Computer Graphics

Section 5:

2D - Transformation

Prepared By:

Eng. Rabie Mohammed Masoud

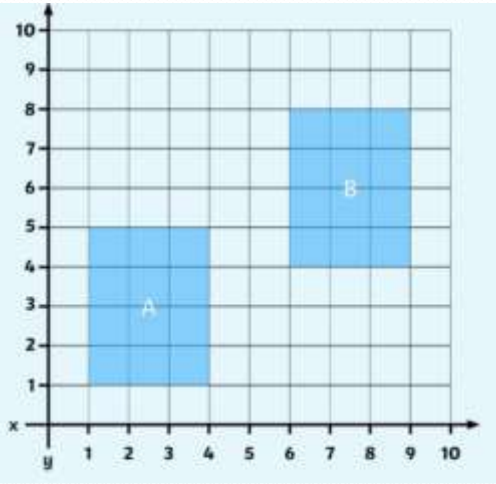
Supervised By: Dr. Yasser Waziri

ICT Department
Korean Egyptian Faculty for Industry and Energy Technology

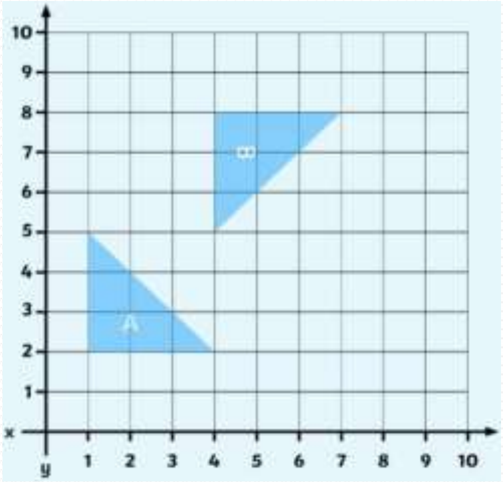
transformations

What is transformations?

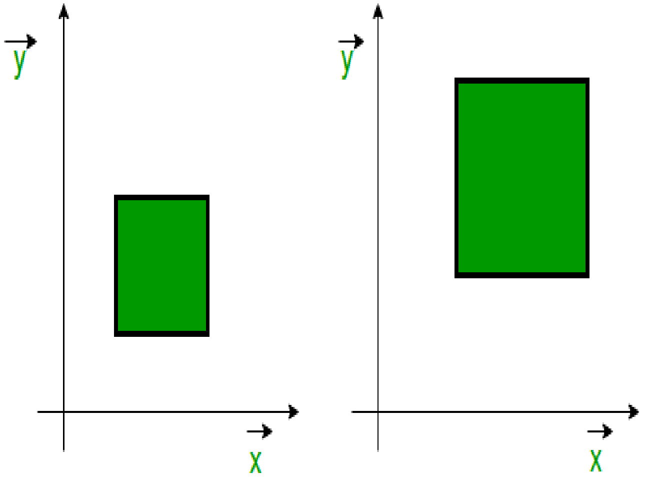
The geometrical changes of an object from a current state to modified state



Translation



Rotation



Scaling

Translation

Translation

A translation moves all points in an object along the same straight-line path to new positions.

The path is represented by a vector, called the translation or shift vector.

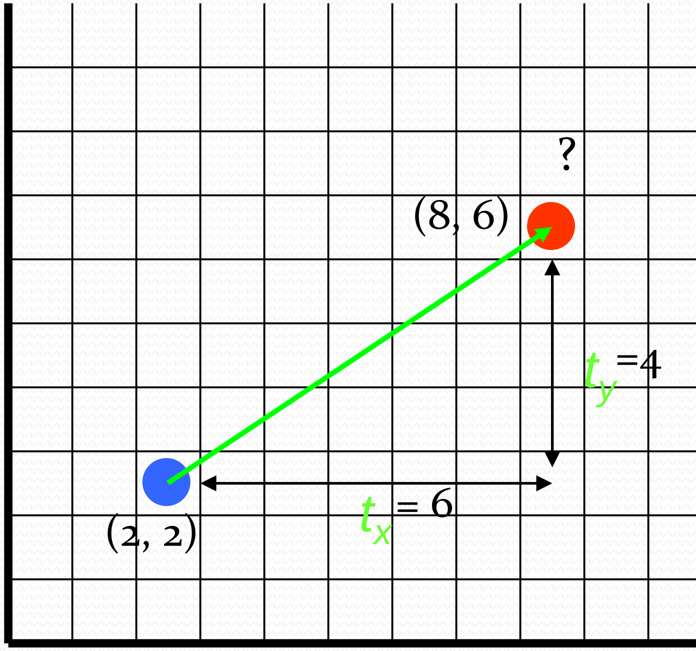
Translation Equations:

