**TRƯỜNG ĐẠI HỌC NHA TRANG  
KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

**BÀI TẬP CÁ NHÂN**

**MÔN HỌC:** **KĨ THUẬT ĐỒ HỌA.**

**GIÁO VIÊN HƯỚNG DẪN: NGUYỄN ĐÌNH CƯỜNG**

**LỚP: 62-CNTT2**

**SINH VIÊN THỰC HIỆN: TRẦN NGUYỄN BẢO TRUNG**

**MSSV: 60130098**

NHA TRANG, THÁNG 4 NĂM 2022

MỤC LỤC

[N**ỘI DUNG BÁO CÁO** 3](#_Toc101021301)

[**Bài 1: Cài đặt thuật toán Bresenham, Midpoint vẽ đường thẳng, đường tròn, Elipse** 3](#_Toc101021302)

[**a.Thuật toán Bresenham vẽ đoạn thẳng** 3](#_Toc101021303)

[**b.Thuật toán Midpoint vẽ đoạn thẳng** 5](#_Toc101021304)

[**c.Thuật toán Bresenham vẽ đường tròn** 8](#_Toc101021305)

[**d.Thuật toán Midpoint vẽ đường tròn** 10](#_Toc101021306)

[**e.Thuật toán Bresenham vẽ Ellipse** 12](#_Toc101021307)

[**f.Thuật toán Midpoint vẽ Ellipse** 14](#_Toc101021308)

[**Bài 2: Phép biến đổi affine sử dụng DirectX, OpenGL** 17](#_Toc101021309)

[**a.Phép Tịnh Tiến** 17](#_Toc101021310)

[**b.Phép biến đổi tỉ lệ** 19](#_Toc101021311)

[**c.Phép Xoay** 20](#_Toc101021312)

[**Bài 3: Texture Vật thể** 21](#_Toc101021313)

[**Bài 5: Vẽ đường cong và mặt cong Bezier** 23](#_Toc101021314)

[**TÀI LIỆU THAM KHẢO** 25](#_Toc101021315)

# NỘI DUNG BÁO CÁO

Họ Và Tên: Trần Nguyễn Bảo Trung

MSSV: 60137367

Lớp 62-CNTT2

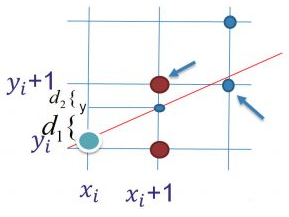
Môn học: Kĩ thuật đồ Họa

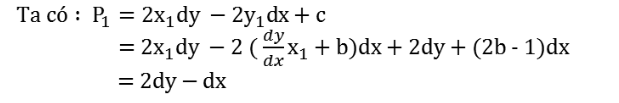
## **Bài 1: Cài đặt thuật toán Bresenham, Midpoint vẽ đường thẳng, đường tròn, Elipse**

### **a.Thuật toán Bresenham vẽ đoạn thẳng**

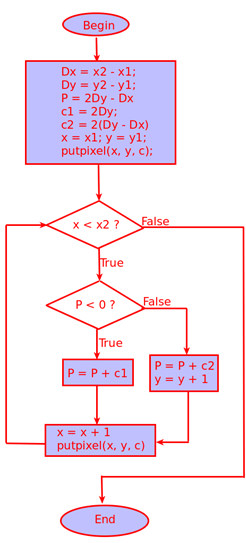
* **Giải thuật vẽ đoạn thẳng của Bresenham**(**Bresenham's line algorithm**) là [giải thuật](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n) xác định các điểm [raster](https://vi.wikipedia.org/wiki/C%E1%BA%A5u_tr%C3%BAc_raster) hai chiều cần vẽ để nhận được xấp xỉ gần đúng của [đoạn thẳng](https://vi.wikipedia.org/wiki/%C4%90o%E1%BA%A1n_th%E1%BA%B3ng) có hai đầu mút là 2 điểm cho trước.
* Đây là một trong những [thuật toán](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n) cổ nhất trong đồ họa máy tính. Thuật toán này được [Jack E. Bresenham](https://vi.wikipedia.org/w/index.php?title=Jack_E._Bresenham&action=edit&redlink=1) thiết kế vào năm [1962](https://vi.wikipedia.org/wiki/1962) tại công ty [IBM](https://vi.wikipedia.org/wiki/IBM).
* Thuật toán được sử dụng rộng rãi, đặc biệt để vẽ đoạn thẳng trên màn hình máy tính. Nó chỉ sử dụng các lệnh cộng trừ số học và lệnh trên [pixel](https://vi.wikipedia.org/wiki/Pixel), có chi phí rẻ và phù hợp với kiến trúc sơ khai của máy tính.
* Đây là một trong những giải thuật [đồ họa máy tính](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_h%E1%BB%8Da_m%C3%A1y_t%C3%ADnh) phát triển sớm nhất. Sự mở rộng của giải thuật này là giải thuật vẽ các đường cong bậc 2.
* **Ý tưởng thuật toán Bresenham:**
* Thay thế các phép toán trên số thực bằng các phép toán trên số nguyên.
* Giảm thời gian của thuật toán hơn so với DDA.
* Hạn chế phép toán được thực hiện để giảm tải thời gian.
* **Thuật giải**

