/delta;

else if(g == max1)

31

31

h = 2 + (b-r)/delta;

else if(b == max1)

h = 4 + (r-g)/delta;

h \*= 60.0;

if(h < 0)

h += 360.0;

h /= 360.0;

printf("The HSV values are:\n");

printf("h = %f s = %f v = %f",h,s,v);

}

}

void main()

{

float r,g,b;

clrscr();

printf("ENTER THE RGB VALUES(values from 0 to 1)\n");

scanf("%f%f%f",&r,&g,&b);

rgbtohsv(r,g,b);

getch();

}

**OUTPUT:**

**CONVERT FROM RGB TO HSV COLOR MODEL**

Enter the RGB values (Enter the values from 0 to 1):

32

0.1

0.9

0.3

The HSV values are:

h=0.375000

s=0.888889

v=0.900000

32

**EX: NO: 6b CONVERSION BETWEEN HSV TO RGB COLOR MODEL**

**AIM:**

To write a C program to convert RGB Color model into HSV Color model.

**ALGORITHM:**

**Step 1:** Start the program.

**Step 2:** Input the H, S, and V values.

**Step 3:** If value of S is 0, assign the value of V to R, G and B.

**Step 4:** If S is not 0, perform the following steps,

 If H is 1, assign the value of 0 to H.

33

 Multiply H by 60. Round the value of H to the nearest integer as assign it

to ‘i’.

 Calculate f as the difference between H and i.

 Calculate the following values:

aa =v (1-s)

bb =v (1-sf)

cc =v (1-s (1-f))

 Based on the value of I, assign the values of R, G and B with aa, bb, and

cc values.

**Step 5:** Print the values of R, G, and B.

**Step 6:** Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<math.h>

void hsvtorgb(float h,float s,float v)

{

int i;

float r,g,b;

float aa,bb,cc,f;

if(s == 0)

r = g = b = v;

else

{

if(h == 1.0)

h = 0;

h \*= 6.0;

i = floor(h);

f = h - i;

aa = v \* (1-s);

bb = v \* (1-(s\*f));

cc = v \* (1-(s\*(1-f)));

switch(i)

{

33

case 0:

r = v;

g = cc;

b = aa;

break;

case 1:

r = bb;

g = v;

b = cc;

break;

case 2:

r = aa;

g = bb;

b = v;

break;

case 3:

r = aa;

g = bb;

b = v;

break;

case 4:

r = cc;

g = aa;

b = v;

break;

case 5:

r = v;

g = aa;

b = bb;

break;

}}

printf("The RGB values are:\n");

printf("r = %f g = %f b = %f",r,g,b);

}

void main()

{

float h,s,v;

clrscr();

printf("ENTER THE HSV VALUES(values from 0 to 1)\n");

scanf("%f%f%f",&h,&s,&v);

hsvtorgb(h,s,v);

getch();

}

34

34

**OUTPUT:**

**CONVERT FROM HSV TO RGB COLOR MODEL**

Enter the HSV values (Enter the values from 0 to 1):

35

0.4

0.5

0.6

The RGB values are:

r=0.300000

g=0.480000

b=0.600000

**EX: NO: 7 IMPLEMENTATION OF TEXT COMPRESSION ALGORITHM**

**AIM:**

To write a C program to compress a Text file using Text Compression Algorithm.

**ALGORITHM:**

**Step 1:** Start the program.

**Step 2:** Open the source Text file to be compressed and the Destination file to store the

compressed version of the source.

**Step 3:** Read the characters from the source file one by one.

**Step 4:** Check for the redundancy of the same read character in the consecutive character

stream till different character is read.

**Step 5:** Write the character and the number of its consecutive occurrence in the

destination file.

