# MỤC LỤC

## Nội dung

# 1. Giới thiệu phần mềm paint

- Đồ án này thực hiện việc mô phỏng lại phần mềm vẽ Paint. Bao gồm các công cụ vẽ hình, tô màu, và các phép biến hình cơ bản.

- Các chức năng cơ bản của Paint đều được thực hiện từng bước thông qua cơ sở lý thuyết, thuật toán.

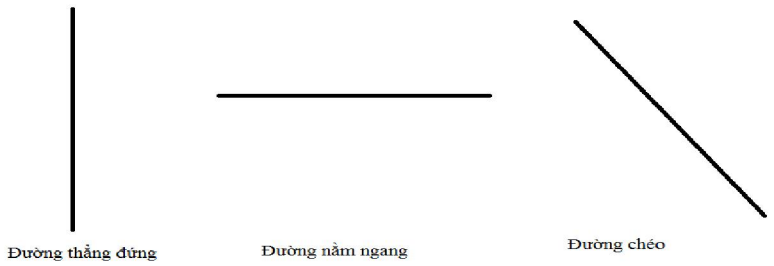
- Phần mềm được viết bằng ngôn ngữ Python, và được biên dịch sang file exe để thực thi.

# 2. Cơ sở lý thuyết và thuật toán vẽ hình

**a. Vẽ đường thẳng (DDA):**

- **Ý tưởng:**

- Ta sẽ xây dựng thuật toán vẽ đường, bằng cách nối 2 điểm (x1, y1), (x2, y2) cho trước, bằng cách tìm các điểm nằm giữa 2 điểm trên để tạo nên đường thẳng sau đó dùng hàm putpixel vào từng điểm.
- Ta sẽ chia vẽ đường thẳng thành 3 trường hợp: đường ngang, đường thẳng đứng, đường xiên.



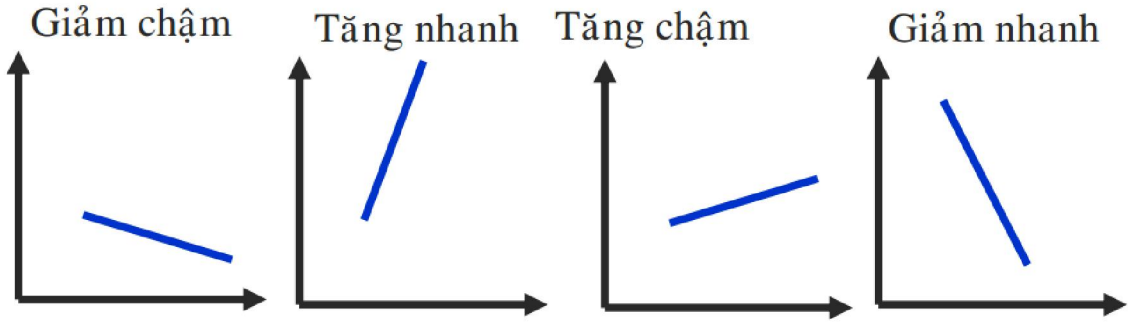

- Trong hàm vẽ đường ta sẽ tính toán 2 thông số dx = endPoint.x – startPoint.x và dy = endPoint.y – startPoint.y, để xác định đường thẳng đang rơi vào trường hợp nào

- Đối với trường hợp đường thẳng đứng (dx = 0, vì tọa độ x của 2 điểm như nhau):
  - Khi đó ta chỉ tính các điểm y kế tiếp sẽ như thế nào, còn x vẫn giữ nguyên.
  - Lúc này ta sẽ thêm một điều kiện nếu tọa độ điểm bắt đầu y nhỏ hơn tọa độ điểm kết thúc y (tức là dy > 0), khi đó ta sẽ đi từ điểm bắt đầu y rồi cộng dồn 1 cho các điểm tiếp theo, vẽ ra các pixel cho đến khi đến điểm kết thúc y.
  - Ngược lại nếu dy < 0, ta cũng sẽ đi từ điểm bắt đầu y, nhưng lúc này với mỗi y tiếp theo ta sẽ trừ dồn đi 1, cho đến điểm kết thúc y.

- Đối với trường hợp đường ngang (dy = 0, vì tọa độ y của 2 điểm như nhau):
  - Lúc này ta sẽ tính các điểm x kế tiếp, còn y vẫn giữ nguyên.
  - Ta sẽ làm ngược lại với cách ở trên, thay vì xét cho dy, và điểm bắt đầu, kết thúc của y ta sẽ thay thế thành xét cho dx, bắt đầu đến điểm kết thúc x.

- Đối với trường hợp đường chéo (dx, dy != 0):
  - Ta sẽ xây dựng phương trình đường thẳng có dạng y = mx + b. Nhưng ở đây sẽ xuất hiện vấn đề đó là nên chọn lặp theo x và tính y theo x hay nên chọn lặp theo y và tính x theo y.