Thuật toán Bresenham đưa ra cách chọn yi+1 là yi hay yi+1 theo một hướng khác . Đó là so sánh khoảng cách giữa điểm thực y với 2 điểm gần kề nó nhất. Nếu điểm nào nằm gần điểm thực hơn thì sẽ được chọn làm điểm vẽ tiếp theo.





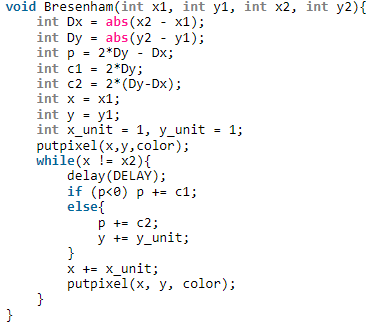
* **Lưu đồ thuật toán**

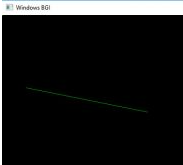


Hình 1a. Lưu đồ thuật toán Bresenham

Giải thuật ban đầu sẽ được trình bày chỉ cho trường hợp [góc phần tám](https://vi.wikipedia.org/w/index.php?title=G%C3%B3c_ph%E1%BA%A7n_t%C3%A1m&action=edit&redlink=1), trong đó đoạn thẳng đi xuống và sang phải (*x*0≤*x*1 và *y*0≤*y*1), và hình chiếu ngang của nó {\displaystyle x\_{1}-x\_{0}} dài hơn hình chiếu đứng {\displaystyle y\_{1}-y\_{0}} (đường thẳng có [hệ số góc](https://vi.wikipedia.org/w/index.php?title=H%E1%BB%87_s%E1%BB%91_g%C3%B3c&action=edit&redlink=1) nhỏ hơn 1 và lớn hơn 0), hay góc nghiêng của đường thẳng so với phương ngang nhỏ hơn 45 độ

* **Code minh họa**

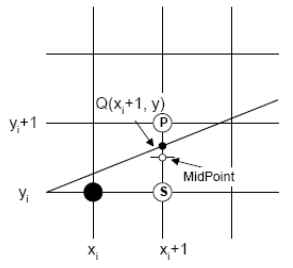




Hình 1a. Demo thuật toán Bresenham vẽ đường thẳng

### **b.Thuật toán Midpoint vẽ đoạn thẳng**

* Dù thuật toán Bresenham đã có bước chuyển mình thành công , đó là cải tiến việc tính toán trên số thực  phức tạp của thuật toán DDA thành chỉ thao tác trên số nguyên, trên phép cộng và nhân , làm tăng tốc thuật toán đáng kể .
* Nhưng việc xây dựng trường hợp tổng quát cho thuật toán Bresenham có vẻ phức tạp hơn nhiều so với thuật toán DDA. Vì vậy , vào năm 1967 , Pitteway đã công bố thuật toán MidPoint (được phát triển dựa trên thuật toán Bresenham) ,sau đó vào năm 1984  được Van Aken cải tiến.
* Thuật toán Midpoint cho kết quả hoàn toàn giống với thuật toán Bresenham , nhưng cách xây dựng thì “max simple” đơn giản hơn rất nhiều.