**Step 6:** Continue the process until the source reaches the end-of-File character.

**Step 7:** Close the source and destination files.

**Step 9:** Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void main()

{

FILE \*fp,\*fp1;

char a,c;

int i=1;

clrscr();

fp = fopen("z:/graphics/read.txt","r");

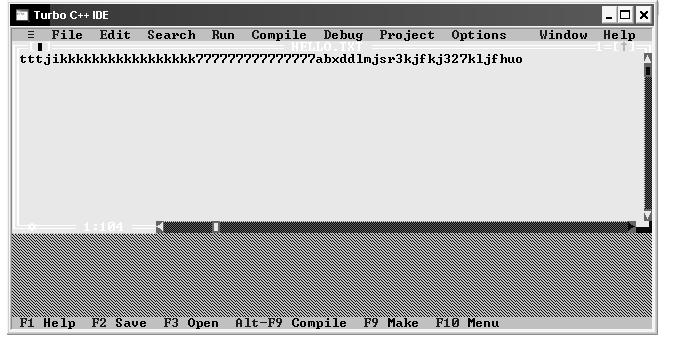
fp1 = fopen("z:/graphics/write.txt","w");

a = fgetc(fp);

while(!feof(fp))

{

35



c = fgetc(fp);

if(c == a)

{

a = c;

i++;

}

else

{

fprintf(fp1,"%d%c",i,a);

a = c;

i = 1;

}

}

printf("SUCCESSFULLY COMPRESSED");

fclose(fp);

fclose(fp1);

getch();

}

**OUTPUT:**

**TEXT COMPRESSION**

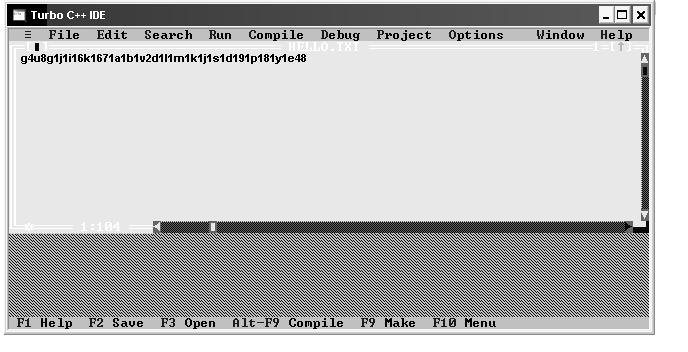
The output is “Successfully Compressed”

1. Create an Input Text File Named “Hello.txt” Before Compiling the Program

36

2. The Output Text file is created and the data is compressed. The Output file created is

36

37

37

38

**EX: NO: 8 IMPLEMENTATION OF IMAGE COMPRESSION ALGORITHM**

**AIM:**

To write a C program to compress a Image file using Image Compression

Algorithm.

**ALGORITHM:**

**Step 1:** Start the program.

**Step 2:** Open source image file to be compressed and the destination file to store the

compressed version of the source.

**Step 3:** Read the pixels from the source image file one by one.

**Step 4:** Check for the redundancy of the same read pixel in the consecutive pixels till a

different color pixel is read.

**Step 5:** Write the pixel and the number of its consecutive occurrence in the destination

file.

**Step 6:** Continue the process until the source reaches the end-of-file.

**Step 7:** Close the source and the destination files.

**Step 8:** Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void main()

{

FILE \*fp1,\*fp2;

char a,c;

int i=1;

clrscr();

fp = fopen("z:/graphics/loadus.bmp","r");

fp1 = fopen("z:/graphics/me.bmp","w");

a = fgetc(fp1);

while(!feof(fp1))

{

c = fgetc(fp1);

if(c == a)

{

a = c;

i++;

}

else

{

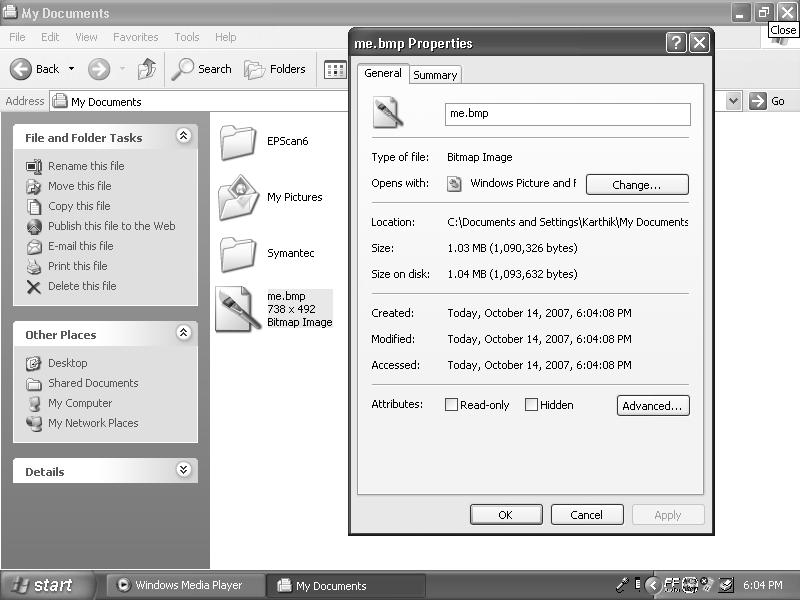
fprintf(fp2,"%d%c",i,a);

a = c;

i = 1;