  - Vấn đề trên được giải quyết bằng cách xác định đường thẳng đó là dạng tăng nhanh giảm nhanh, hay tăng chậm giảm chậm.




  - Để biết đâu là tăng nhanh giảm nhanh hay tăng chậm giảm chậm, ta dựa trên dx và dy, biểu thị độ biến thiên ít hay nhiêu của đường. Nếu dx > dy, ta sẽ thực hiện tăng x lên 1 sau mỗi vòng lặp và tính y theo x. Ngược lại nếu dx < dy, ta sẽ thực hiện tăng y lên 1 và tính x theo y.

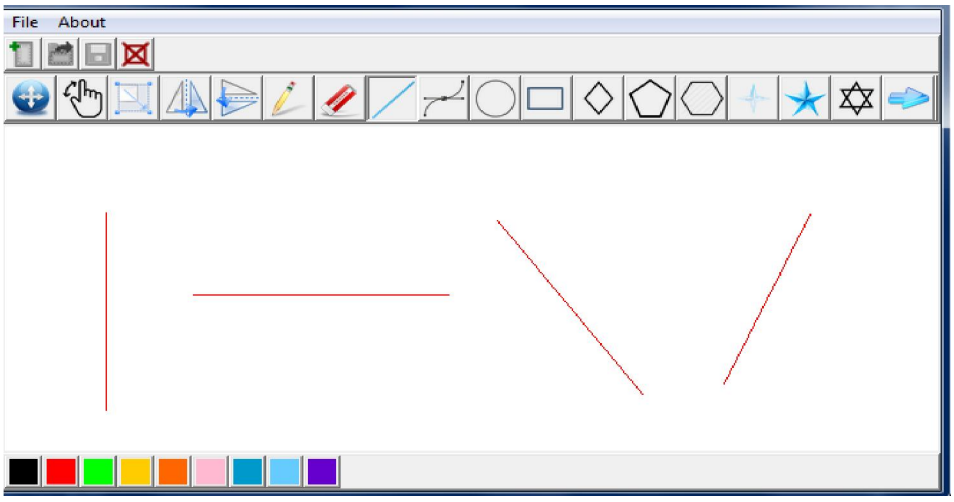
  - Nếu như không phân biệt các đường chéo đó sẽ dẫn đến tình trạng, có đường thẳng vẽ đứt khúc rời, không liền mạch do khi đó ta chọn độ biến thiên quá nhỏ.

  - Sau khi xác định được dạng đường chéo, ta lần lượt tăng dần biến có độ biến thiên nhiều hơn, và tính biến còn lại. Sau khi có kết quả x, y ta tiến hiện làm tròn và vẽ theo chúng theo hàm putpixel(x, y).

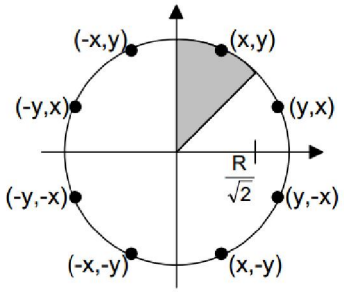
- **Thuật giải:**
  - Phần này được trình bày chi tiết trong file source code drawPattern.py -> hàm lineDDA.

- **Ưu nhược điểm:**
  - Ưu điểm: Ý tưởng thuật toán đơn giản, dễ cài đặt, gần gũi với con người.
  - Nhược điểm: Vì trong hàm có sử dụng các phép nhân và chia y = mx + b, m có thể là dạng số thập phân. Nếu quá trình lặp nhiều lần sẽ khiến cho thuật toán chậm. Nhưng có một cách tối ưu đó là sử dụng lại biến y đã tính để cộng dồn cho y tiếp theo ví dụ như y2 = y1 + m hoặc x2 = x1 + m sẽ rút ngắn được thời gian tính toán.

- **Cách sử dụng:**
  - Ấn và kéo thả chuột tạo đường thẳng mong muốn. Nếu muốn vẽ dạng đường nằm ngang hoặc thẳng đứng có thể ấn Shift.

