BASIC GRAPHICS FUNCTION

**1) initgraph()**

initgraph() function initializes the graphics mode and clears the screen.

**Declaration:**

void far initgraph(int far \*driver, int far \*mode, char far \*path)

**2)**  **detectgraph()**

Detectgraph function determines the graphics hardware in the system, if the function finds a graphics adapter then it returns the highest graphics mode that the adapter supports.

**Declaration:** void far detectgraph(int far \*driver, int far \*mode)

Integer that specifies the graphics driver to be used. You can give graphdriver a value using a constant of the graphics\_drivers enumeration type.

**3) closegraph()**

closegraph() function switches back the screen from graphcs mode to text mode. It clears the screen also.

A graphics program should have a closegraph function at the end of graphics. Otherwise DOS screen will not go to text mode after running the program.

**4)getpixel()**

getpixel function returns the color of pixel present at location(x, y).

**Declaration :-**

int getpixel(int x, int y);

**5) putpixel()**

putpixel function plots a pixel at location (x, y) of specified color.

**Declaration :-**

void putpixel(int x, int y, int color);

For example if we want to draw a GREEN color pixel at (35, 45) then we will write putpixel(35, 35, GREEN); in our c program, putpixel function can be used to draw circles, lines and ellipses using various algorithms.

**6) line()**

line function is used to draw a line from a point(x1,y1) to point(x2,y2) i.e. (x1,y1) and (x2,y2) are end points of the line.

**Declaration :-**

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

**7) lineto()**

lineto function draws a line from current position(CP) to the point(x,y), you can get current position using getx and gety function.

**8) circle()**

circle function is used to draw a circle with center (x,y) and third parameter specifies the radius of the circle.

**Declaration :-**

void circle(int x, int y, int radius);

**9)ellipse()**

Ellipse is used to draw an ellipse (x,y) are coordinates of center of the ellipse, stangle is the starting angle, end angle is the ending angle, and fifth and sixth parameters specifies the X and Y radius of the ellipse. To draw a complete ellipse strangles and end angle should be 0 and 360 respectively.

**Declaration :-**void ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);

**10) drawpoly()**

drawpoly function is used to draw polygons i.e. triangle, rectangle, pentagon, hexagon etc.

**Declaration :-**

void drawpoly( int num, int \*polypoints );

num indicates (n+1) number of points where n is the number of vertices in a polygon, polypoints points to a sequence of (n\*2) integers . Each pair of integers gives x and y coordinates of a point on the polygon. We specify (n+1) points as first point coordinates should be equal to (n+1)th to draw a complete figure.

To understand more clearly we will draw a triangle using drawpoly, consider for example the array :-  
int points[] = { 320, 150, 420, 300, 250, 300, 320, 150};

points array contains coordinates of triangle which are (320, 150), (420, 300) and (250, 300). Note that last point(320, 150) in array is same as first.

## 11) outtext ()

outtext function displays text at current position.

**Declaration :-**

void outtext(char \*string);

## 12) outtextxy ()

outtextxy function display text or string at a specified point(x,y) on the screen.

**Declaration :-**

void outtextxy(int x, int y, char \*string);  
x, y are coordinates of the point and third argument contains the address of string to be displayed.

**13)rectangle()**

Rectangle function is used to draw a rectangle. Coordinates of left top and right bottom corner are required to draw the rectangle. left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner.

**Declaration :-**

void rectangle(int left, int top, int right, int bottom);

**14) floodfill()**

floodfill function is used to fill an enclosed area. Current fill pattern and fill color is used to fill the area.(x, y) is any point on the screen if (x,y) lies inside the area then inside will be filled otherwise outside will be filled,border specifies the color of boundary of area.

**Declaration :-**

void floodfill(int x, int y, int border);

**15)fillpoly()**

f illpoly function draws and fills a polygon. It require same arguments as drawpoly.

**Declaration :-**

void drawpoly( int num, int \*polypoints );

**16)fillellipse()**

f illellipse function draws and fills a polygon.

**Declaration:-** void fillellipse(int x, int y, int xradius, int yradius);  
 x and y are coordinates of center of the ellipse, xradius and yradius are x and y radius of ellipse respectively.

**//EX.NO: 1 Basic Primitives (Generation of Smiley Face)**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TC\\BGI");

setcolor(BLACK);

circle(300,300,100);

circle(250,280,10);

circle(350,280,10);

line(300,300,290,330);

line(290,330,310,330);

arc(300,300,240,300,70);

getch();

}

OUTPUT:

RESULT:

**//Ex. No. 2(a) DDA Line Drawing Algorithm**

ALGORITHM:

1 : Start.

2 : Initialize the graphics header files and functions.

3 : Declare the required variables and functions.

4 : Get the four points for drawing a line namely x1,x2,y1,y2.

5 : Calculate dx = x2-x1

dy = y2-y1.

6 : Check

if (dx > dy)

then

steps=dx

else

steps=dy

7 : Calculate

xinc= dx / steps

yinc= dy / steps

8: Plot the first point(x1,y1)

9: For steps times calculate

x=x+xinc

y=y+yinc

plot all the points.

10: Stop

PROGRAM:

#include "stdio.h"

#include "conio.h"

#include "math.h"

#include "graphics.h"

main()

{

int gd=DETECT,gm;

int xa,xb,ya,yb;

int dx,dy,steps,k,xinc,yinc,x,y;

initgraph(&gd,&gm,"c:\\tc\\bgi");

printf("Enter the two left end pixel points(xa,ya):\n");

scanf("%d%d",&xa,&ya);

printf("Enter the two Right end pixel points(xb,yb):\n");

scanf("%d%d",&xb,&yb);

dx=xb-xa;

dy=yb-ya;

if(abs(dx)>abs(dy))

steps=abs(dx);

else

steps=abs(dy);

xinc=dx/steps;

yinc=dy/steps;

x=xa;

y=ya;

putpixel(x,y,6);

for(k=1;k<=steps;k++)

{

x=x+xinc;

y=y+yinc;21

putpixel(x,y,6);

}

getch();

return(0);

}

**OUTPUT**

Enter The Two Left Endpoints(xa,ya) : 0 0

Enter The Two Right Endpoints(xb,yb) : 50 150

RESULT:

**// EX.No 2(b) Bresenham’s Line Drawing Algorithm**

**ALGORITHM:**

1 : Start.

2 : Initialize the graphics header files and functions.

3 : Declare the required variables and functions.

4 : Get the four points for drawing a line namely x1,x2,y1,y2.

5: Calculate dx = x2-x1

dy = y2-y1.

6:Calculate P0= 2\*dy-dx

7: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

8: Perform step4 ∆x times

9: Stop

PROGRAM:

#include "stdio.h"

#include "conio.h"

#include "math.h"

#include "graphics.h"

main()

{

int gd=DETECT,gm;

int xa,xb,ya,yb;

int dx,dy,x,y,xend,p;

initgraph(&gd,&gm,"c:\\tc\\bgi");

printf("Enter The Two Left Endpoints(xa,ya):\n");

scanf("%d%d",&xa,&ya);

printf("Enter The Two Right Endpoints(xb,yb):\n");

scanf("%d%d",&xb,&yb);

dx=abs(xa-xb);

dy=abs(ya-yb);

p=2\*dy-dx;

if(xa>xb)

{

x=xb;

y=yb;

xend=xa;

}

else

{

x=xa;

y=ya;

xend=xb;

}

putpixel(x,y,6);

while(x<xend)

{

x=x+1;

if(p<0)

{

p=p+2\*dy;

}

else

{

y=y+1;

p=p+2\*(dy-dx);

}

putpixel(x,y,6);

}

getch();

return(0);

}

**INPUT**

Enter The Two Left Endpoints(xa,ya): 234 124

Enter The Two Right Endpoints(xb,yb): 578 321

Result

**//Ex. No 2(c) Mid-Point Circle Drawing Algorithm**

**ALGORITHM:**

1: Start.

2: Initialize the graphics header files and functions.

3: Declare the required variables and functions.