}

38



}

printf("SUCCESSFULLY COMPRESSED");

fclose(fp1);

fclose(fp2);

getch();

}

**OUTPUT:**

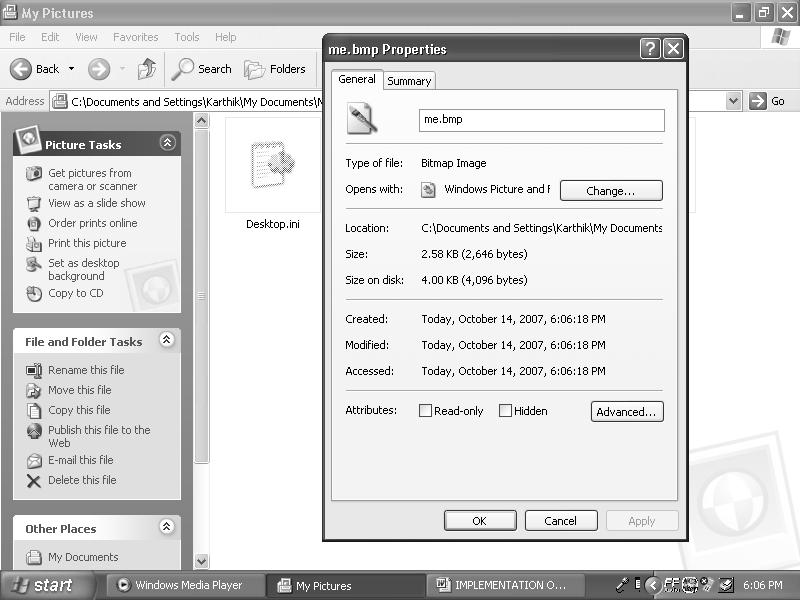
**IMAGE COMPRESSION**

The Output will be “Compressed”

The size of the picture file “me.bmp” before compression:

39

39



The size of the picture file “me.bmp” after compression:

40

40

**EX: NO: 9 ANIMATION USING MACROMEDIA FLASH**

**MACROMEDIA FLASH:**

41

Flash is versatile multimedia presentation software for producing interactive

animated representation with new features like motion, tweening, motion guiding,

importing, and other multimedia elements such as image, sound, and video.

**MOVIE:**

The final product of macromedia flash presentation is known as movie. Flash

presentation is known as movie. It contains animated paths and transitions of various

multimedia elements which are classified as follows. Movies are stored with the

extension .fla.

**SCENE:**

A flash movie is a collection of one or more scenes. Each scene contains

several layers which are time based.

**LAYERS:**

A layer is nothing but a collection of frames which are arranged on a timeline

(msec). There are three types of layers,

 Normal layer

 Mask layer

 Motion Guide layer.

**TIMELINE:**

 It is used to show the parts of animation in milliseconds. Each Layer is having a

timeline and it has up to 640 ms of frames. The timeline is equally divided into

rectangle representation called frames. Timeline will help the user to look and

design the faces of animation whenever needed.

 Collection of frames will make up a timeline. Frames can be added to continue

the presentation with the constant source of multimedia elements.

**TYPES OF FRAMES:**

**KEYFRAME:**

 It is a frame which contains a key element which can be tweened.

 TWEENIG is the process of flash to determine the Animation path between the

two key frame positions of multimedia element.

**BLANK KEYFRAME:**

 These key frames are used to insert audio clips which are synchronized with other

animation layers.

 Frames can be copied from one scene to another, one layer to another or one

portion of the timeline to another. Frames can be detected, cleared, copied, and

pasted.

**MASK LAYER:**

This acts as a filter to the next layer where the animation in the next layer is

visible only through the objects in the mask layer.