$$p'_x = p_x + t_x$$
$$p'_y = p_y + t_y$$

Or in matrix form

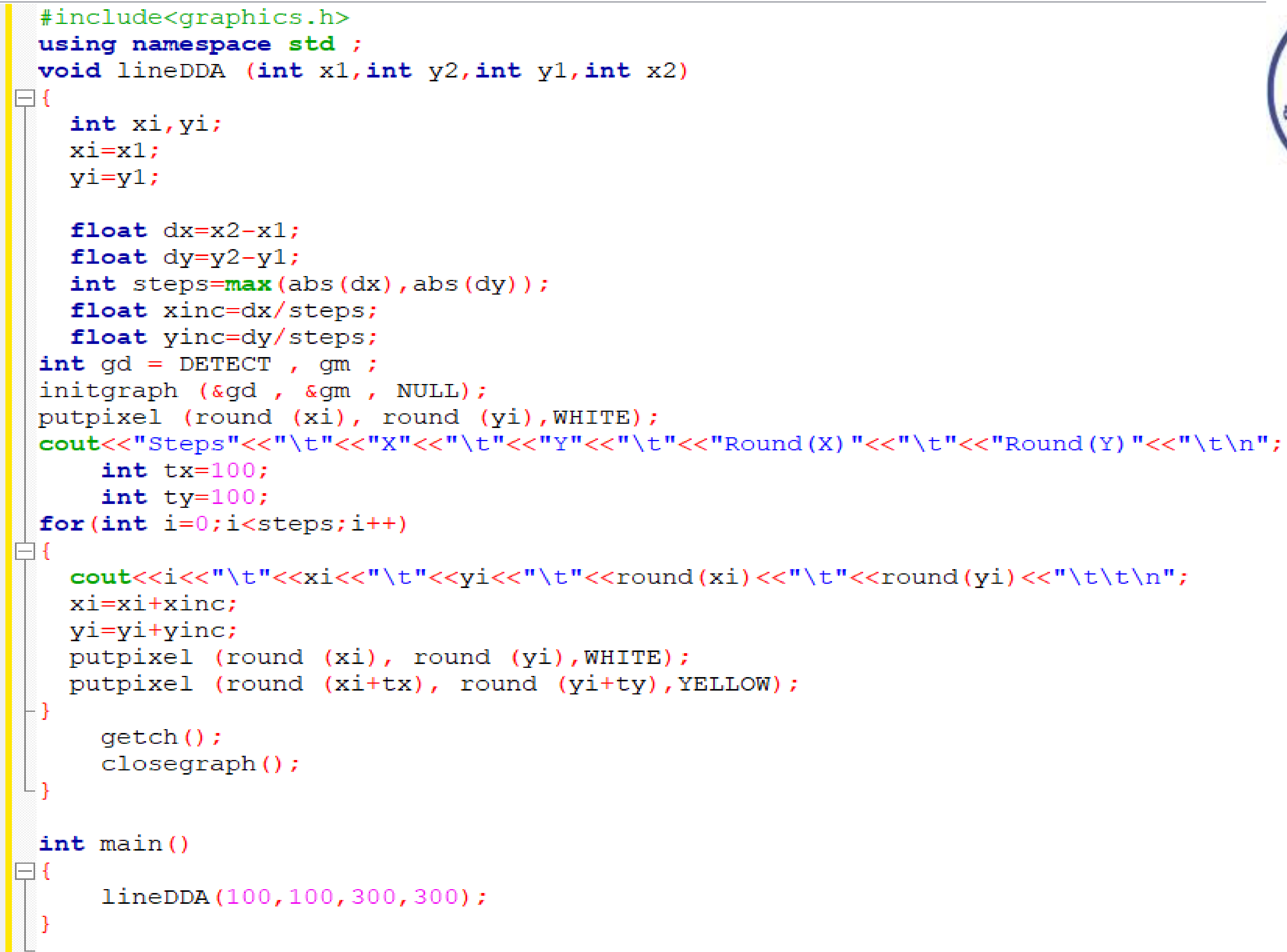
$$P' = P + T$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$