4: Input radius r and circle center (xc,yc) and obtain the first point on the circumference of the circle centered on the origin as (x0,y0) = (0,r) .

5: Calculate the initial value of the decision parameter as P0=(5/4)-r

6: 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+1-2 yk+1

Where 2xk+1=2xk+2 and 2yk+1=2yk-2

7: Determine symmetry points in the other seven octants.

8: Move each calculated pixel position (x,y) onto the circular path centered at (xc,yc) and plot the coordinate values.

x=x+xc y=y+yc

9: Repeat step 6 through8 until x>=y.

10: Stop.

PROGRAM:-

#include "stdio.h"

include "conio.h"

include "math.h"

include "graphics.h"

main()

{

int gd=DETECT,gm;

int xcenter,ycenter,radius;

int p,x,y;

initgraph(&gd,&gm,"c:\\tc\\bgi");

x=0;

printf("Enter The Radius Value:\n");

scanf("%d",&radius);

y=radius;

printf("Enter The xcenter and ycenter Values:\n");

scanf("%d%d",&xcenter,&ycenter);

plotpoints(xcenter,ycenter,x,y);

p=1-radius;

while(x<y)

{

if(p<0)

x=x+1;

else

{

x=x+1;

y=y-1;

}

if(p<0)

p=p+2\*x+1;

else

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

plotpoints(xcenter,ycenter,x,y);

}

getch();

return(0);

}

int plotpoints(int xcenter,int ycenter,int x,int y)

{

putpixel(xcenter+x,ycenter+y,1);

putpixel(xcenter-x,ycenter+y,1);

putpixel(xcenter+x,ycenter-y,1);

putpixel(xcenter-x,ycenter-y,1);

putpixel(xcenter+y,ycenter+x,1);

putpixel(xcenter-y,ycenter+x,1);

putpixel(xcenter+y,ycenter-x,1);

putpixel(xcenter-y,ycenter-x,1);

}

**Input**

Enter The Radius Value: 30

Enter The xcenter and ycenter Values: 100 100

**//Ex.No 2(d) Mid-point Ellipse Drawing Algorithm**

**ALGORITHM:**

1. Start.
2. Initialize the graphics header files and functions.
3. Declare the required variables and functions.
4. 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)
5. Calculate the initial value of the decision parameter in region 1 as

P10=ry2-rx2ry +(1/4)rx2

1. At each xk position in region1 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 +2 ry2xk +1 + ry2

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

p1k+1 = p1k +2 ry2xk +1 - 2rx2 yk+1 + ry2

with 2 ry2xk +1 = 2 ry2xk + 2ry2

2 rx2yk +1 = 2 rx2yk + 2rx2

And continue until 2ry2 x>=2rx2 y

1. Calculate the initial value of the decision parameter in region 2 using the last point (x0,y0) is the last position calculated in region 1.

p20 = ry2(x0+1/2)2+rx2(yo-1)2 – rx2ry2

1. At each position yk 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 ellipse is (xk+1,yk-1) and

p2k+1 = p2k + 2ry2xk+1 – 2rxx2yk+1 + rx2

Using the same incremental calculations for x any y as in region 1.

1. Determine symmetry points in the other three quadrants.
2. Move each calculate pixel position (x,y) onto the elliptical path centered on (xc,yc) and plot the coordinate values

x=x+xc, y=y+yc

1. Repeat the steps for region2 unit y > 0.
2. Stop.

PROGRAM:-

#include "stdio.h"

#include "conio.h"

#include "math.h"

#include "graphics.h"

main()

{

int gd=DETECT,gm;

int xcenter,ycenter,rx,ry;

int p,x,y,px,py,rx1,ry1,rx2,ry2;

initgraph(&gd,&gm,"c:\\tc\\bgi");

printf("Enter The Radius Value:\n");

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

printf("Enter The xcenter and ycenter Values:\n");

scanf("%d%d",&xcenter,&ycenter);

ry1=ry\*ry;

rx1=rx\*rx;

ry2=2\*ry1;

rx2=2\*rx1;

**/\* REGION 1 \*/**

x=0;

y=ry;

plotpoints(xcenter,ycenter,x,y);

p=(ry1-rx1\*ry+(0.25\*rx1));

px=0;

py=rx2\*y;

while(px<py)

{

x=x+1;

px=px+ry2;

if(p>=0)

y=y-1;

py=py-rx2;

if(p<0)

p=p+ry1+px;

else

p=p+ry1+px-py;

plotpoints(xcenter,ycenter,x,y);

**/\* REGION 2\*/**

p=(ry1\*(x+0.5)\*(x+0.5)+rx1\*(y-1)\*(y-1)-rx1\*ry1);

while(y>0)

{

y=y-1;

py=py-rx2;

if(p<=0)

{

x=x+1;

px=px+ry2;

}

if(p>0)

p=p+rx1-py;

else

p=p+rx1-py+px;

plotpoints(xcenter,ycenter,x,y);

}

}

getch();

return(0);

}

int plotpoints(int xcenter,int ycenter,int x,int y)

{

putpixel(xcenter+x,ycenter+y,6);

putpixel(xcenter-x,ycenter+y,6);

putpixel(xcenter+x,ycenter-y,6);

putpixel(xcenter-x,ycenter-y,6);

}

**INPUT**

Enter The Radius Value(Rx,Ry) : 10 30

Enter The xcenter and ycenter Values : 300 150

**Result**/**Ex. No 4 – 2D Transformation**

#include<graphics.h>

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#include<math.h>

void display(int n, float c[][3])

{

float maxx,maxy;

int i;

maxx=getmaxx();

maxy=getmaxy();

maxx=maxx/2;

maxy=maxy/2;

i=0;

while(i<n-1)

{

line(maxx+c[i][0],maxy-c[i][1],maxx+c[i+1][0],maxy-c[i+1][1]);

i++;

}

i=n-1;

line(maxx+c[i][0],maxy-c[i][1],maxx+c[0][0],maxy-c[0][1]);

setcolor(GREEN);

line(0,maxy,maxx\*2,maxy);

line(maxx,0,maxx,maxy\*2);

setcolor(WHITE);

}

void mul(int n,float b[][3],float c[][3],float a[][3])

{

int i,j,k;

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

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

a[i][j]=0;

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

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

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

{

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

}

}

void translation(int n,float c[][3],float tx,float ty)

{

int i;

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

{

c[i][0]=c[i][0]+tx;

c[i][1]=c[i][1]+ty;

}

}

void scaling(int n,float c[][3],float sx,float sy)

{

float b[10][3],a[10][3];

int i=0,j;

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

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

b[i][j]=0;

b[0][0]=sx;

b[1][1]=sy;

b[2][2]=1;

mul(n,b,c,a);

setcolor(RED);

display(n,a);

}

void rotation(int n,float c[][3],float ra)

{

int i=0,j;

float b[10][3],xp,yp,a[10][3];

xp=c[0][0];

yp=c[0][1];

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

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

b[i][j]=0;

b[0][0]=b[1][1]=cos(ra\*3.14/180);

b[0][1]=sin(ra\*3.14/180);

b[1][0]=-sin(ra\*3.14/180);

b[2][0]=(-xp\*cos(ra\*3.14/180))+(yp\*sin(ra\*3.14/180))+xp;

b[2][1]=(-xp\*sin(ra\*3.14/180))-(yp\*cos(ra\*3.14/180))+yp;

b[2][2]=1;

mul(n,b,c,a);

setcolor(RED);

display(n,a);

getch();

}

void reflection(int n,float c[][3])

{

float b[10][3],a[10][3];

int i=0,ch,j;

cleardevice();

printf("\nabout X-axis");

display(n,c);

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

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

{

b[i][j]=0;

if(i==j)

b[i][j]=1;

}

b[1][1]=-1;

mul(n,b,c,a);

setcolor(RED);

display(n,a);

getch();

}

void shearing(int n,float c[][3])

{

float b[10][3],sh,a[10][3];

int i=0,ch,j;

cleardevice();

printf("\nX-shearing");

printf("\nEnter the val for shear : ");

scanf("%f",&sh);

clrscr();

cleardevice();