41

**MOTION GUIDE:**

Animation path of a layer can be determined by the users with the help of

motion guide layer, like masking. Two layers are involved,

1. Guided Layer

2. Guiding Layer.

**GUIDED LAYER:**

Animation sequence formed in this layer simply follows a previous guide

path motion tweens to create an animation sequence.

**GUIDING LAYER:**

42

This layer contains a hand tool drawing which acts as a motion path for next

layer animation.

**SYMBOLS:**

An animation scene can be saved as a symbol. A symbol is nothing but a

multimedia element such us text image, sound or another scene created using flash.

Each symbol may be added to the library.

**LIBRARIES:**

A Flash movie library contains dip buttons, images, sound files and other

symbols. It is often used to store multimedia elements among flash movie buttons used to

set predefined actions during the animation sequence.

**TOOL BARS:**

It contains text, rectangle, circle, ellipse, pen, full border, color ink/touch

color, erasers, line picture tools and free hand tools.

**ACTIONS:**

Macromedia flash contains very simple scripting language to control the

sequence of animation during course of a movie.

**EXPORTING:**

Collection of scenes can be tested using text movie option and exported.

The final product can be integrated with macromedia director for processing a complete

interactive multimedia presentation. A movie can be exported to several other forms

including BMP, GIF, TIFT, etc…

**SIMPLE PRESENTATION USING MOTION TWEEN**

**PROCEDURE:**

**Step 1:** Create a view presentation using motion tween.

**Step 2:** Draw any object in the workspace and fix its critical place.

**Step 3:** The frames on the timeline are copied from the initial frame up to the frame by

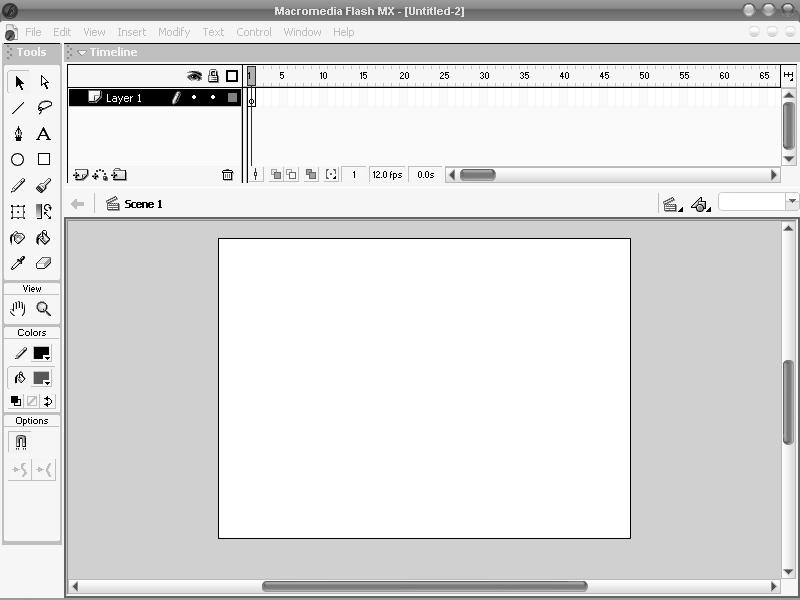
the right clicking of mouse and choosing copy frames.

**Step 4:** The object is moved to its target position from its initial position as soon as the

final frame are being copied in the timeline.

**Step 5:** The object is provided with a motion or any movements any where the motion

42



tween is created for that object, so right click the mouse and choose create

motion tween in the last but one frame.

**Step 6:** This creates the arrow head that notifies the duration of animation frame. The

first frame position to the last frame position.