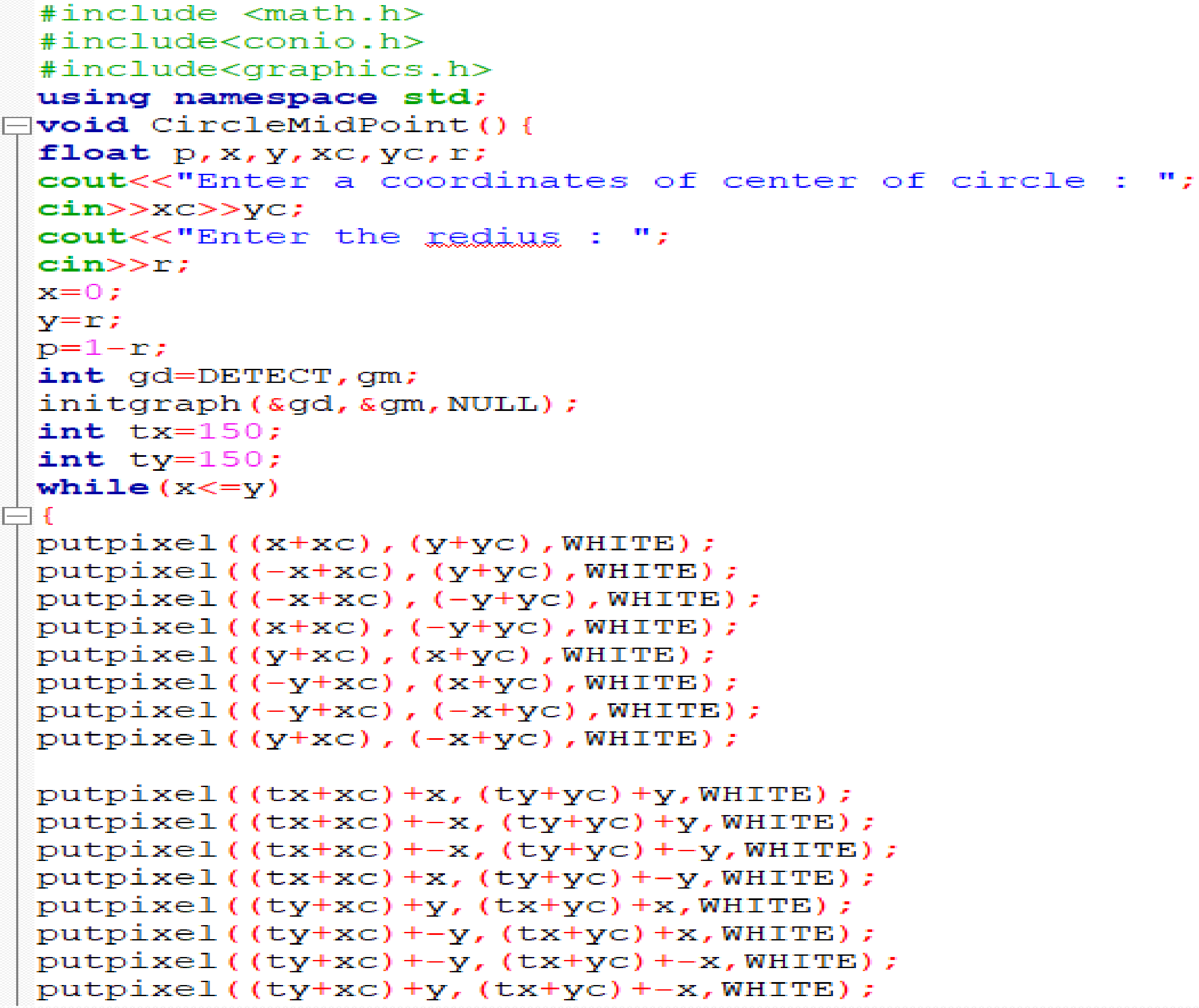
Translation

Line DDA - Translation



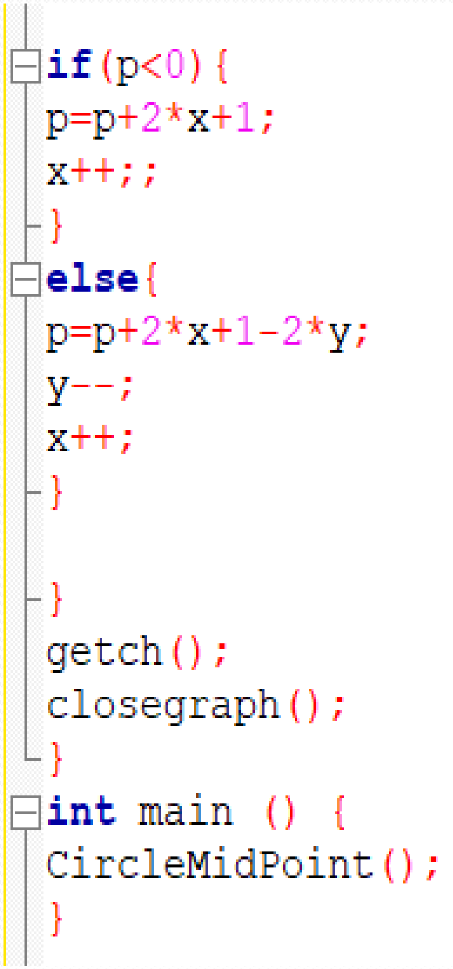
```
#include<graphics.h>
using namespace std ;
void lineDDA (int x1,int y2,int y1,int x2)
{
  int xi,yi;
  xi=x1;
  yi=y1;

  float dx=x2-x1;
  float dy=y2-y1;
  int steps=max(abs(dx),abs(dy));
  float xinc=dx/steps;
  float yinc=dy/steps;
int gd = DETECT , gm ;
initgraph (&gd , &gm , NULL);
putpixel (round (xi), round (yi),WHITE);
cout<<"Steps"<<"\t"<<"X"<<"\t"<<"Y"<<"\t"<<"Round(X)"<<"\t"<<"Round(Y)"<<"\t\n";
    int tx=100;
    int ty=100;
for(int i=0;i<steps;i++)
{
  cout<<i<<"\t"<<xi<<"\t"<<yi<<"\t"<<round(xi)<<"\t"<<round(yi)<<"\t\t\n";
  xi=xi+xinc;
  yi=yi+yinc;
  putpixel (round (xi), round (yi),WHITE);
  putpixel (round (xi+tx), round (yi+ty),YELLOW);
}
    getch();
    closegraph();
}

int main()
{
    lineDDA(100,100,300,300);
}
```