* Thuật toán Midpoint đưa ra cách chọn điểm yi+1là yihay yi+1 bằng cách so sánh điểm thực Q(xi+1 , y) với điểm Midpoint là trung điểm của S và P.
* Nếu điểm Q nằm dưới điểm Midpoint thì ta chọn điểm S là điểm vẽ tiếp theo. Ngược lại , nếu điểm Q nằm trên điểm Midpoint thì ta chọn P.
* Ta có dạng tổng quát của PT đường thẳng :

Ax + By +C=0

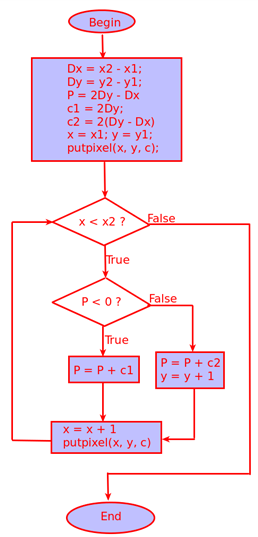
Với A= yo –y1; B= -(x2 –x1) ; C=x2y1–x1y2

Đặt F(x,y)= Ax+ By +C

* Lúc này việc chọn các điểm S, P ở trên được đưa về việc xét dấu của

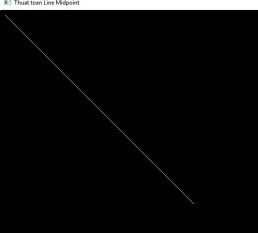
pi = 2F(Midpoint) = 2F(xi +1 , yi +1/2)

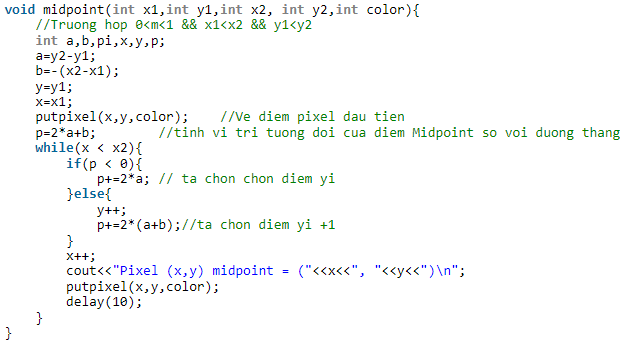
* Nếu pi < 0 ⇒ Midpoint nằm phía trên đường thẳng ⇒Lúc này điểm thực Q nằm phía dưới điểm Midpoint  ⇒ Chọn S(xi+1,yi) .
* Nếu pi >= 0 ⇒ Midpoint nằm phía dưới đường thẳng ⇒ Lúc này điểm thực Q nằm trên điểm Midpoint ⇒ Chọn P(xi+1,yi+1).
* **Lưu đồ thuật toán (tương tự Bresenham)**



Hình 1b. Sơ đồ thuật toán Midpoint

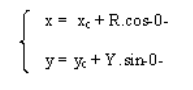
* **Code minh họa**



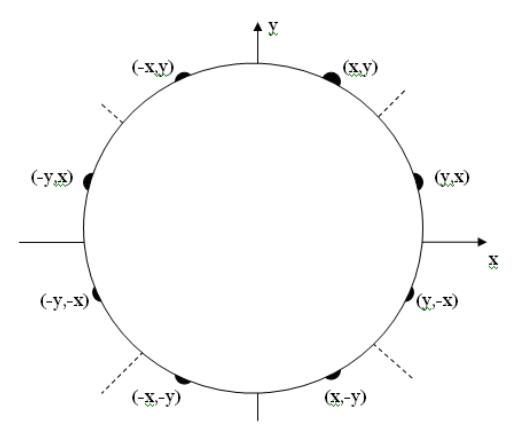


Hình 1b. Demo thuật toán Midpoint

### **c.Thuật toán Bresenham vẽ đường tròn**

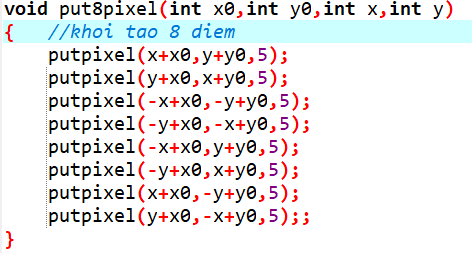
* Trong hệ tọa độ Descartes, phương trình đường tròn bán kính R có dạng:
  + Với tâm O(0,0) : x2 + y2 = R2
  + Với tâm C(xc,yc): (x - xc)2 + (y - yc)2 = R2
* Trong hệ tọa độ cực :
* Ý tưởng:

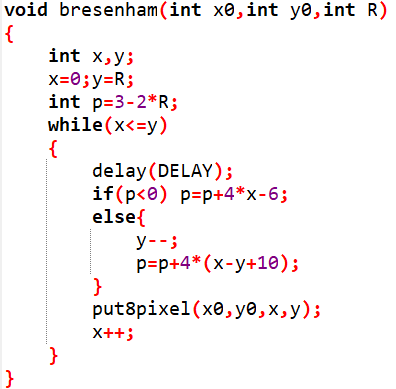
Do tính đối xứng của đường tròn C, nếu (x.y) thuộc (C) thì các điểm (y,x), (-y,x), (-x.y), (-x,-y), (-y,-x), (y,-x), (x,-y) cũng thuộc (C). Nên ta chỉ cần vẽ 1/8 cung tròn, sau đó lấy đối xứng qua 2 trục tọa độ và 2 đường phân giác thì ta vẽ được cả đường tròn.