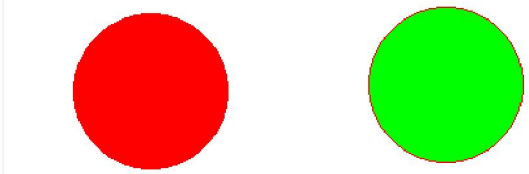



**b. Vẽ đường tròn (thuật toán midPoint):**



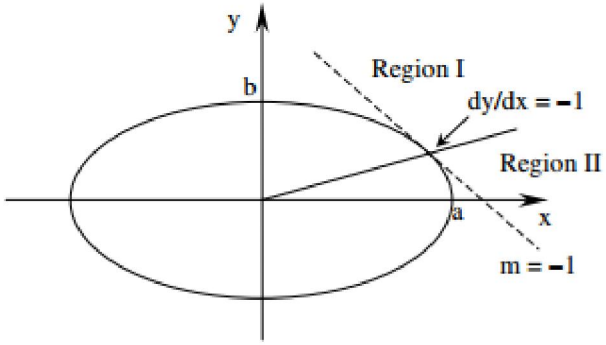

- **Ý tưởng:**
  - Ta sẽ chia đường tròn thành 8 phần bằng nhau, khi đó ta chỉ cần vẽ 1/8 đường tròn và có thể lấy đối xứng cho tất cả các phần còn lại.
  - Vậy ta chỉ cần xây dựng hàm vẽ 1/8 cung tròn xuất phát từ trục Oy sang chiều dương của trục Ox như hình vẽ, rồi viết hàm lấy đối xưng các cặp (x, y).
  - Ta sẽ dựa trên thuật toán mid point đó là ứng với một điểm tiếp theo bằng cách tính x = x + 1, y = y – 1 (với thiết lập x = 0, y=R)
  - Nếu điểm (x, y) đó đã nằm trong thì ta sẽ giữ nguyên y, ngược lại nếu điểm đó nằm ngoài tức điểm có xu hướng đi ra ngoài khi đó ta phải kéo y vào trong gần hơn thì y = y – 1.
  - Thế thì làm sao để có thể xác định được điểm đang nằm trong hay ngoài đường tròn? Khi đó ta cần phải xây dựng hàm f để biết được điều này, hàm f được xây dựng dựa trên Nếu fcũ < 0, fmới = fcũ + 2xi + 3, Yi+1 = Yi, ngược lại nếu fcũ >= 0, fmới = fcũ + 2(Xi-Yi) + 5, Yi+1 = Yi – 1. Giá trị f đầu tiên được thiết lập là f = 1 – R.
  - Quá trình lặp lại x = x + 1, cho đến khi x <= y.
  - Cuối cùng sau khi có tọa độ các điểm, ta chỉ cần cộng với tọa độ tâm vẽ được đường tròn theo tâm và bán kính.
- **Thuật giải:**
  - Phần này được trình bày chi tiết trong file source code drawPattern.py -> hàm circleMidPoint.
- **Cách sử dụng:**

- Chọn một điểm làm tâm sau đó ấn shift để vẽ đường tròn



**c. Vẽ đường eclipse (midPoint):**

- **Ý tưởng:**
  - Ta cũng sẽ chia eclipse thành 4 phần con, sau đó vẽ một phần của eclipse và lấy đối xứng cho các phần còn lại.
  - Ta sẽ vẽ eclipse ở cung thứ nhất, bằng cách chia cung đó thành 2 cung con region 1, region 2.




  - Ta sẽ bắt đầu vẽ từ trục tung điểm b, đến điểm độ dóc là region 1. Và region 2 sẽ là từ điểm độ dóc cho đến trục hoành Ox.
  - Để xác định được điểm độc dốc ta cần 2 yếu tố px, py. Được tính như sau

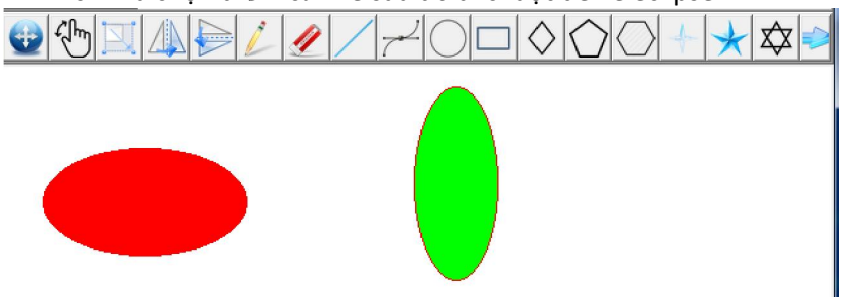
```
rxSq = rx ** 2
rySq = ry ** 2
x = 0
y = ry
px = 0
py = 2 * rxSq * y
drawEclipse(centerPoint, x, y, color, img)
p = rySq - (rxSq * ry) + (0.25 * rxSq)
while px < py:
    x = x + 1
    px = px + 2*rySq
    if p < 0:
        p = p + rySq + px
    else:
        y = y - 1
        py = py - 2*rxSq
        p = p + rySq + px - py
    drawEclipse(centerPoint, x, y, color, img)
```

  - Trong vòng lặp while, khi px bằng py, tức đến điểm dốc, thuật toán sẽ ngưng lại, và tiếp túc vẽ đến region 2.

```
p = rySq*(x+0.5)*(x+0.5) + rxSq*(y-1)*(y-1) - rxSq*rySq
while y > 0:
    y = y - 1
    py = py - 2 * rxSq;
    if p > 0:
        p = p + rxSq - py;
    else:
        x = x + 1
        px = px + 2 * rySq;
        p = p + rxSq - py + px;
    drawEclipse(centerPoint, x, y, color, img);
```

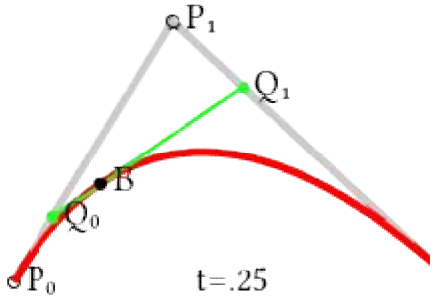
- **Thuật giải:**
  - Phần này được trình bày chi tiết trong file source code drawPattern.py -> hàm eclipseMidPoint.