Circle MidPoint- Translation



```
#include <math.h>
#include<conio.h>
#include<graphics.h>
using namespace std;
void CircleMidPoint(){
float p,x,y,xc,yc,r;
cout<<"Enter a coordinates of center of circle : ";
cin>>xc>>yc;
cout<<"Enter the redius : ";
cin>>r;
x=0;
y=r;
p=1-r;
int gd=DETECT,gm;
initgraph(&gd,&gm,NULL);
int tx=150;
int ty=150;
while(x<=y)
{
putpixel((x+xc),(y+yc),WHITE);
putpixel((-x+xc),(y+yc),WHITE);
putpixel((-x+xc),(-y+yc),WHITE);
putpixel((x+xc),(-y+yc),WHITE);
putpixel((y+xc),(x+yc),WHITE);
putpixel((-y+xc),(x+yc),WHITE);
putpixel((-y+xc),(-x+yc),WHITE);
putpixel((y+xc),(-x+yc),WHITE);

putpixel((tx+xc)+x,(ty+yc)+y,WHITE);
putpixel((tx+xc)+-x,(ty+yc)+y,WHITE);
putpixel((tx+xc)+-x,(ty+yc)+-y,WHITE);
putpixel((tx+xc)+x,(ty+yc)+-y,WHITE);
putpixel((ty+xc)+y,(tx+yc)+x,WHITE);
putpixel((ty+xc)+-y,(tx+yc)+x,WHITE);
putpixel((ty+xc)+-y,(tx+yc)+-x,WHITE);
putpixel((ty+xc)+y,(tx+yc)+-x,WHITE);
```

Circle MidPoint- Translation



```
if(p<0){
p=p+2*x+1;
x++;;
}
else{
p=p+2*x+1-2*y;
y--;
x++;
}


}
getch();
closegraph();
}
int main () {
CircleMidPoint();
}
```

Scaling

Scaling

Scaling changes the size of an object and involves two scale factors S_x and S_y for the x- and y- coordinates respectively

Scaling Equations:

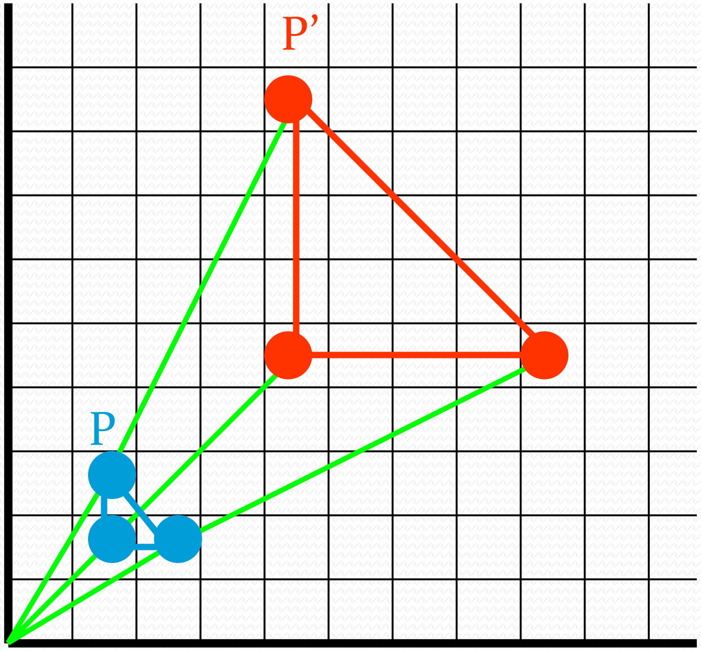
$$p'_x = p_x * S_x$$
$$p'_y = p_y * S_y$$

Or in matrix form

$$P' = S \bullet P$$

Scale matrix as:

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

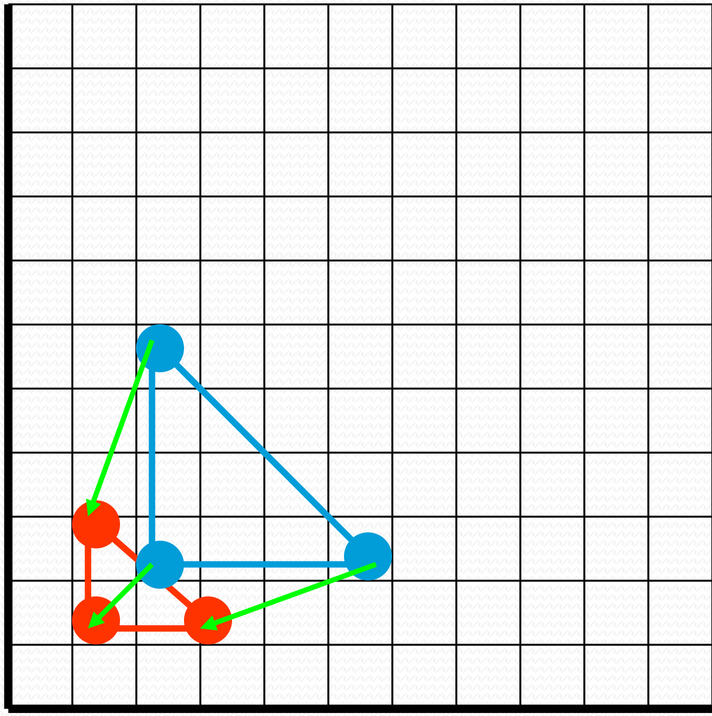


Scaling

Scaling

- If the scale factors are in between 0 and 1 ➔ the points will be moved closer to the origin ➔ the object will be smaller.

- Example :
  - P(2, 5), Sx = 0.5, Sy = 0.5
  - Find P' ?