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

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

b[i][j]=0;

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

b[i][i]=1;

b[1][0]=sh;

mul(n,b,c,a);

setcolor(RED);

display(n,a);

}

void main()

{

int i,j,k,cho,n;

int gd=DETECT,gm;

float c[10][3],tx,ty,sx,sy,ra;

initgraph(&gd,&gm,"Z:\\tc\\bgi");

printf("\nEnter no. of vertices : ");

scanf("%d",&n);

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

{

printf("\nEnter coordinates of %d vertex : ",i+1);

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

c[i][2]=1;

}

do

{

clrscr();

cleardevice();

printf("\n1.Translation\n2.Scaling\n3.Rotation\n4.Reflection\n5.Shearing\n6.Exit");

printf("\nEnter UR choice : ");

scanf("%d",&cho);

switch(cho)

{

case 1:

printf("\nTranslation factor for x and y-axis : ");

scanf("%f%f",&tx,&ty);

clrscr();

cleardevice();

setcolor(BLUE);

display(n,c);

getch();

translation(n,c,tx,ty);

setcolor(RED);

display(n,c);

getch();

break;

case 2:

printf("\nScaling factor for x and y-axis : ");

scanf("%f%f",&sx,&sy);

clrscr();

cleardevice();

setcolor(BLUE);

display(n,c);

getch();

scaling(n,c,sx,sy);

getch();

break;

case 3:

printf("\nEnter angular rotation : ");

scanf("%f",&ra);

clrscr();

cleardevice();

setcolor(BLUE);

display(n,c);

getch();

rotation(n,c,ra);

setcolor(RED);

break;

case 4:

clrscr();

cleardevice();

setcolor(BLUE);

display(n,c);

getch();

reflection(n,c);

getch();

break;

case 5:

clrscr();

cleardevice();

setcolor(BLUE);

display(n,c);

getch();

shearing(n,c);

getch();

break;

case 6:

exit(0);

default :

printf("Invalid choice!!!");

break;

}

} while(cho!=6);

getch();

closegraph();

}

**OUTPUT:**

Enter no. of vertices : 2

Enter coordinates of 1 vertex :10

10

Enter coordinates of 2 vertex :40

40

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter UR choice : 1

Translation factor for x and y-axis : 40

40

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter UR choice : 2

Scaling factor for x and y-axis : 2

2

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter UR choice : 3

Enter angular rotation : 180

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter UR choice : 4

About X-axis

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter UR choice : 5

X-shearing

Enter the val for shear : 2

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter UR choice : 6

**Result**

//**Ex. No 9– Window to Viewport mapping**

**ALGORITHM:**

Objects and primitives in the application model will be stored in world coordinates. To display the appropriate images on screen it is necessary to map world coordinates to screen or device coordinates. This transformation is known as **window-to-viewport transformation**.

Window - Area that defines what is to be displayed.

Viewport - Area that defines where it is to be displayed.

1.      Start.

2.     Initialize graphics mode.

3.     Draw a window.

4.     Get the object.

5.     Translate the object together with its window until the lower left corner of the window is at origin.

6.     Object and window are scaled until the window has the dimensions of the viewport.

7.     Translate the viewport to its correct position on the screen.

8.     Display the contents inside the viewport.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

main()

{

float sx,sy;

int w1,w2,w3,w4,x1,x2,x3,x4,y1,y2,y3,y4,v1,v2,v3,v4;

int gd=DETECT,gm;

initgraph(&gd,&gm,"c:\\tc\\bgi");

printf("Enter The Coordinate x1,y1,x2,y2,x3,y3\n");

scanf("%d%d%d%d%d%d",&x1,&y1,&x2,&y2,&x3,&y3);

cleardevice();

w1=5;

w2=5;

w3=635;

w4=465;

rectangle(w1,w2,w3,w4);

line(x1,y1,x2,y2);

line(x2,y2,x3,y3);

line(x3,y3,x1,y1);

getch();

v1=425;

v2=75;

v3=550;

v4=250;

sx=(float)(v3-v1)/(w3-w1);

sy=(float)(v4-v2)/(w4-w2);

rectangle(v1,v2,v3,v4);

x1=v1+floor(((float)(x1-w1)\*sx)+.5);

x2=v1+floor(((float)(x2-w1)\*sx)+.5);

x3=v1+floor(((float)(x3-w1)\*sx)+.5);

y1=v2+floor(((float)(y1-w2)\*sy)+.5);

y2=v2+floor(((float)(y2-w2)\*sy)+.5);

y3=v2+floor(((float)(y3-w2)\*sy)+.5);

line(x1,y1,x2,y2);

line(x2,y2,x3,y3);

line(x3,y3,x1,y1);

getch();

return 0; }

**OUTPUT**

Enter The Coordinate x1,y1,x2,y2,x3,y3

100 200 300 400 500 350



//**Ex. No 5 – 2D Composite Transformation**

//**Consecutive Translation**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void draw2d(int x[],int y[],int fs)

{

int i;

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

{

if(i!=(fs-1))

line(x[i],y[i],x[i+1],y[i+1]);

else

line(x[i],y[i],x[0],y[0]);

}

}

void main()

{

int gd=DETECT,gm,fs,x[10],y[10],tx,ty,ax,ay,i;

initgraph(&gd,&gm,"Z:\\TC\\BGI");

printf("Enter the number of sides: ");

scanf("%d",&fs);

printf("Enter the points\n");

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

{

printf("x[%d] y[%d]:",i,i);

scanf("%d%d",&x[i],&y[i]);

}

draw2d(x,y,fs);

printf("Enter the 1st translation:(tx & ty) ");

scanf("%d%d",&tx,&ty);

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

{

x[i]+=tx;

y[i]+=ty;

}

draw2d(x,y,fs);

printf("Enter the second translation:(ax & ay) ");

scanf("%d%d",&ax,&ay);

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

{

x[i]+=ax;

y[i]+=ay;

}

draw2d(x,y,fs);

getch();

}

OUTPUT:

Consecutive translation:

Enter the number of sides:3

Enter the corner points:

x0 y0 : 200 200

x1 y1 : 150 250

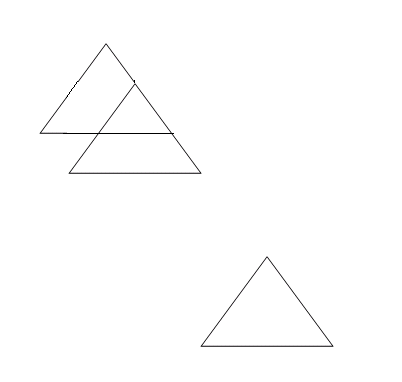
x2 y2 : 250 250

Enter the 1st translation: (tx and ty) 25 25

Enter the 2nd translation: (tx and ty) 100 100

Before translation:

After translation:



RESULT:

//**Consecutive Rotation**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<stdlib.h>

#include<math.h>

void draw2d(float fs,float x[],float y[])

{

int i;

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

{

if(i!=(fs-1))

line(x[i],y[i],x[i+1],y[i+1]);

else

line(x[i],y[i],x[0],y[0]);

}

}

void main()

{

int i,gd=DETECT,gm;

float x[20],y[20],a[20],b[20],tx=1,ty,fs;

initgraph(&gd,&gm,"Z:\\TC\\BGI");

printf("Enter the number of sides:");

scanf("%f",&fs);

printf("Enter the corner points:\n");

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

{

printf("x%d y%d: ",i,i);

scanf("%f%f",&x[i],&y[i]);

}

draw2d(fs,x,y);

printf("Enter the first rotation angle(tx): ");

scanf("%f",&tx);

printf("Enter the second rotation angle: ");

scanf("%f",&ty);

tx+=ty;

ty=tx\*M\_PI/180;

printf("The radian is %f",ty);

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

{

a[i]=x[i]\*cos(ty)-y[i]\*sin(ty);

b[i]=x[i]\*sin(ty)+y[i]\*cos(ty);

}

draw2d(fs,a,b);

getch();