- **Cách sử dụng:**
  - Ta chọn điểm cần vẽ sau đó di chuột để vẽ eclipse.



**d. Vẽ đường cong:**

- **Ý tưởng:**
  - Ta sẽ áp dụng thuật toán Bezier dạng đường cong cấp 2.
  - Đầu tiên ta sẽ tạo 2 đỉnh cố định cho đường cong, sau đó sẽ có thêm một đỉnh để dịch chuyển độ cong mong muốn.



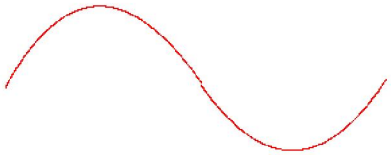

  - Thuật giải chính như sau:

```
while t < 1:
    x = int(p0[0]*(1-t)**2 + 2*(1-t)*t*p1[0] + p2[0]*t**2)
    y = int(p0[1]*(1-t)**2 + 2*(1-t)*t*p1[1] + p2[1]*t**2)
    img.putpixel((x,y), color)
    t = t + 0.001
```

  - p0, p2 là các tọa độ điểm cố định, p1 là tọa độ điểm di chuyển để điều chỉnh độ cong, khi t được cộng dồn một giá trị càng nhỏ thì đường cong sẽ càng mượt, và mịn hơn.

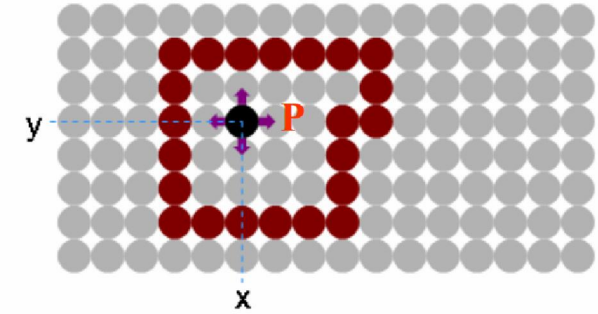
- **Thuật giải:**
  - Phần này được trình bày chi tiết trong file source code drawPattern.py -> hàm curve.

- **Cách sử dụng**:
  - Đầu tiên ta tạo 2 điểm cố định, sau đó tạo thêm một điểm và điều chỉnh độ cong cho điểm đó.



**e. Tô màu:**

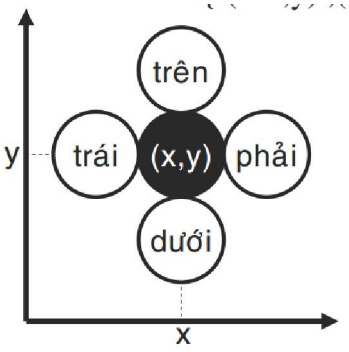
- **Ý tưởng:**
  - Ta sẽ thực hiện bằng phương pháp tô màu loang khử đệ quy.




  - Đầu tiên ta sẽ có điểm bắt đầu nằm trong vùng biên cần tô. Ta xem như điểm cần tô là 1 seed, và bỏ seed vào trong list.

  - Vòng lặp sẽ luôn kiểm tra nếu còn seed trong list thì tiếp tục thực thi.

  - Trong vòng lặp này, ta sẽ thực hiện lấy màu của điểm đó bằng hàm getPixel, nếu màu đó giống với màu cần tô (tức đã tô qua điểm đó) hoặc điểm đó ra ngoài khung viền màn hình thì sẽ loại bỏ, ngược lại nếu màu đó giống với màu nền (tức chưa được tô).

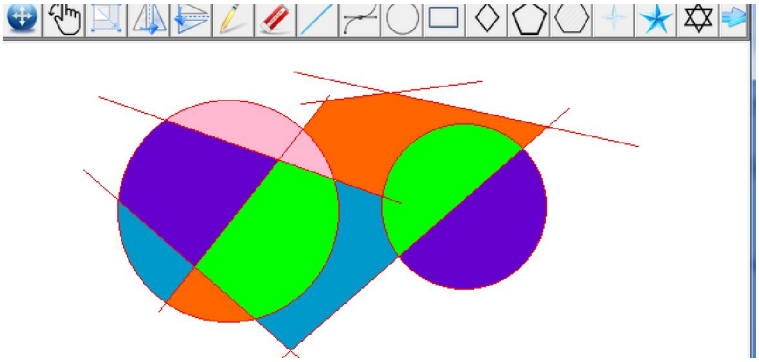



  - Ta sẽ tô màu tại điểm đó bằng hàmputpixel . Sau đó ta sẽ thêm các seed mới vào list, với các seed mới là các điểm lân cận của điểm hiện tại đang xét.