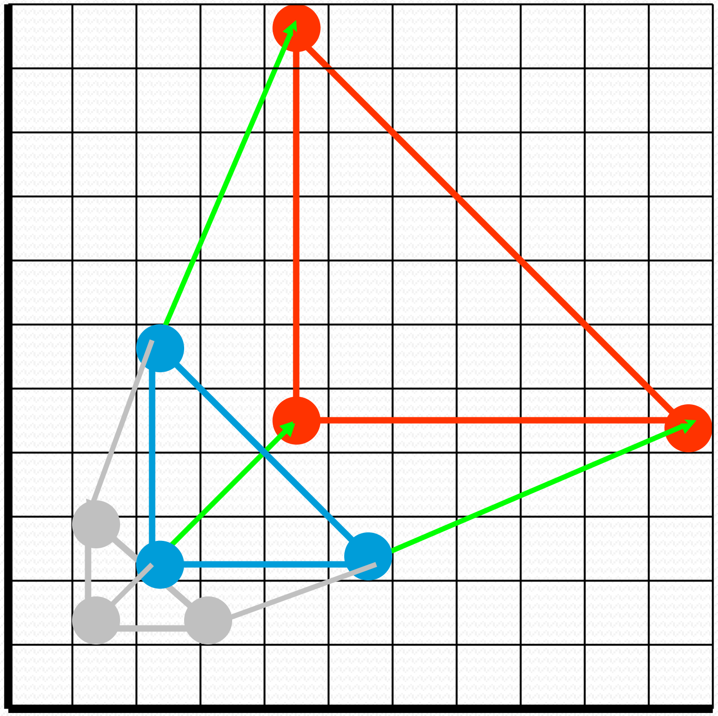


Scaling

Scaling

- If the scale factors are larger than 1 ➔ the points will be moved away from the origin ➔ **the object will be larger.**

- **Example :**
  - P(2, 5), **Sx = 2, Sy = 2**
  - **Find P' ?**



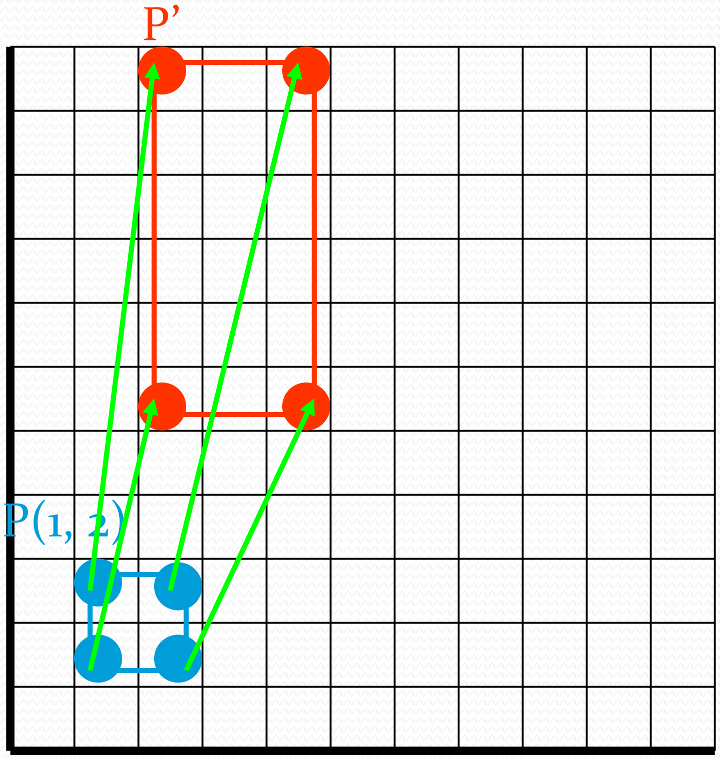
Scaling

Scaling

- If the scale factors are the same, $S_x = S_y$ ➔ **uniform scaling** (Only change in size (as previous example)