}

OUTPUT:

Consecutive rotation:

Enter the number of sides:3

Enter the corner points:

x0 y0 : 200 200

x1 y1 : 150 250

x2 y2 : 250 250

Enter the first rotation angle(tx) : 10

Enter the second rotation angle : 5

The radian is : 0.261799

Before rotation:

After rotation:

RESULT:

//**Pivot Point Rotation**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<stdlib.h>

#include<math.h>

void draw2d(float fs,float x[],float y[])

{

int i;

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

{

if(i!=(fs-1))

line(x[i],y[i],x[i+1],y[i+1]);

else

line(x[i],y[i],x[0],y[0]);

}

}

void main()

{

int i,gd=DETECT,gm;

float x[20],y[20],a[20],b[20],tx=1,ty,fs,ax,ay;

initgraph(&gd,&gm,"Z:\\TC\\BGI");

printf("Enter the number of sides:");

scanf("%f",&fs);

printf("Enter the corner points:\n");

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

{

printf("x%d y%d: ",i,i);

scanf("%f%f",&x[i],&y[i]);

}

draw2d(fs,x,y);

printf("Enter the pivot point:(ax & ay) ");

scanf("%f%f",&ax,&ay);

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

{

x[i]+=ax;

y[i]+=ay;

}

draw2d(fs,x,y);

printf("Enter the rotation angle(tx): ");

scanf("%f",&tx);

ty=tx\*M\_PI/180;

printf("The radian is %f",ty);

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

{

a[i]=x[i]\*cos(ty)-y[i]\*sin(ty);

b[i]=x[i]\*sin(ty)+y[i]\*cos(ty);

}

draw2d(fs,a,b);

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

{

a[i]-=ax;

b[i]-=ay;

}

draw2d(fs,a,b);

getch();

}

OUTPUT:

Enter the number of sides:3

Enter the corner points:

x0 y0 : 100 100

x1 y1 : 150 100

x2 y2 : 125 125

Enter the pivot point: (ax and ay) 100 100

Enter the rotation angle(tx): 10

The radian is 0.174533

RESULT:

//**Consecutive Scaling**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<stdlib.h>

void draw2d(int fs,int x[],int y[],int tx,int ty)

{

int i;

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

{

if(i!=(fs-1))

line(x[i]\*tx,y[i]\*ty,x[i+1]\*tx,y[i+1]\*ty);

else

line(x[i]\*tx,y[i]\*ty,x[0]\*tx,y[0]\*ty);

}

}

void main()

{

int i,gd=DETECT,gm,x[20],y[20],tx=1,ty=1,ax[10],ay[10],fs,n;

initgraph(&gd,&gm,"Z:\\TC\\BGI");

printf("Enter the number of sides:");

scanf("%d",&fs);

printf("Enter the corner points:\n");

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

{

printf("x%d y%d: ",i,i);

scanf("%d%d",&x[i],&y[i]);

}

printf("Enter the number of scaling transformation: ");

scanf("%d",&n);

draw2d(fs,x,y,tx,ty);

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

{

printf("Enter the scaling unit%d(tx and ty): ",i);

scanf("%d%d",&ax[i],&ay[i]);

}

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

{

tx\*=ax[i];

ty\*=ay[i];

}

draw2d(fs,x,y,tx,ty);

getch();

}

OUTPUT:

Consecutive scaling:

Enter the number of sides:3

Enter the corner points:

x0 y0 : 200 200

x1 y1 : 150 250

x2 y2 : 250 250

Enter the number of scaling transformation: 2

Enter the scaling unit1(tx and ty): 2 2

Enter the scaling unit2(tx and ty): 2 2

Before scaling:

After scaling:

RESULT:

//**Ex. No 6 – Cohen Sutherland Line Clipping Algorithm**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#define LEFT 0x1

#define RIGHT 0x2

#define BOTTOM 0x4

#define TOP 0x8

float wxmin,wymin,wxmax,wymax;

struct line

{

float x1,y1,x2,y2,done,draw;

unsigned char code1,code2;

float m;

}s[20];

unsigned char encode(float x,float y)

{

unsigned char code=0x00;

if(x<wxmin)

code=code | LEFT;

else if(x>wxmax)

code=code | RIGHT;

if(y<wymin)

code=code | TOP;

else if(y>wymax)

code=code | BOTTOM;

return (code);

}

void process(struct line \*temp)

{

float t;

unsigned char ct;

if(temp->x1 != temp->x2)

temp->m=(float)(temp->y2-temp->y1)/(temp->x2-temp->x1);

temp->code1=encode(temp->x1,temp->y1);

temp->code2=encode(temp->x2,temp->y2);

if(!(temp->code1 | temp->code2))

{

temp->done=1;

}

else if(temp->code1 & temp->code2)

{

temp->done=1;

temp->draw=0;

}

else

{

if(!temp->code1)

{

t=temp->x1;

temp->x1=temp->x2;

temp->x2=t;

t=temp->y1;

temp->y1=temp->y2;

temp->y2=t;

ct=temp->code1;

temp->code1=temp->code2;

temp->code2=ct;

}

if(temp->code1 & LEFT)

{

temp->y1+=(wxmin-temp->x1)\*temp->m;

temp->x1=wxmin;

}

else if(temp->code1 & RIGHT)

{

temp->y1+=(wxmax-temp->x1)\*temp->m;

temp->x1=wxmax;

}

else if(temp->code1 & TOP)

{

if(temp->x1 != temp->x2)

temp->x1+=(wymin-temp->y1)/temp->m;

temp->y1=wymin;

}

else if(temp->code1 & BOTTOM)

{

if(temp->x1 != temp->x2)

temp->x1+=(wymax-temp->y1)/temp->m;

temp->y1=wymax;

}

}

}

void main()

{

int i,j,gd=DETECT,gm,n;

printf("Enter the window details(xmin,xmax,ymin,ymax): ");

scanf("%f%f%f%f",&wxmin,&wxmax,&wymin,&wymax);

printf("Enter the number of lines: ");

scanf("%d",&n);

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

{

printf("Enter the end points(x1,y1,x2,y2) of line%d: ",i+1);

scanf("%f%f%f%f",&s[i].x1,&s[i].y1,&s[i].x2,&s[i].y2);

s[i].code1=0x00;

s[i].code2=0x00;

s[i].draw=1;

process(&s[i]);

}

initgraph(&gd,&gm,"Z:\\TC\\BGI");

line(wxmin,wymin,wxmin,wymax);

line(wxmin,wymin,wxmax,wymin);

line(wxmin,wymax,wxmax,wymax);

line(wxmax,wymin,wxmax,wymax);

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

{

printf("The slope is %f....x1 = %f, y1 = %f, x2 = %f, y2 = %f", s[i].m, s[i].x1, s[i].y1, s[i].x2, s[i].y2);

printf("\nThe clipped line is ");

if(s[i].draw)

line((int)s[i].x1,(int)s[i].y1,(int)s[i].x2,(int)s[i].y2);

}

getch();