43

**Step 7:** Then press cntrl+Enter keys or select text scene from control menu to view the

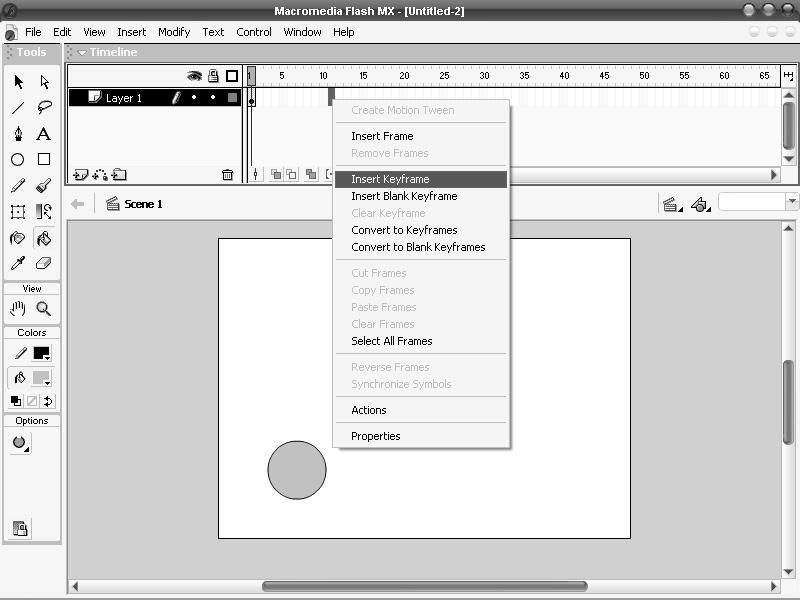
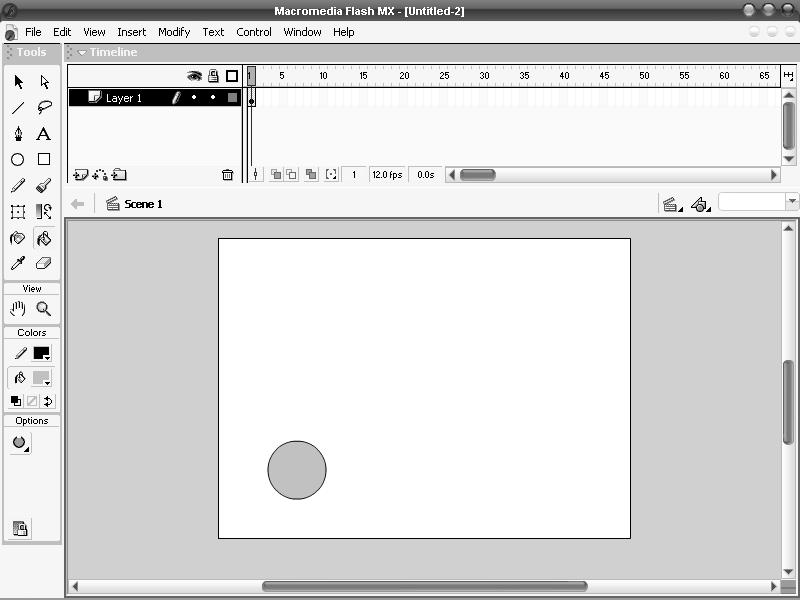
animation.

**Step 8:** Thus the motion tween is created.

**MACROMEDIA FLASH**

**1.** **SELECT FILE -> NEW**

43



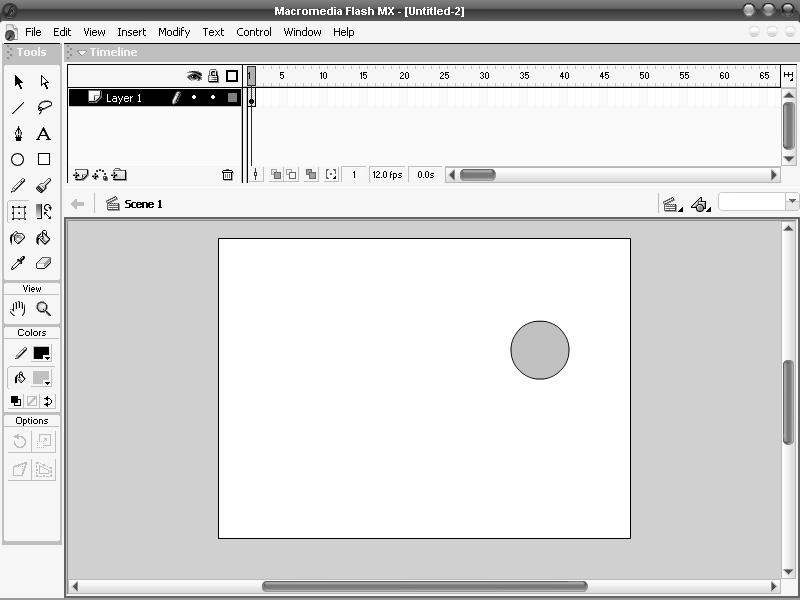
**2. DRAW OR SELECT THE OBJECT TO PERFORM ANIMATION**