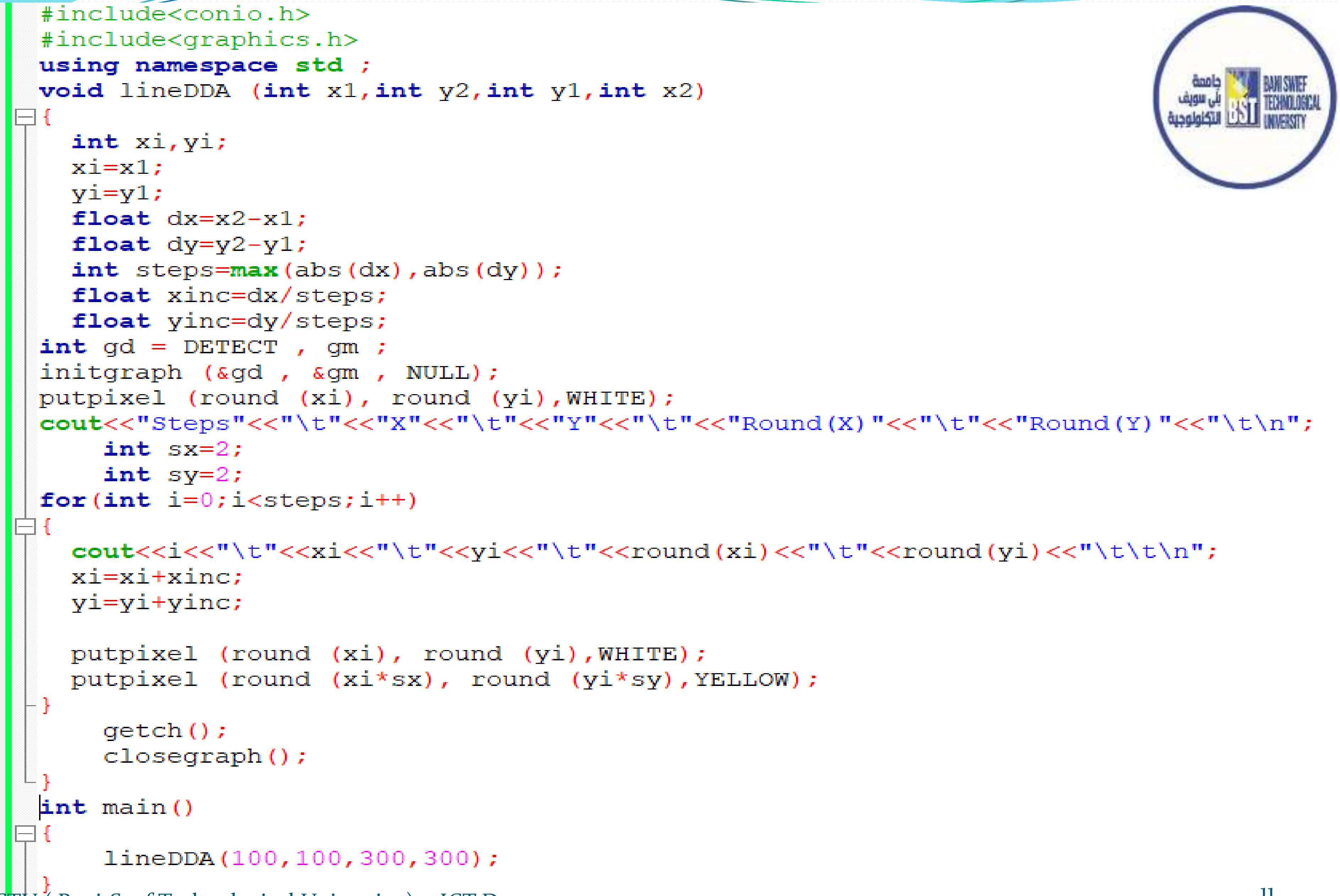
- If $S_x \neq S_y$ ➔ differential scaling.
- Change in size and shape
- Example : square → rectangle