}

Output

Enter the window details(xmin,xmax,ymin,ymax) : 50 50 200 200

Enter the number of lines: 1

Enter the end points(x1,y1,x2,y2) of line%d: 25 25 225 225

The Clipped line is

Result

**//EX. NO. 8 3D Transformation – Translation, Rotation & Scaling**

#include<stdio.h>  
#include<conio.h>  
#include<graphics.h>  
#include<math.h>  
int maxx,maxy,midx,midy;  
void axis()  
{  
  getch();  
  cleardevice();  
  line(midx,0,midx,maxy);  
  line(0,midy,maxx,midy);  
}  
void main()  
{  
  int gd,gm,x,y,z,ang,x1,x2,y1,y2;  
  detectgraph(&gd,&gm);  
  initgraph(&gd,&gm,"C:/TC/BGI");  
  setfillstyle(3,25);  
  maxx=getmaxx();  
  maxy=getmaxy();  
  midx=maxx/2;  
  midy=maxy/2;  
  outtextxy(100,100,"ORIGINAL OBJECT");  
  line(midx,0,midx,maxy);  
  line(0,midy,maxx,midy);  
  bar3d(midx+100,midy-20,midx+60,midy-90,20,5);  
  axis();  
  outtextxy(100,20,"TRANSLATION");  
  printf("\n\n Enter the Translation vector: ");  
  scanf("%d%d",&x,&y);  
  bar3d(midx+100,midy-20,midx+60,midy-90,20,5);  
  bar3d(midx+(x+100),midy-(y+20),midx+(x+60),midy-(y+90),20,5);  
  axis();  
  outtextxy(100,20,"SCALING");  
  printf("\n Enter the Scaling Factor: ");  
  scanf("%d%d%d",&x,&y,&z);  
  bar3d(midx+100,midy-20,midx+60,midy-90,20,5);  
  bar3d(midx+(x\*100),midy-(y\*20),midx+(x\*60),midy-(y\*90),20\*z,5);  
  axis();  
  outtextxy(100,20,"ROTATION");  
  printf("\n Enter the Rotation angle: ");  
  scanf("%d",&ang);  
  x1=100\*cos(ang\*3.14/180)-20\*sin(ang\*3.14/180);  
  y1=100\*sin(ang\*3.14/180)+20\*sin(ang\*3.14/180);  
  x2=60\*cos(ang\*3.14/180)-90\*sin(ang\*3.14/180);  
  y2=60\*sin(ang\*3.14/180)+90\*sin(ang\*3.14/180);  
  axis();  
  printf("\n After rotating about z-axis\n");  
  bar3d(midx+100,midy-20,midx+60,midy-90,20,5);  
  bar3d(midx+x1,midy-y1,midx+x2,midy-y2,20,5);  
  axis();  
  printf("\n After rotating about x-axis\n");  
  bar3d(midx+100,midy-20,midx+60,midy-90,20,5);  
  bar3d(midx+100,midy-x1,midx+60,midy-x2,20,5);  
  axis();  
  printf("\n After rotating about y-axis\n");  
  bar3d(midx+100,midy-20,midx+60,midy-90,20,5);  
  bar3d(midx+x1,midy-20,midx+x2,midy-90,20,5);  
  axis();  
  closegraph();  
}

**Input :**

Enter the translation vector : 65 70

Enter the scaling factor : 2 2 2

Enter the Rotation Angle : 100

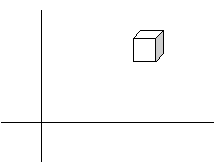
**OUTPUT**

**Translation**

Enter Translation Factor : 50 60 70

Before Translation

After Translation



**Scaling**

Enter Scaling Factor : 80 90 95

After Scaling

Rotation

Enter Rotating Angle : 60

After Rotation about Z-Axis

After Rotation about X-Axis

After Rotation about Y-Axis :

**RESULT:**

## **Program For 3D Projections:**

## #include "stdio.h"

## #include "stdlib.h"

## #include"graphics.h"

## #include"conio.h"

## void draw3d(int s,int x[20],int y[20],int d);

## void main()

## {

## int gd=DETECT,gm;

## int x[20],y[20],i,s,d;

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

## printf("Enter the No of sides : ");

## scanf("%d",&s);

## for(i=0;i< s;i++)

## {

## printf("(x%d,y%d) :",i,i);

## scanf("%d%d",&x[i],&y[i]);

## }

## printf("Depth :");

## scanf("%d",&d);

## draw3d(s,x,y,d);

## getch();

## setcolor(14);

## for(i=0;i< s-1;i++)

## {

## line(x[i]+200,y[i],x[i+1]+200,y[i+1]);

## }

## line(x[i]+200,y[i],x[0]+200,y[0]);

## getch();//top view

## for(i=0;i< s-1;i++)

## {

## line(x[i],300,x[i+1],300);

## line(x[i],300+d\*2,x[i+1],300+d\*2);

## line(x[i],300,x[i],300+d\*2);

## line(x[i+1],300,x[i+1],300+d\*2);

## }

## getch();//side view

## for(i=0;i< s-1;i++)

## {

## line(10,y[i],10,y[i+1]);

## line(10+d\*2,y[i],10+d\*2,y[i+1]);

## line(10,y[i],10+d\*2,y[i]);

## line(10,y[i+1],10+d\*2,y[i+1]);

## }

## getch();

## closegraph();

## }

## void draw3d(int s,int x[20],int y[20],int d)

## {

## int i,j,k=0;

## for(j=0;j< 2;j++)

## {

## for(i=0;i< s-1;i++)

## line(x[i]+k,y[i]-k,x[i+1]+k,y[i+1]-k);

## line(x[i]+k,y[i]-k,x[0]+k,y[0]-k);

## k=d;

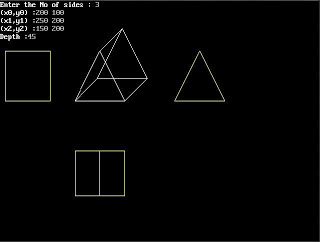
## }

## for(i=0;i< s;i++)

## line(x[i],y[i],x[i]+d,y[i]-d);

## }

Output:

[](http://3.bp.blogspot.com/__0zIL__Aayc/S2LkohXEt-I/AAAAAAAABBg/5WxhzGRLpwE/s1600-h/proj.JPG)

## OpenGL to Work with Visual C++

### Installation

### 1. Install Visual C++ 2008 Express Edition (To support OpenGL).

2. Copy all the **.h** files into the C:\Program Files\Microsoft SDKs\Windows\v6.1\Include\GL folder.

***Header Files (.h files) :*** *Gl.h, glu.h, glut.h, freeglut.h, freeglut\_ext.h , freeglut\_std.h*

3. Copy all the **.lib** files into the C:\Program Files\Microsoft SDKs\Windows\v6.1\Lib folder.

***Library files (.lib files) :*** *opengl32.lib, glut32.lib, glu32.lib*

4. Copy all the **.dll** files into the C:\Windows\system32 folder.

***Dynamic Link Library Files (.dll) :***  *freeglut.dll* , *glut32.dll*

**Working with Console Application Program in OpenGL**

### 1. Creating a Project

1. Start Visual C++ and Under the File menu select New → Project (Ctrl+Shift+N).
2. Select project types as Win32 and Win32 Console Application.
3. Give a User name for the Project.
4. Add a GLUT program to the project in the window.

**2. Linking OpenGL Libraries**

1. Under the Project menu select Project Properties (Alt+F7) at the bottom.
2. Select Configuration Properties → Select “C/C++” → “Preprocessor” → In preprocessor definitions additionally include the path where gl/glut.h is present.

Example : C:\Program Files\Microsoft \ SDKs \Windows \v6.0A \Include

1. Select "Linker" folder and click "Input" .
2. Select All Configurations from the Configuration drop-down box at the top of the dialog. This ensures you are changing the settings for both the Debug and Release configurations.
3. Select "Additional Dependencies" and type the following contents:

*opengl32.lib glut32.lib glu32.lib*

#### 3. Compiling the Application

Choose "Build" from the menu options.

Select "Build filename".

#### 4. Run the Application

Choose "Debug" from the menu options.

Select "Start without Debugging".

1. **Save the Project**

Select “File Menu” → select Save all (Ctrl+Shift+S).

Save all the documents before quitting.

**Working with Windows Application Program in OpenGL**

### 1. Creating a Project

1. Start Visual C++ and Under the File menu select New → Project (Ctrl+Shift+N).
2. Select Win32 Project, enter a Username, and click OK.
3. In the Wizard click Next, then in Additional options check the box of Empty Project, and click Finish.
4. Under the Project menu select Add New Item (Ctrl+Shift+A).
5. Select C++ File (.cpp), enter a Name, and click OK.
6. Add a GLUT program to the project in the window.

**2. Link to the OpenGL libraries:**

1. Under the Project menu select Project Properties (Alt+F7) at the bottom.
2. Select Configuration Properties → Linker → Input from the navigation panel on the left.
3. Select All Configurations from the Configuration drop-down box at the top of the dialog.
4. Type “*opengl32.lib glut32.lib glu32.lib*” in Additional Dependencies and click OK.