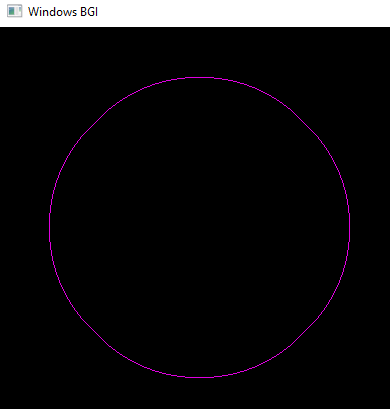


Hình 1c. Đường tròn và các diểm đối xứng

* **Code Minh họa**



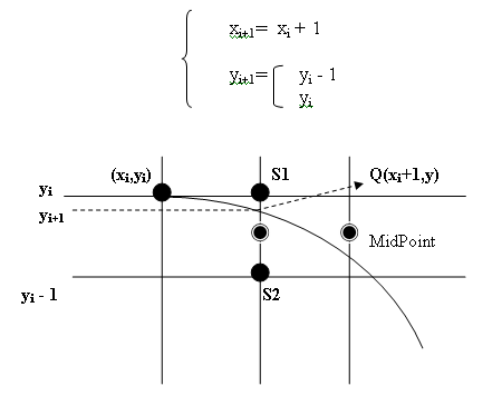




Hình 1c. Demo thuật toán Bresenham đường tròn

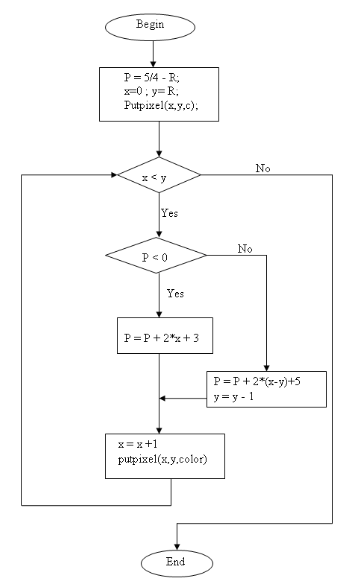
### **d.Thuật toán Midpoint vẽ đường tròn**

* Do tính đối xứng của đường tròn nên ta chỉ cần vẽ 1/8 cung tròn, sau đó lấy đối xứng là vẽ được cả đường tròn. Thuật toán MidPoint đưa ra cách chọn yi+1 là yi hay yi-1 bằng cách so sánh điểm thực Q(xi+1,y) với điểm giữa MidPoind là trung điểm của S1 và S2.
* Chọn điểm bắt đầu để vẽ là (0,R). Giả sử (xi, yi) là điểm nguyên đã tìm được ở bước thứ i, thì điểm (xi+1, yi+1) ở bước i+1 là sự lựa chọn giữa S1 và S2.