- **P(1, 3), $S_x = 2$, $S_y = 5$, P' ?**



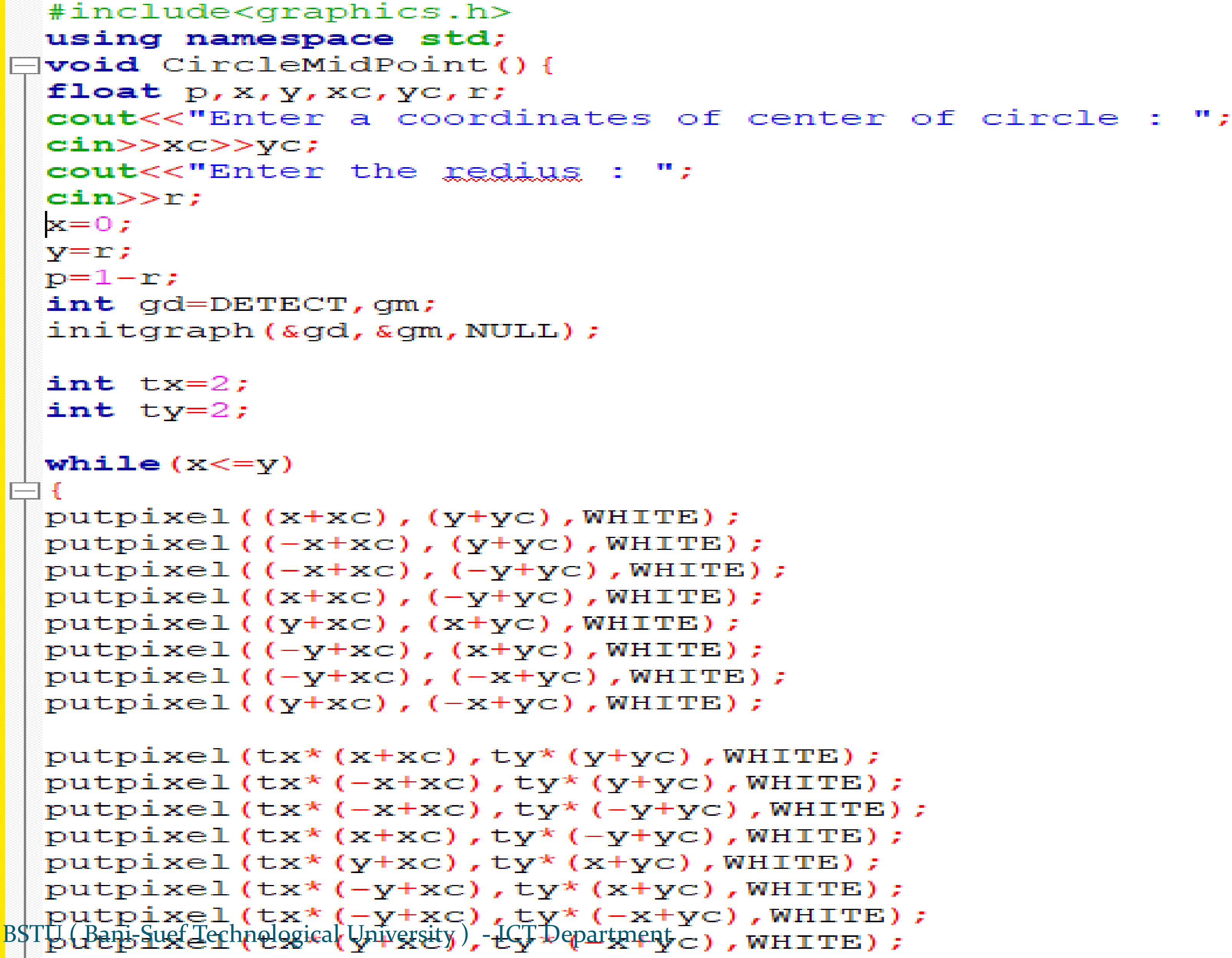
Scaling

Line DDA - Scaling



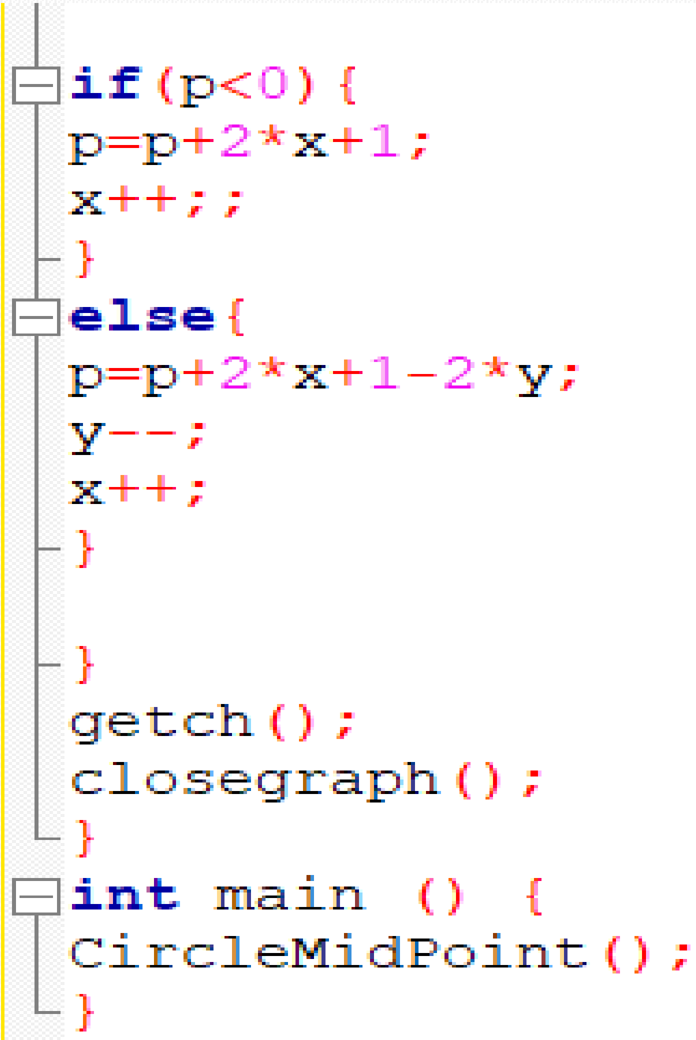
```
#include<conio.h>
#include<graphics.h>
using namespace std ;
void lineDDA (int x1,int y2,int y1,int x2)
{
  int xi,yi;
  xi=x1;
  yi=y1;
  float dx=x2-x1;
  float dy=y2-y1;
  int steps=max(abs(dx),abs(dy));
  float xinc=dx/steps;
  float yinc=dy/steps;
int gd = DETECT , gm ;
initgraph (&gd , &gm , NULL);
putpixel (round (xi), round (yi),WHITE);
cout<<"Steps"<<"\t"<<"X"<<"\t"<<"Y"<<"\t"<<"Round(X)"<<"\t"<<"Round(Y)"<<"\t\n";
    int sx=2;
    int sy=2;
for(int i=0;i<steps;i++)
{
  cout<<i<<"\t"<<xi<<"\t"<<yi<<"\t"<<round(xi)<<"\t"<<round(yi)<<"\t\t\n";
  xi=xi+xinc;
  yi=yi+yinc;

  putpixel (round (xi), round (yi),WHITE);
  putpixel (round (xi*sx), round (yi*sy),YELLOW);
}
    getch();
    closegraph();
}
int main()
{
    lineDDA(100,100,300,300);
}
```