#### Compiling the Application

Choose "Build" from the menu options.

Select "Build filename".

#### Run the Application

Choose "Debug" from the menu options.

Select "Start Without Debugging".

**5. Save the Project**

Select “File Menu” → select Save all (Ctrl+Shift+S).

Save all the documents before quitting.

**//Ex. No 10 Basic OpenGL Graphics Primitives**

#include <gl/glut.h>

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT); // Clear the color buffer

glBegin(GL\_QUADS); // Each set of 4 vertices form a quad

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

glVertex2f(-0.7f, -0.1f); // x, y

glVertex2f(-0.1f, -0.1f);

glVertex2f(-0.1f, 0.5f);

glVertex2f(-0.7f, 0.5f);

glEnd();

glBegin(GL\_TRIANGLES); // Each set of 3 vertices form a triangle

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

glVertex2f(0.2f, -0.3f);

glVertex2f(0.8f, -0.3f);

glVertex2f(0.5f, 0.2f);

glEnd();

glBegin(GL\_POLYGON); // The vertices form one closed polygon

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

glVertex2f(0.2f, 0.3f);

glVertex2f(0.4f, 0.3f);

glVertex2f(0.5f, 0.5f);

glVertex2f(0.4f, 0.7f);

glVertex2f(0.2f, 0.7f);

glVertex2f(0.1f, 0.5f);

glEnd();

glFlush(); // Render now

}

// GLUT runs as a Console Application

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

glutInit(&argc, argv); // Initialize GLUT

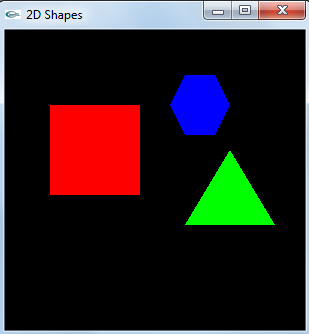
glutCreateWindow("2D Shapes"); // Create a window with the given title

glutDisplayFunc(display); // Register callback handler for window re-paint

glutMainLoop(); // Enter infinitely event-processing loop

return 0;

}



**//Ex No. 11 Drawing three dimensional objects and Scenes**

#include <gl/glut.h>

#include <stdlib.h>

void init (void)

{

GLfloat light\_ambient[] = { 0.0, 0.0, 0.0, 1.0 };

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

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

GLfloat light\_position[] = { 1.0, 1.0, 1.0, 0.0 };

glLightfv (GL\_LIGHT0, GL\_AMBIENT, light\_ambient);

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

glLightfv (GL\_LIGHT0, GL\_SPECULAR, light\_specular);

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

glEnable (GL\_LIGHTING);

glEnable (GL\_LIGHT0);

glEnable(GL\_DEPTH\_TEST);

}

void display (void)

{

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

glPushMatrix ();

glRotatef (20.0, 1.0, 0.0, 0.0);

glPushMatrix ();

glTranslatef (-0.75, 0.5, 0.0);

glRotatef (90.0, 1.0, 0.0, 0.0);

glutSolidTorus (0.275, 0.85, 15, 15);

glPopMatrix ();

glPushMatrix ();

glTranslatef (-0.75, -0.5, 0.0);

glRotatef (270.0, 1.0, 0.0, 0.0);

glutSolidCone (1.0, 2.0, 15, 15);

glPopMatrix ();

glPushMatrix ();

glTranslatef (0.75, 0.0, -1.0);

glutSolidSphere (1.0, 15, 15);

glPopMatrix (); glPopMatrix ();

glFlush ();

}

void reshape(int w, int h)

{

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

glMatrixMode (GL\_PROJECTION);

glLoadIdentity ();

if (w <= h)

glOrtho (-2.5, 2.5, -2.5\*(GLfloat)h/(GLfloat)w,

2.5\*(GLfloat)h/(GLfloat)w, -10.0, 10.0);

else

glOrtho (-2.5\*(GLfloat)w/(GLfloat)h,

2.5\*(GLfloat)w/(GLfloat)h, -2.5, 2.5, -10.0, 10.0);

glMatrixMode (GL\_MODELVIEW);

glLoadIdentity ();

}

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

{

glutInit(&argc, argv);

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

glutInitWindowSize (500, 500);

glutCreateWindow (argv[0]);

init ();

glutReshapeFunc (reshape);

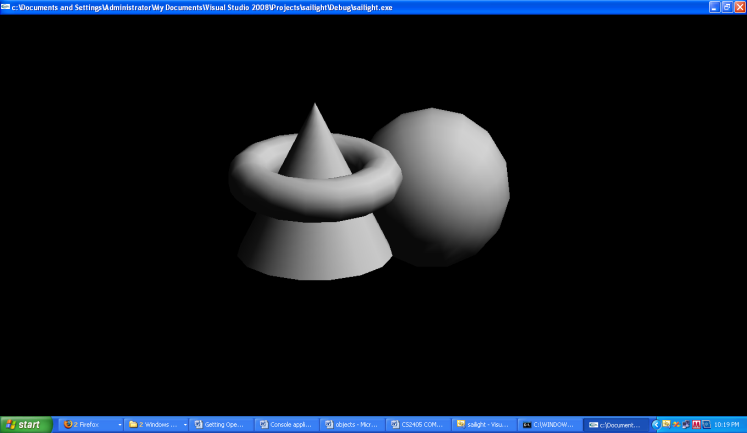
glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**Output**



**Result**

**//Ex. No 12. Sierpinski Gasket**

#include <stdlib.h>

#include <GL/glut.h>

typedef float point[3];

/\* initial tetrahedron \*/

point v[]={{0.0, 0.0, 1.0}, {0.0, 0.942809, -0.33333},

{-0.816497, -0.471405, -0.333333}, {0.816497, -0.471405, -0.333333}};

static GLfloat theta[] = {0.0,0.0,0.0};

int n;

void triangle( point a, point b, point c)

/\* display one triangle using a line loop for wire frame, a single

normal for constant shading, or three normals for interpolative shading \*/

{

glBegin(GL\_POLYGON);

glNormal3fv(a);

glVertex3fv(a);

glVertex3fv(b);

glVertex3fv(c);

glEnd();

}

void divide\_triangle(point a, point b, point c, int m)

{

/\* triangle subdivision using vertex numbers

righthand rule applied to create outward pointing faces \*/

point v1, v2, v3;

int j;

if(m>0)

{

for(j=0; j<3; j++) v1[j]=(a[j]+b[j])/2;

for(j=0; j<3; j++) v2[j]=(a[j]+c[j])/2;

for(j=0; j<3; j++) v3[j]=(b[j]+c[j])/2;

divide\_triangle(a, v1, v2, m-1);

divide\_triangle(c, v2, v3, m-1);

divide\_triangle(b, v3, v1, m-1);

}

else(triangle(a,b,c)); /\* draw triangle at end of recursion \*/

}

void tetrahedron( int m)

{

/\* Apply triangle subdivision to faces of tetrahedron \*/

glColor3f(1.0,0.0,0.0);

divide\_triangle(v[0], v[1], v[2], m);

glColor3f(0.0,1.0,0.0);

divide\_triangle(v[3], v[2], v[1], m);

glColor3f(0.0,0.0,1.0);

divide\_triangle(v[0], v[3], v[1], m);

glColor3f(0.0,0.0,0.0);

divide\_triangle(v[0], v[2], v[3], m);

}

void display(void)

{

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

glLoadIdentity();

tetrahedron(n);

glFlush();

}

void myReshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

glOrtho(-2.0, 2.0, -2.0 \* (GLfloat) h / (GLfloat) w,

2.0 \* (GLfloat) h / (GLfloat) w, -10.0, 10.0);

else

glOrtho(-2.0 \* (GLfloat) w / (GLfloat) h,

2.0 \* (GLfloat) w / (GLfloat) h, -2.0, 2.0, -10.0, 10.0);

glMatrixMode(GL\_MODELVIEW);

glutPostRedisplay();

}

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

{

n=4;

glutInit(&argc, argv);