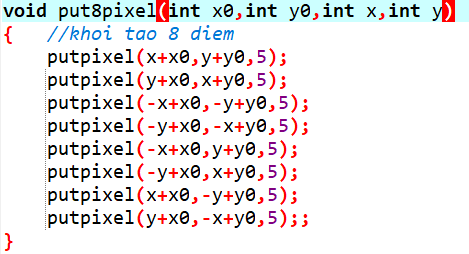


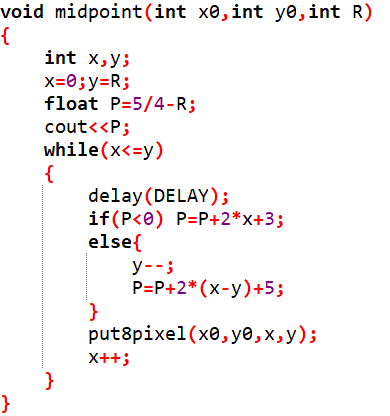
Hình 1d. Thuật toán Midpoint đường tròn

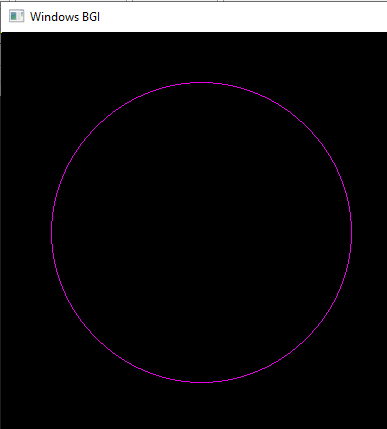
* **Lưu đồ thuật toán**



* **Code Minh họa**





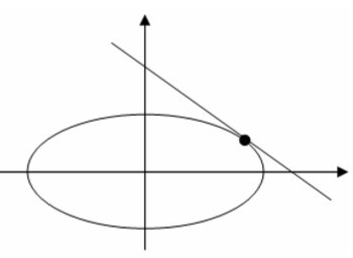


Hình 1d. Demo thuật toán Midpoint đường tròn

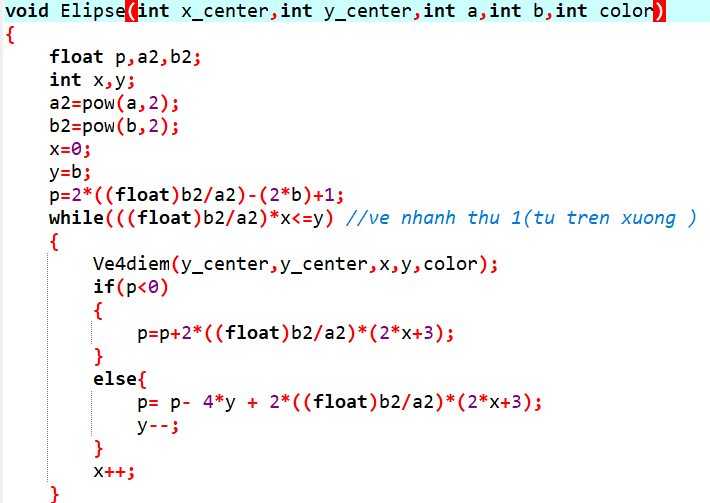
### **e.Thuật toán Bresenham vẽ Ellipse**

* **Ý tưởng:**
* Tính khoảng cách từ điểm thực đến 2 điểm vẽ được, xem khoảng cách nào ngắn hơn.
* Vẽ 1 nửa elip, sau đó lấy đối xứng qua các đường phân giác.

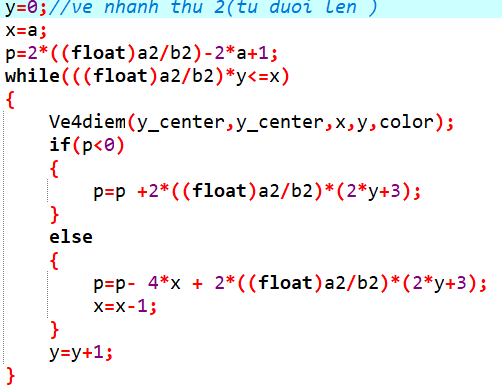
Đầu tiên, theo ý tưởng bài toán, chúng ta sẽ đi vẽ từng nửa elip. Ở mỗi nửa elip, chúng ta lại phải vẽ theo hai nhánh khác nhau. Đó là một nhánh từ trên xuống, và một nhánh từ dưới lên. Hai nhánh này cắt nhau tại giao điểm mà ở đó hệ số góc của tiếp tuyến với Elip bằng -1

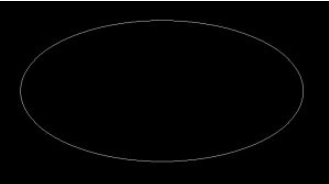


* Thực chất của việc chia hai nhánh vẽ elip là chúng ta đi tìm các điểm kế tiếp theo hướng tính y theo x (hay tính x theo y) cho trước.
* Khi x tăng lên 1 đơn vị thì y chỉ tăng lên rất ít ; và ngược lại với nhánh vẽ từ dưới lên: khi y tăng lên 1 đơn vị thì x chỉ tăng lên rất ít.
* Nếu như chúng ta chọn cách vẽ liền một mạch thì việc tăng lên không đồng đều của tọa độ x và y sẽ khiến đường elip không min, đứt gãy, rất xấu. Do đó, muốn vẽ được đường elip mịn nét, chúng ta sẽ phải chia elip thành 2 nhánh để vẽ.
* **Code Minh Họa**
* Vẽ Nhánh thứ nhất (trên xuống)