  - Sau khi qua nhiều vòng lặp ta sẽ có được hình cần tô.

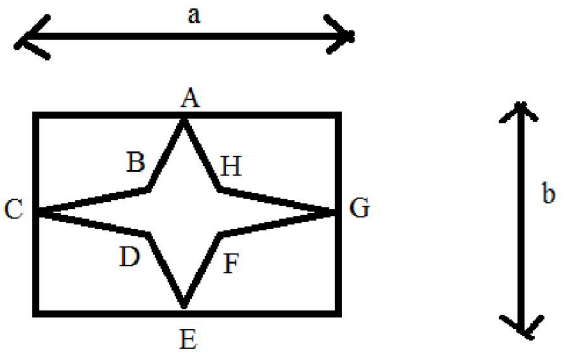
- Thuật toán này có nhược điểm đó là lặp lại nhiều lần khiến cho thời gian xử lý sẽ chậm hơn so với thuật toán tô màu theo dòng quét, nhưng quá trình cài đặt thì dễ dàng, và có thể tạm chấp nhận.

- **Thuật giải**
  - Phần này được trình bày chi tiết trong file source code drawPattern.py -> hàm fillColor.
- **Cách sử dụng**:
  - Ta sẽ chọn vào điểm trong biên cần tô.



**f. Một số hình cơ bản**

- **Ý tưởng:**
  - Dưới đây là một vài ý tưởng về cách vẽ hình đa giác như tam giác, thoi vuông, ngũ giác, lục giác, hình sao …
  - Đầu tiên ta cần đặt đối tượng vào khung hình chữ nhật, sau đó xác định các điểm của khối hình cần vẽ. Lúc này ta chỉ cần dùng hàm vẽ đường thẳng để nối các điểm đó lại tạo thành hình mong muốn. Ta có ví dụ như sau:

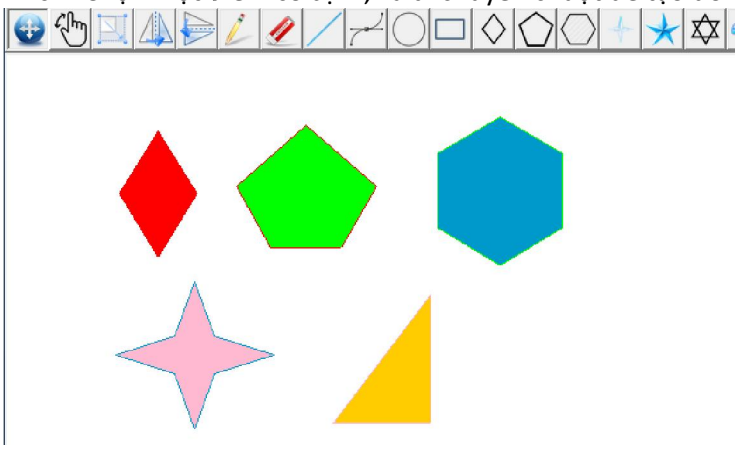



  - Giả sử ta có hình ngôi sao 4 cánh cần vẽ. Ta sẽ lồng vào trong khối hình chữ nhất với chiều dài a, rộng b. Để cân đối, ta sẽ cho các đỉnh của ngôi sao trùng với trung điểm của các cạnh hình chữ nhật đó là A, C, E, G.
  - Thế thì làm sao xác định được các điểm còn lại B, D, F, H? Việc này ta chỉ có thể đánh giá bằng mắt để tạo sự cân đối cho hình, ví dụ để có tọa độ B, ta sẽ ước chừng B(¾(a/2), ¾(b/2)), và như thế ta sẽ áp dụng cho các điểm D, F, H.
  - Sau khi có được tọa độ các điểm, ta sẽ tiến hành dùng hàm vẽ đường để nối các điểm lại tạo thành hình mong muốn.

- Tuy nhiên với cách này có một vài hạn chế đó là các hình có thể sẽ không cân xứng, vì đo bằng mắt thường. Nhưng có thể điều chỉnh tọa độ để phù hợp.

- **Thuật giải:**
  - Phần này được trình bày chi tiết trong file source code drawPattern.py -> các hàm fourStar, star, sixStar, fivePolygon ….

- **Cách sử dụng:**
  - Chọn một điểm cố định, và di chuyển chuột để tạo đối tượng mong muốn.