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

glutInitWindowSize(500, 500);

glutCreateWindow("3D Gasket");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

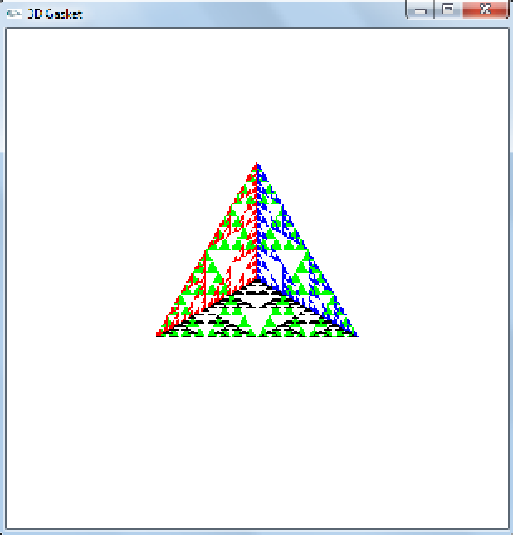
glEnable(GL\_DEPTH\_TEST);

glClearColor (1.0, 1.0, 1.0, 1.0);

glutMainLoop();

}

**Output**



**Result**

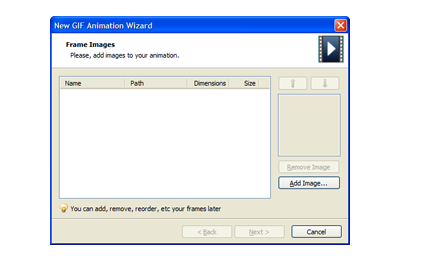
# Creating Animation Using GIF Animator

# Creating Your First Animation

This tutorial will help you to learn how to create GIF animation from individual pictures (frames). To learn GIF animation basics

After you have created or got somewhere else two or more subsequent frames, it is time to put them together.

#### Step 1. Start Animation Wizard.

* Select the **File -> Wizard...** menu item to start Animation Wizard. Animation Wizard dialog will appear.   
    
  

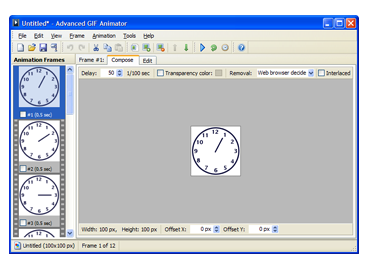
#### Step 2. Add Images.

* Click the **Add Image...** button to start adding images to your animation. Open dialog will appear.
* Navigate to the ***My Documents\GIF Animations\Samples\Clock*** folder.
* Select images *clock1.gif, clock2.gif, ... ,clock12.gif* and click **Open**.
* Repeat previous operation to add more images if necessary.
* Rearrange your images in the right order using and buttons if necessary.
* Click **Next >**

#### Step 3. Set Animation Parameters.

* Select **Top left corner of the frame** option and click **Next >**.
* Set frame delay value to 50 and loop count to "Play animation infinitely". Click **Finish**

Animation Wizard will create animation frames for you and load it into GIF Animator



#### Step 4. Preview Animation.

* To preview the animation, click the **Animate** toolbar button, or **Animation -> Animate** menu item.  
  
* To stop the animation, click the **Stop** toolbar button, or select **Animation -> Stop** menu item.  
  

Want the clock ticking faster?

* Click the **Frame Delay** toolbar button, or select **Animation -> Delay...** menu item.  
    
  The Frame Delay dialog will appear.
* Set **Delay** value to 25
* Select **Apply to all frames** option and click OK

#### Step 5. Save Animation.

* To save the animation, click the **Save** toolbar button, or select **File -> Save** menu item.  
  

# **PhotoPad Photo Editing Software**

PhotoPad **picture editing software** is an easy digital photo editor.

* Easily edit digital photos and other pictures
* Supports all popular image formats
* Crop, rotate, resize and flip photos fast

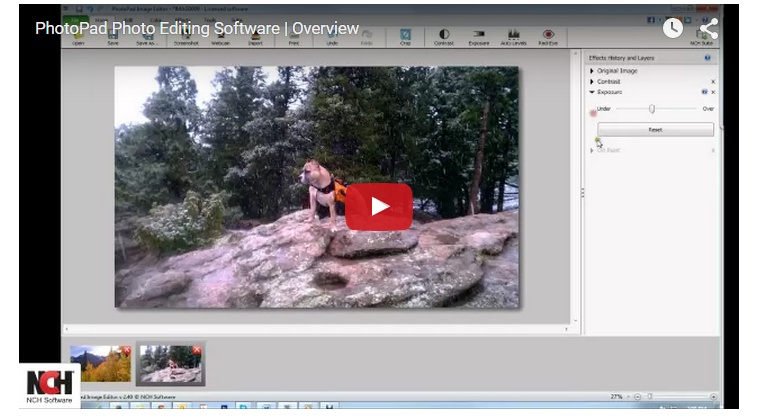
PhotoPad is designed to be ready to open and edit your photos quickly.

|  |  |
| --- | --- |
|  |  |

Whether you’re a professional photographer or just want to edit your personal photos, PhotoPad offers the powerful tools and features you need to improve your digital images.[Download PhotoPad](http://www.nchsoftware.com/photoeditor/ppadsetup.exe) and start enhancing your images today.

##### Photo Editing Features

* Crop, rotate, resize and flip pictures
* Touch up photos to remove red-eye and blemishes
* Apply photo effects, including oil paint, cartoon, vignette, sepia and many more
* Improve photo quality and focus with blur, sharpening and noise reduction tools
* Adjust image color/hue, saturation, brightness and contrast
* Create collages and photo mosaics with your photographs
* Convert your photo to cross stitch patterns, paint by numbers or add an oil painting effect
* Photo stitching to make your own panorama images
* Load jpg, gif, png and other popular image formats
* Add text and captions to photos
* Insert clipart from the included clipart library
* Add frames and borders around your photographs
* Change the aspect ratio of an image without distorting key features using the liquid resize effect
* Non-destructive editing allows easy modifications of individual photo effects on the layers list
* Upload edited pictures directly to Facebook or Flickr





# Blender Introduction

**Why Blender?**

Blender was first conceived in December 1993 and born as a usable product in August 1994 as an integrated application that enables the creation of a diverse range of 2D and 3D content. Blender provides a broad spectrum of modeling, texturing, lighting, animation and video post-processing functionality in one package. Through its open architecture, Blender provides cross-platform interoperability, extensibility, an incredibly small footprint, and a tightly integrated workflow. Blender is one of the most popular Open Source 3D graphics applications in the world.

Aimed at media professionals and artists world-wide, Blender can be used to create 3D visualizations, stills as well as broadcast and cinema quality videos, while the incorporation of a real-time 3D engine allows for the creation of 3D interactive content for stand-alone playback.

Key Features:

Animation is making an object move or change shape over time. Objects can be animated in many ways:

**Moving as a whole object**

Changing their position, orientation or size in time;

**Deforming them**

Animating their vertices or control points;

**Inherited animation**

Causing the object to move based on the movement of another object (e.g. its parent, hook, armature, etc...).

In this chapter we will cover the first two, but the basics given here are actually vital for understanding the following chapters as well.

Animation is typically achieved with the use of [Key Frames](http://www.blender.org/manual/animation/basics/key_frames.html).

# Key Frames

A Key Frame is simply a marker in time which stores the value of a property.

For example, a key frame might indicate that the horizontal position of a cube is at 3m on frame 1.

The purpose of a key frame is to allow for interpolated animation, meaning, for example, that the user could then add another key on frame 10, specifying the cube’s horizontal position at 20m, and Blender will automatically determine the correct position of the cube for all the frames between frame 1 and 10 depending on the chosen interpolation method (e.g. linear, bezier, quadratic, etc...).

## Adding Key Frames

There are several methods of adding new keys. Namely:

* In the 3D View, pressing I will bring up a menu to choose what to add a key frame to.
* Hovering over a property and pressing I.
* RMB a value and choose Insert Keyframe from the menu.