* Vẽ Nhánh thứ hai (từ dưới lên)





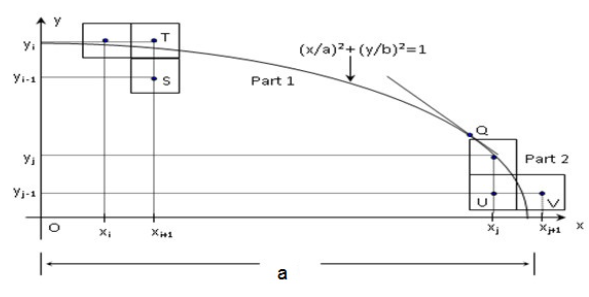
Hình 1e. Demo thuật toán Bresenham Ellipse

### **f.Thuật toán Midpoint vẽ Ellipse**

* **Ý tưởng:**
* Chia Elip làm 2 phần tại điểm Q nơi có hệ số góc của tiếp tuyến với Elip bằng -1 (vector gradient bằng 1). Tại vùng thứ nhất, x biến thiên nhanh hơn y và tại vùng thứ hai , y biến thiên nhanh hơn x. Nhớ lại công thức hệ số góc của đường cong :

dx/dy = fx/fy = (2b2x) /( 2a2y)

trong đó: fx và fy là đạo hàm riêng phần của f(x,y) theo x, theo y.



Hình 1f. Thuât toán Midpoint Ellipse

* Trong phần thứ nhất: Giả sử đã vẽ được điểm (xi,yi) , điểm tiếp theo trên elip được chọn trong bước nhảy i+1 là T hoặc S.  Trung điểm I của TS sẽ quyết định điểm nào được chọn. Giá trị của f(x,y) tại điểm I:

di = f(xi + 1, yi – ½) = b2(xi + 1)2 + a2(yi – ½)2 – a2b2

+ Nếu di ≥ 0 thì trung điểm I nằm ngoài đường elip  => điểm được chọn là S.

+ Nếu di < 0 thì trung điểm I nằm trong đường elip =>  điểm được chọn là T.

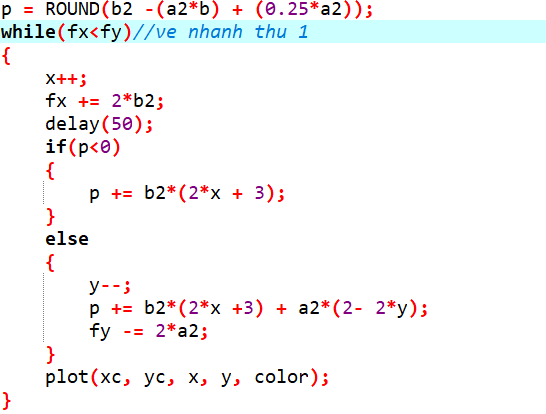
* Trong phần thứ hai: Chúng ta tính toán như phần 1. Giả sử ta phải xác định điểm  (xj+1,yj+1) tiếp theo trên elip  trong bước j+1. Điểm được chọn là U hoặc V. Trung điểm K của UV sẽ quyết định việc chọn điểm U hay điểm V. Giá trị của f(x,y) tại điểm K:

ej = f(xj + ½, yj – 1) = b2(xj + ½)2 + a2(yj – 1)2 – a2b2

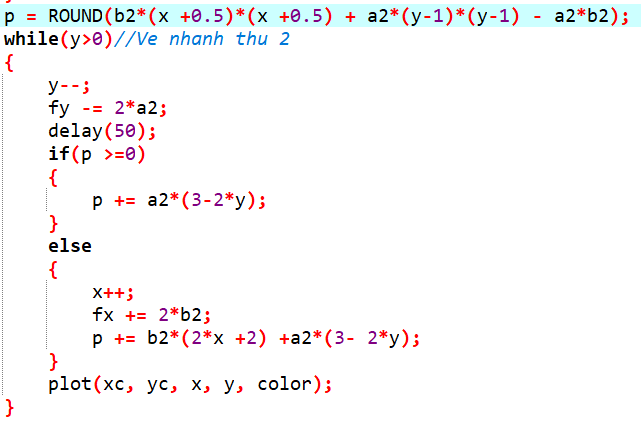
+ Nếu ej ≥ 0 điểm được chọn là U.