**g. Cropping:**

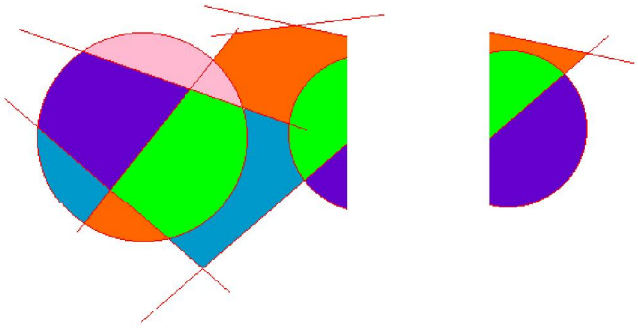
- **Ý tưởng:**
  - Cropping được sử dụng trong việc cắt một phần ảnh để tạo thành một đối tượng  có thể tịnh tiến, di chuyển ….
  - Cách đơn giản được thực hiện đó là ta sẽ xác định 2 điểm đầu và cuối (tao thành một khung hình chữ nhật) để xác định đối tượng cần cắt.
  - Sau đó ta sẽ lần lượt quét toàn bộ các pixel có trong khung này, với những pixel trùng màu nền, ta sẽ loại bỏ. Ngược lại khác màu nền ta sẽ lưu lại tọa độ pixel và color. Rồi lưu vào danh sách các list pixel Object, sau đó ta sẽ xóa bỏ các đối tượng cũ bằng cách tô lại màu nền cho pixel đó.
  - Lúc này ta sẽ có được listPixelObject.
- **Thuật giải:**
  - Phần này được trình bày chi tiết trong file source code drawPattern.py -> hàm cropping.

**h. Phép tịnh tiến**

- **Ý tưởng:**
  - Ta sẽ tính độ dời giữa điểm trong đối tượng đã cropping và vị trí mới cần tịnh tiến đến.
  - Sau đó lận lượt cộng các giá trị trong listPixelObject với độ dời tương ứng theo deltaX và deltaY. Và vẽ các pixel đó lên màn hình.
- **Thuật giải:**

- Phần này được trình bày chi tiết trong file source code drawPattern.py -> hàm moveTransition.

- **Cách sử dụng**:
  - Ta sẽ chọn khung đối tượng cần cropping sau đó di chuột đến vị trí mong muốn



**i. Phép quay:**

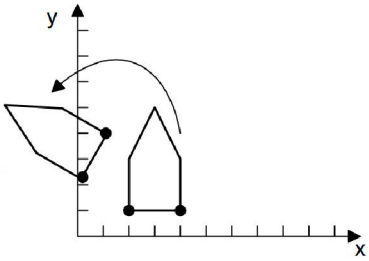
- **Ý tưởng:**
  - Sau khi đã cropping đối tượng, ta đã có các dữ liệu input đầu vào là pixelListObject, tọa độ center (vị trí tâm cần quay), góc quay alpha.
  - Ta sẽ có công thức áp dụng như sau:

- Tâm quay : O
- - Góc quay : α

$$f : R^2 \xrightarrow{\alpha} R^2$$

$$P(x,y) \longrightarrow Q(x,y)$$

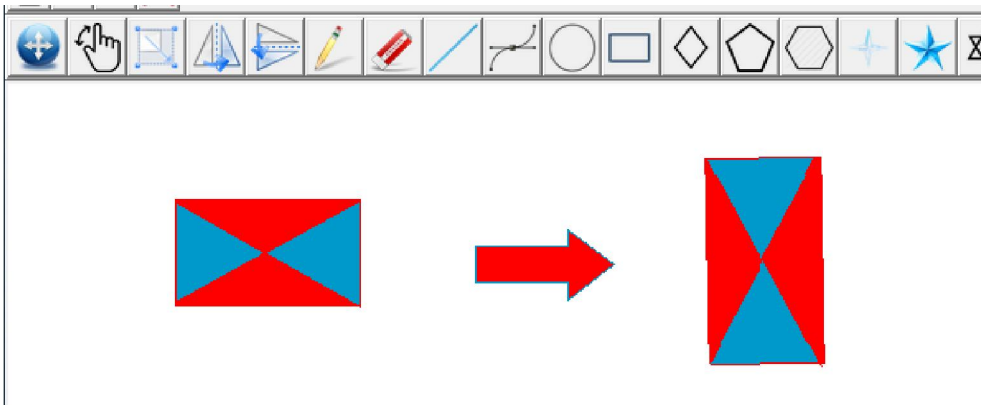
$$Qx = Px\ Cos\alpha - Py Sin\alpha$$
$$Qy = Px\ Sin\alpha + Py\ Cos\alpha$$




  - Trong đó Px, Py chính là độ biến thiên của từng phần tử trong pixelListObject so với vị trí tâm đã chọn.
  - Thế nhưng phép xoay có thể ảnh hưởng đến hình ảnh gốc ban đầu, như đó là các pixel sẽ không liền mạch, và bị lệch, đứt khúc.