## Removing Key Frames

There are several methods of removing key frames

* In the 3D View press Alt-I to remove keys on the current frame for selected objects.
* When the mouse is over a value press Alt-I.
* RMB a value and choose Delete Keyframe from the menu.

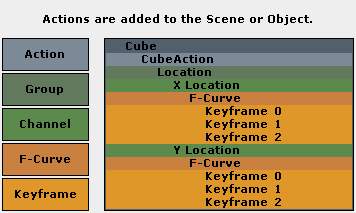
## Editing Key Frames

For editing key frames go to the [Graph Editor](http://www.blender.org/manual/animation/editors/graph.html) or to the [Dopesheet](http://www.blender.org/manual/animation/editors/dopesheet.html)

# Actions

## Actions

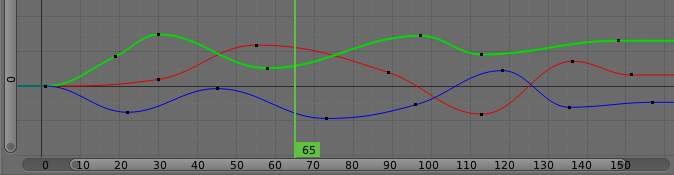
When animating objects and properties in blender, Actions record and contain the data.



Actions.

So when you animate an object by changing its location with keyframes, the animation is saved to the Action.

Each property has a channel which it is recorded to, for example, Cube.location.x is recorded to Channel X Location.



Graph Editor. Each Channel has an F-Curve represented by the lines between the keyframes.

**Actions**

Record and contain animation data.

**Groups**

Are groups of channels.

**Channels**

Record properties.

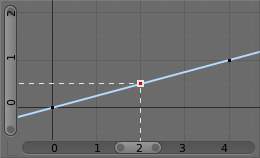
**F-Curves**

Are used to interpolate the difference between the keyframes.

**Keyframes**

Are used to set the values of properties.

## F-Curve Interpolation



Graph Editor: Channel F-Curve.

The keyframes are set values by the user.

The F-Curve is used to interpolate the difference between the keyframes.

The F-Curve has different types of interpolation and also [F-Curve Modifiers](http://www.blender.org/manual/animation/editors/graph/fmodifiers.html).

Most the settings for the [F-Curve](http://www.blender.org/manual/animation/editors/graph/fcurves.html) are found in the [Graph Editor](http://www.blender.org/manual/animation/editors/graph.html).

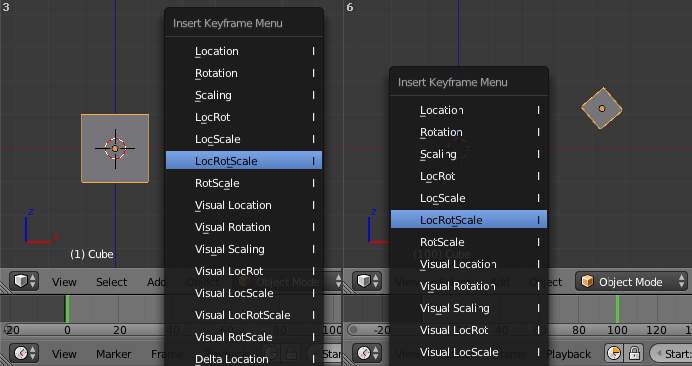
## Basic Animation

These are some common ways to animate objects. These methods can be used on different objects, like armature bones in pose mode.

### Insert Keyframes

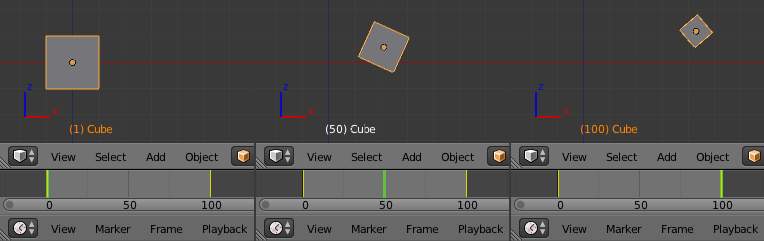
This example shows you how to animate a cubes location, rotation, and scale.

* First, in the Timeline, or other animation editors, set the frame to 1.
* With the Cube selected in Object Mode, press I in the 3D View.
* From the Insert Keyframe Menu select LocRotScale.
* This will record the location, rotation, and scale, for the Cube on frame 1.
* Set the frame to 100.
* Use Grab/Move G, Rotate R, Scale S, to transform the cube.
* Press I in the 3D View. From the Insert Keyframe Menu select LocRotScale.

[](http://www.blender.org/manual/_images/Actions_Insert_Keyframe_00.jpg)

Insert Keyframes.

To test the animation, press Alt-A to play.

[](http://www.blender.org/manual/_images/Actions_Insert_Keyframe_01.jpg)

The animation on frames 1, 50, 100.

### Auto Keyframe

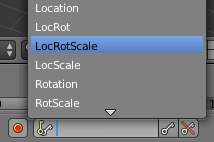


Timeline Auto Keyframe.

Auto Keyframe is the red record button in the Timeline header. Auto Keyframe adds keyframes automatically to the set frame if the value for transform type properties changes.

See [Timeline V Keyframe Control](http://www.blender.org/manual/animation/editors/timeline.html#animation-editors-timeline-autokeyframe) for more info.

### Keying Sets



Timeline Keying Sets.

Keying Sets are a set of keyframe channels. They are used to record multiple properties at the same time. There are some built in keying sets, ‘LocRotScale’, and also custom keying sets can be made.

To use the keying set, first select a keying set from the Timeline header, or the Keying Sets Panel.

Now when you press I in the 3D view, blender will add keyframes for all the properties in the active keying set.

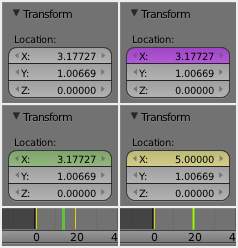
See [Keying Sets](http://www.blender.org/manual/animation/basics/keying_sets.html) for more info.

### Properties



Keyframe properties.

Keyframes can be used to animate lots of different properties in blender. To add keyframes to a property in the UI, RMB the property, then select Insert Single Keyframe, or Insert Keyframes. Insert Keyframes I will add a keyframes for the set of properties.



Properties, Drivers, Keyframes.

Properties have different colors and menu items for different states.

**Gray - Property is not animated with Keyframes or Drivers.**

Insert Keyframes I. Insert Single Keyframe. Add Drivers. Add Single Driver. Paste Driver.

**Purple - Property value is controlled with a Driver.**

Delete Drivers. Delete Single Driver. Copy Driver. Paste Driver.

**Green - Property has Channel with Keyframes.**

Insert Keyframes I. Insert Single Keyframe. Clear Keyframes Alt-Shift-I Clear Single Keyframes.

**Yellow - Property has Keyframes on the current Frame.**

Replace Keyframes I. Replace Single Keyframe. Delete Keyframes Alt-I. Delete Single Keyframe. Clear Keyframes Alt-Shift-I Clear Single Keyframes.

**Each property also has some Keying Set options.**

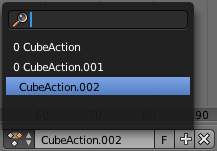
Add All to Keying Set K. Add Single to Keying Set. Remove from Keying Set.

### Editing

**3D View.**

Insert Keyframes on current frame I Delete Keyframes on current frame Alt-I

## Working with Actions



Action Browser.

When you first animate an object by adding keyframes, blender creates an Action to record the data.

Actions can be managed with the Action Browser in the [DopeSheet](http://www.blender.org/manual/animation/editors/dopesheet.html) Action Editor header, or the properties region of the [NLA Editor](http://www.blender.org/manual/animation/editors/nla.html).

If you are making multiple actions for the same object, press the **F** button for each action, this will give the actions a Fake User and will make blender save the unlinked actions.

Objects can only use one Action at a time for editing, the [NLA Editor](http://www.blender.org/manual/animation/editors/nla.html) is used to blend mutiple actions together.