Circle MidPoint- Scaling



```
#include<graphics.h>
using namespace std;
void CircleMidPoint(){
float p,x,y,xc,yc,r;
cout<<"Enter a coordinates of center of circle : ";
cin>>xc>>yc;
cout<<"Enter the redius : ";
cin>>r;
x=0;
y=r;
p=1-r;
int gd=DETECT,gm;
initgraph(&gd,&gm,NULL);

int tx=2;
int ty=2;

while(x<=y)
{
putpixel((x+xc),(y+yc),WHITE);
putpixel((-x+xc),(y+yc),WHITE);
putpixel((-x+xc),(-y+yc),WHITE);
putpixel((x+xc),(-y+yc),WHITE);
putpixel((y+xc),(x+yc),WHITE);
putpixel((-y+xc),(x+yc),WHITE);
putpixel((-y+xc),(-x+yc),WHITE);
putpixel((y+xc),(-x+yc),WHITE);

putpixel(tx*(x+xc),ty*(y+yc),WHITE);
putpixel(tx*(-x+xc),ty*(y+yc),WHITE);
putpixel(tx*(-x+xc),ty*(-y+yc),WHITE);
putpixel(tx*(x+xc),ty*(-y+yc),WHITE);
putpixel(tx*(y+xc),ty*(x+yc),WHITE);
putpixel(tx*(-y+xc),ty*(x+yc),WHITE);
putpixel(tx*(-y+xc),ty*(-x+yc),WHITE);
putpixel(tx*(y+xc),ty*(-x+yc),WHITE);
```

Circle MidPoint- Scaling



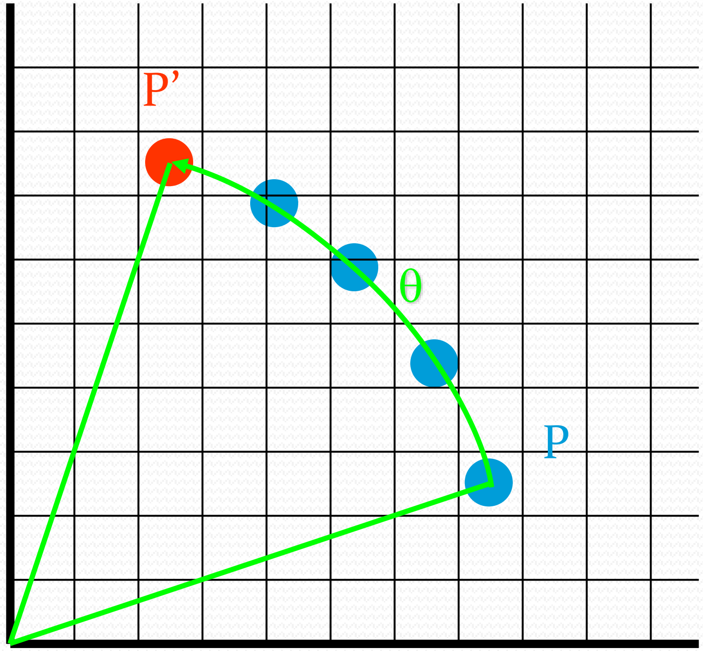
```
if(p<0){
p=p+2*x+1;
x++;;
}
else{
p=p+2*x+1-2*y;
y--;
x++;
}

}
getch();
closegraph();
}
int main  ()  {
CircleMidPoint();
}
```

Rotation

Rotation

A rotation repositions all points in an object along a circular path in the plane centered at the pivot point.

First, we'll assume the pivot is at the origin.



Rotation

Rotation

Scaling Equations:

$$p'_x = p_x \cos\theta - p_y \sin\theta$$

$$p'_y = p_x \sin\theta + p_y \cos\theta$$

Or in matrix form

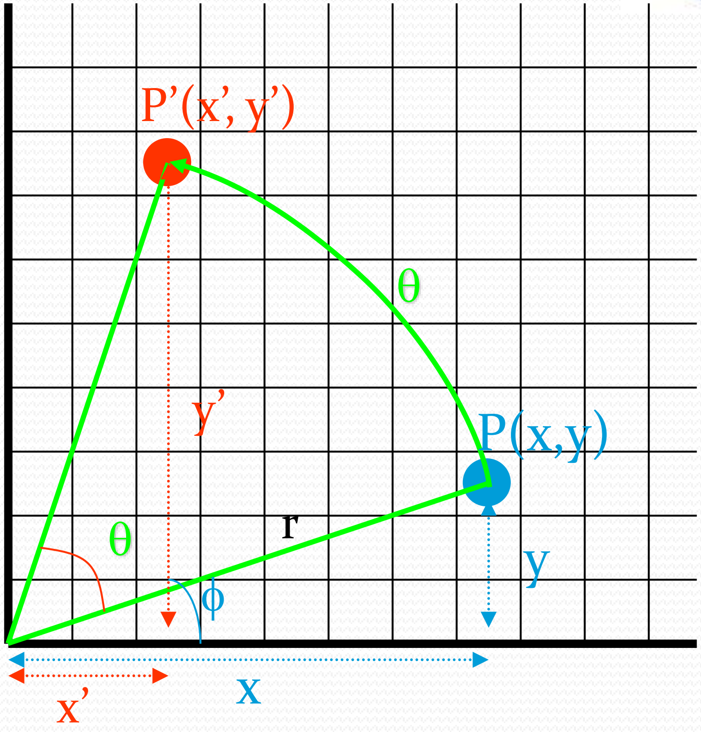
P' = R • P

- θ can be clockwise (-ve) or counterclockwise (+ve)

- Rotation matrix

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix}$$


P'(x', y')
θ
y'
P(x,y)
θ
r
ϕ
y
x
x'