**3. MOVES THE OBJECT AND SET *INSERT KEY FRAME* IN SELECTED**

**PLACES**

44

44



**4. MOVING THE OBJECT**

45

45

46

**EX: NO: 10 BASIC OPERATIONS ON IMAGE USING ADOBE PHOTOSHOP**

The Adobe Photoshop work area includes the command menus at the top

of the screen and a variety of tools and palettes for editing and adding elements to your

image. Commands and filters can also be added to the menus by installing third party

software known as plug-in modules.

**USING THE TOOLBOX:**

The first time the application is started, the tool box appears on the left

side of the screen. The tools in the toolbox lets us select, paint, edit and view images.

These tools have options that appear in the options box. Other controls choose foreground

and background colors and change the slice or screen display mode.

**TO SHOW OR HIDE THE TOOL BOX:**

 Window Show Tools

 Window Hide Tools

Both from the Menu Bar.

**TO MOVE THE TOOLBOX:**

Drag the toolbox by holding the mouse on its title bar.

**ROTATING AND FLIPPING ENTIRE IMAGES:**

The Rotate canvas command lets us rotate or flip an entire image. The

commands do not work on individual layers or parts of layers, paths or selection border.

**TO ROTATE OR FLIP AN ENTIRE IMAGE:**

Choose Image Rotate canvas and choose one of the options.

 90° to rotate the image clockwise by a quarter-turn.

 180° to rotate the image by a half-turn.

 90° counter clock-wise to rotate the image couter clockwise by a

quarter turn.

 Arbitrary to rotate the image by the angle you specify. If this option is

choosen, enter an angle between -359.99 and 359.99 in the angle text

box, and then select clockwise or counterclockwise to rotate clockwise

or counter clockwise and then click ok.

 Flip Horizontal to flip the image horizontally, along the vertical axis.

 Flip vertical to flip the image vertically, along the horizontal axis.

**USING FILTERS:**

To use a filter, choose the appropriate submenu command from the filter menu.

Guidelines to choose filters:

 The last filter chosen appears at the top of menu.

 Filters are applied to the active, visible layer.

 Filters cannot be applied to Bitmap-mode, indexed-mode or 16-bit per

channel images.

 Some filters only work on RGB Images.

 Some filters are entirely processed in RAM.

**DISTORT FILTERS:**

The Distort filters geometrically distort an image, creating 3D or other reshaping

effects. These filters can be very intensive in memory.

46

**DIFFUSE GLOW:**

Renders as image as though it were viewed through a self diffusion filter. The

47

filter adds see- through white noise to an image, with the glow fading from the center of a

selection.

**DISPLACE FILE:**

 Uses an image called a displacement map, to determine how to distort a

selection. For eg, using a parabola shaped displacement map, you can

create an image that appears to be printed on a cloth held at its corners.

 Creates displacement maps using a flattened file saved in Adobe

Photoshop format(except Bitmap mode images) or one that is saved with

the include composite image with layered files options selected in the

saving files preferences. Files included with the software can also be used.

**TO USE THE DISPLACE FILTER:**

 Choose FilterDistort Displace.

 Enter the scale for the magnitude of the displacement. When the horizontal

and vertical scale are set to 100% the greatest displacement is 128 pixels.

 If the Displacement map is not the same size as the selection, choose how

the map will fit the image sketch to fit to resize the map, or tile to fill the

selection by repeating the map in a pattern.

 Choose Wrap Around or Repeat Edge pixels to determine how undistorted

areas of the image will be treated.

 Click the Ok button.

 Select and open the displacement map. The distortion is applied to the

image.

 The displaced filter shifts a selection using a color value from the

displacement map.0 is the maximum negative shift, 255 is the maximum

positive shift, and a gray scale of 128 produces no displacement. If a map

has one channel, the image shifts along a diagonal defined by the

horizontal and vertical scale ratios. If the map has more than 1 channel,

the first channel controls the horizontal displacement and the second

channel controls the vertical displacement.