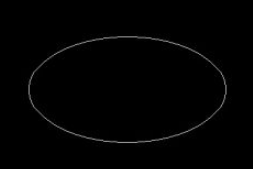
+ Nếu ej < 0 điểm được chọn là V.

* **Code Minh Họa**
* Vẽ Nhánh thứ nhất



* Vẽ Nhánh thứ hai





Hình 1f. Demo thuật toán Midpoint Ellipse

## **Bài 2: Phép biến đổi affine sử dụng DirectX, OpenGL**

* Bản chất của phép biến đổi hình học là thay đổi các mô tả về tọa độ của đối tượng, từ đó làm đối tượng thay đổi về hướng, kích thước, hình dạng.
* Có hai quan điểm về phép biến đổi hình học, đó là:

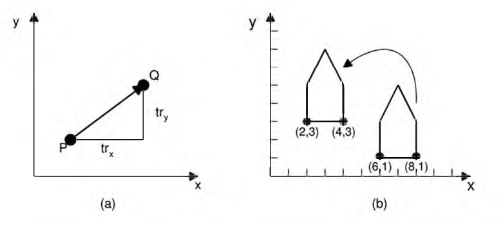
• Biến đổi đối tượng : thay đổi tọa độ của các điểm mô tả đối tượng theo một qui tắc nào đó.

• Biến đổi hệ tọa độ : tạo ra một hệ tọa độ mới và tất cả các điểm mô tả đối tượng sẽ được chuyển về hệ tọa độ mới.

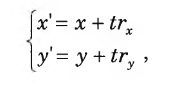
* Các phép biến đổi hình học cơ sở : tịnh tiến, quay, biến đổi tỉ lệ.

### **a.Phép Tịnh Tiến**

Phép tịnh tiến dùng để dịch chuyển đối tượng từ vị trí này sang vị trí khác.

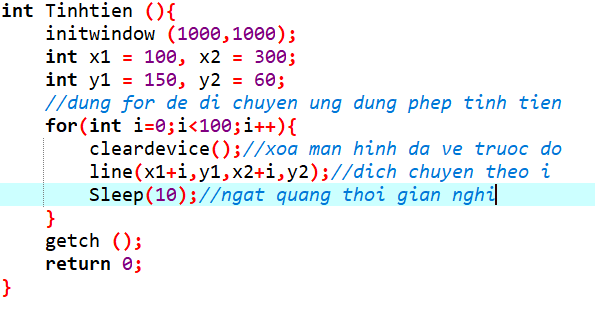


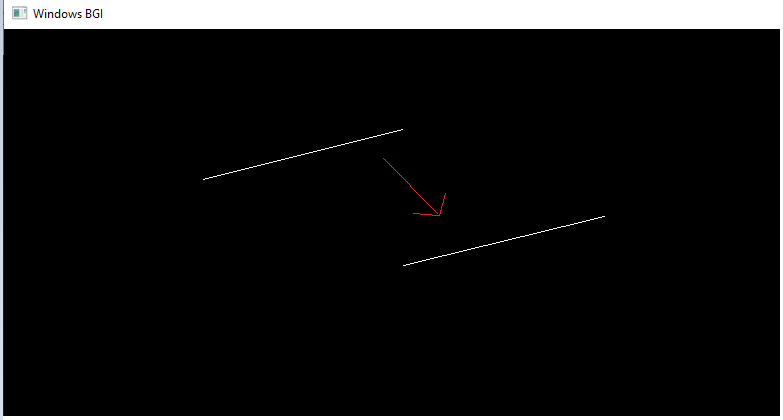
Nếu gọi tr, và tr, lần lượt là độ dời theo trục hoành và trục tung thì tọa độ của điểm mới Q(x', y') sau khi tịnh tiến điểm P(x, y) sẽ là :



(trx, try) được gọi là vector tịnh tiến hay vector độ dời.