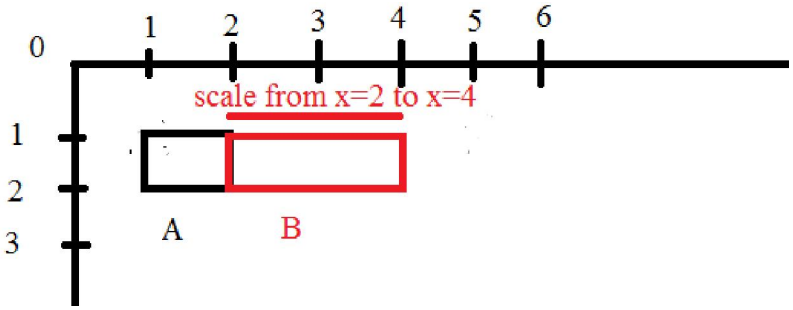
- **Thuật giải:**
  - Phần này được trình bày chi tiết trong file source code drawPattern.py -> hàm moveRotation.

- **Cách sử dụng**:
  - Ta chọn đối tượng cần xoay, sau đó chọn tâm quay và thực hiện quay theo góc mong muốn.



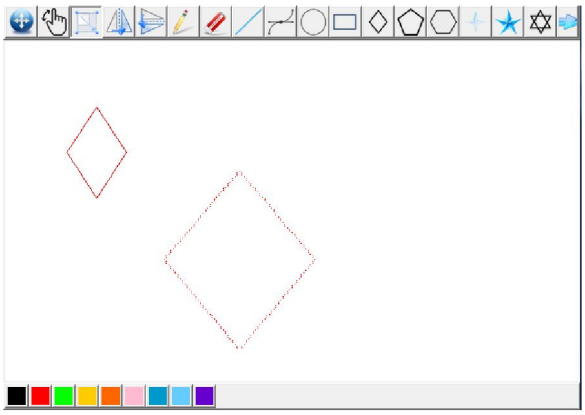
**j. Phép tỉ lệ:**

- **Ý tưởng:**
  - Sau khi đã cropping đối tượng và có được tỉ lệ scalex, scaleY. Ta sẽ lần lượt nhân các phần tử trong listPixelObject tương ứng theo tỉ lệ scaleX, scaleY để ra được hình ảnh cần biến đổi tỉ lệ.
  - scaleX, scaleY ở đây chính là độ biến đổi khi ta bấm và kéo chuột, ta sẽ so sánh điểm đã cố định với độ di chuột là bao nhiêu để tính ra.



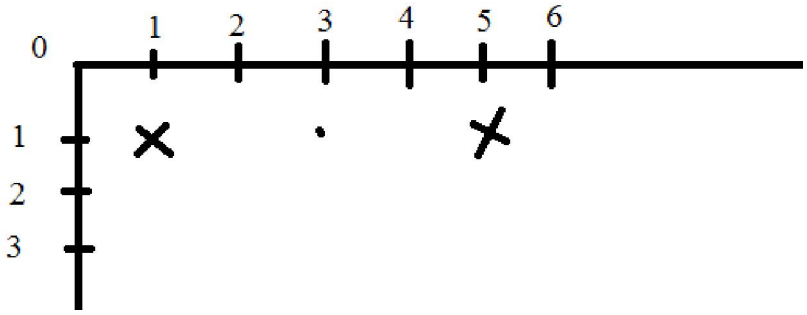

  - Như ta thấy hình ảnh minh họa trên, hình chữ nhật A là ban đầu, ta sẽ kéo từ điểm cố định x=2, đến x=4, khi đó tỉ lệ scaleX =2, scaleY = 1. Ta sẽ lần lượt nhân scaleX, scaleY với các phần tử listPixelObject và ra được hình chữ nhật B mới.
  - Nhưng ở đây sẽ có 2 điều xảy ra đó là: vị trí của đối tượng ban đầu sẽ bị thay đổi, nhưng ta có thể khắc phục bằng cách dùng phép tịnh tiến để kéo về vị trí ban đầu.
  - Nhưng vấn đề thứ 2 đó là vì kéo dãn các phần tử pixel ra nên khoảng cách của các pixel cũng sẽ dang rộng ra, khiến cho các khoảng pixel sẽ không liền mạch tạo ra hình ảnh đứt đoạn. Đây giống như hình ảnh khi ta phóng to đối tượng thì sẽ làm vỡ ảnh tạo ra các lỗ, không đều.
- **Thuật giải:**
  - Phần này được trình bày chi tiết trong file source code drawPattern.py -> hàm scaling.

- **Cách sử dụng**:
  - Ta chọn đối tượng cần scaling, sau đó một điểm cố định và kéo thả để chỉnh tỉ lệ phù hợp.