**RIPPLE:**

Creates an undulating pattern on a selection like ripples on the surface of a pond.

For greater control, use the wave filter. Options include the amount and size of ripples.

**SHEAR:**

Distorts an image along a curve. Specify the curve by dragging the line in the

box to form a curve for the distortion. You can adjust any point along the curve. Click

default to return the curve to a straight line. In addition, you choose how to treat

undistorted areas.

**SPHERIZE:**

Gives objects a 3D effect by wrapping a selection around a spherical shape,

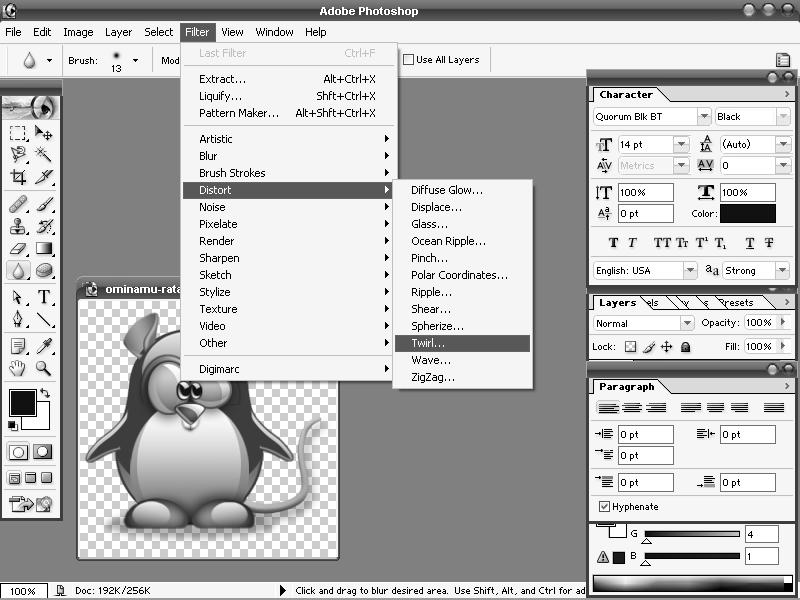
distorting the image and stretching it to fit the selected curve.

**TWIRL:**

Rotates a selection more sharply in the center than at edges. Specifying an angle

produces a twirl pattern.

47



**OPEN AN IMAGE FILE**

**ADOBE PHOTOSHOP**

48

**SELECT FILTER -> DISTORT -> TWIRL**

48

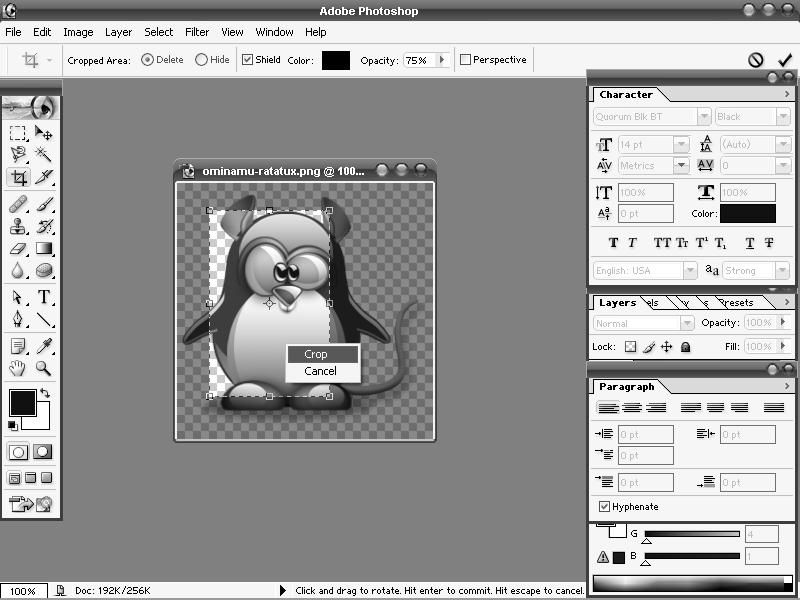


**THE TWRILED IMAGE:**

**SELECT THE CROP TOOL:**

49

49



**SELECT THE PORTION TO CROP:**

50

50