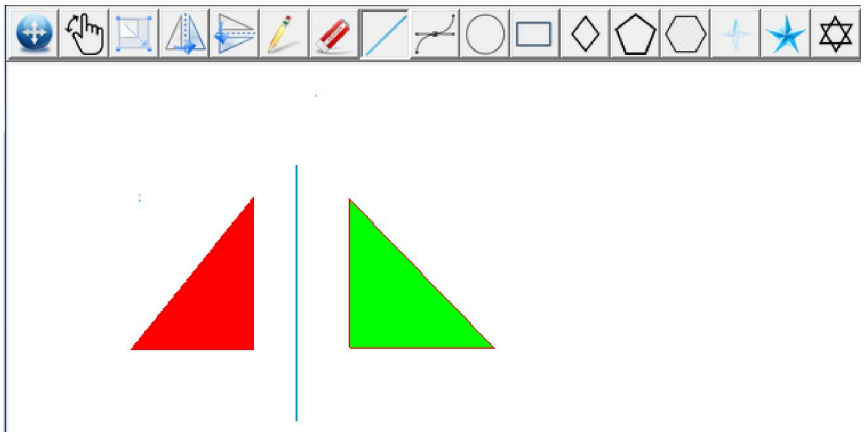
**k. Phép đối xứng**

- **Ý tưởng:**
  - Sau khi chọn được đối tượng cần biến đổi, ta sẽ cho biết vị trí cần lấy đối xứng đó là theo trục x hay trục y
  - Nếu theo trục y, tức khi đó từng phần tử y trong listPixelObject, sẽ giữ nguyên không đổi, chỉ đổi x. Thế thì x mới phải có giá trị thế nào mới hợp lý? Giả sử ta có đối tượng như sau.




  - Với dấu chéo (1, 1) (gọi là A) là đối tượng ban đầu, điểm chấm (3,1) (gọi là B) là điểm đối xứng, và dấu chéo còn lại (5,1) (gọi là C) là vị trí cần vẽ. Vì 2 điểm chéo có khoảng cách với điểm đối xứng là như nhau, nên xA + xC = 2xB -> xC = 2xB – xA = 2*3 – 1 = 5
  - Với cách trên ta sẽ áp dụng cho từng điểm trong listPixelObject, để tính ra các điểm mới.
  - Tượng tự đối với lấy đối xứng theo trục x (trục ngang), tức tọa độ x của các phần tử sẽ không đổi, chỉ thay đổi tọa độ y ta cũng áp dụng tương tự các trên.
- **Thuật giải:**
  - Phần này được trình bày chi tiết trong file source code drawPattern.py -> hàm flipVertical và flipHorizontal.

- **Cách sử dụng**:
  - Chọn đối tượng cần thực hiện, sau đó chọn 1 điểm để lấy đối xứng.



I. **Tổng hợp các phép quay:**

- Trong những ví dụ trên, ta chỉ xây dựng các giải thuật, tạo ra các công cụ tịnh tiến, tỉ lệ, quay, đối xứng … cho một trường hợp nào đó thôi. Nhưng thực tế một đối tượng có thể kết hợp nhiều phép trong đó, nên ta cần một công cụ toán học đó là chuyển thành dạng tọa độ ma trận.

- Dạng ma trận của các phép biến đổi affine cơ sở giúp cho các đối tượng có thể biến đổi một cách nhanh chóng thay vì phải thực hiện từng bước một.

- Ví dụ ta có một đối tượng ta cần kết hợp cả phép tịnh tiến, đối xứng, xoay để tạo thành một đối tượng mới, ta chỉ cần dùng phép biến đổi affine tương ứng với các phép tịnh tiến, quay…. Nhân ma trận lại với nhau, là có thể tạo ra đối tượng mới. Giúp thuận tiện hơn, thay vì phải làm từng bước.

# 3. Tổng kết

- Phần mềm paint sử dụng các cơ sở lý thuyết, thuật toán để vẽ các hình dạng cơ bản, và tạo các công cụ cần thiết.
- Thế nhưng trong phần mềm paint này vẫn còn nhiều tính năng cần được cải thiện thêm để phục vụ thực sự cho người dùng.
- Source code gồm: config.py, drawPattern.py, initState.py, paint.py, setup.py. Trong đó config.py chứa các đường dẫn cho icon, nội dung, initState.py thiết lập giao diện màn hình, drawPattern.py chứa các thuật toán chính vẽ hình, paint.py kết nối các file *.py còn lại. Phần setup.py chỉ giúp cho việc convert sang file exe.

# 4. Một số tài liệu tham khảo:


- http://homepages.inf.ed.ac.uk/rbf/HIPR2/rotate.htm
- http://homepages.inf.ed.ac.uk/rbf/HIPR2/reflect.htm
- http://www.efg2.com/Lab/Graphics/Jean-YvesQueinecBezierCurves.htm
- http://winnyefanho.net/research/MEA.pdf