* **Code Minh Họa (Phép tịnh tiến trên đường thẳng)**

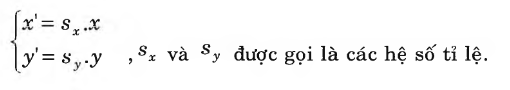




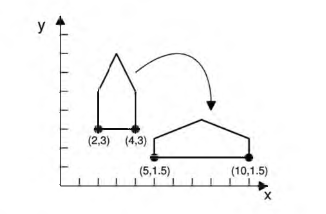
Hình 2a. Demo Phép tịnh tiến

### **b.Phép biến đổi tỉ lệ**

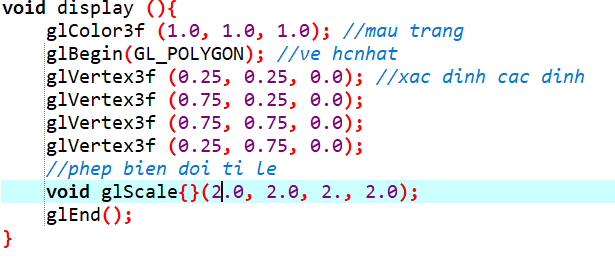
* Phép biến đổi tỉ lệ làm thay đổi kích thước đối tượng. Đế co hay giãn tọa độ của một điểm P(x, y) theo trục hoành và trục tung lần lượt là Sx và Sy, ta nhân Sx và 8y lần lượt cho các tọa độ của P.

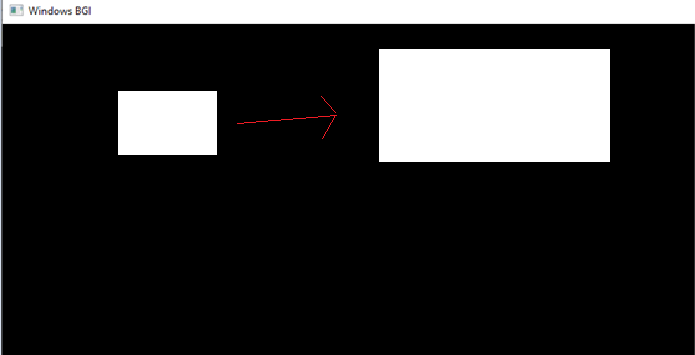


* Khi các giá trị Sx , Sy nhỏ hơn 1, phép biến đổi sẽ thu nhỏ đối tượng, ngược lại khi các giá trị này lớn hơn 1, phép biến đổi sẽ phóng lớn đối tượng.
* Khi Sx, Sy bằng nhau, ta gọi đó là phép đồng dạng (uniform scaling), phép đồng dạng là phép biến đổi bảo toàn tính cân xứng của đối tượng.



* Nhận xét rằng khi phép biến đổi tỉ lệ thu nhỏ đối tượng, đối tượng sẽ đưoc doi về gần gốc tọa độ hơn, tương tự khi phóng lớn đối tượng, đối tượng sẽ được dịch chuyển xa gốc tọa độ hơn.
* **Code Minh Họa: sử dụng thư viện OpenGL**

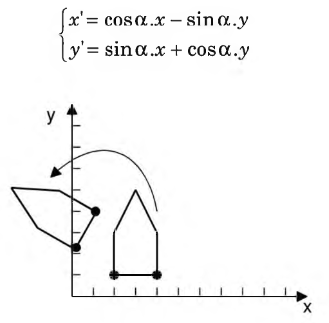




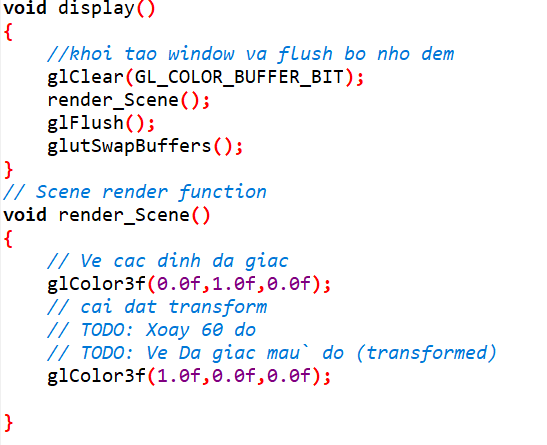
Hình 2b. Demo phép biến đổi tỉ lệ

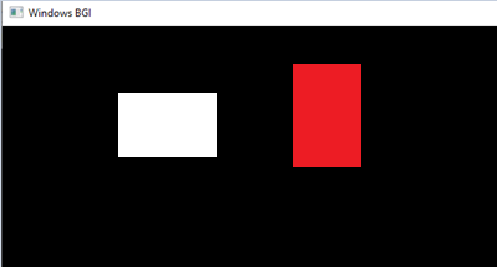
### **c.Phép Xoay**

* Phép quay làm thay đổi hướng của đối tượng.
* Một phép quay đòi hỏi phải có tâm quay, góc quay. Góc quay dương thường được quy ước là chiều ngược chiều kim đồng hồ. Ta có công thức biến đổi của phép quay điểm P(x, y) quanh gốc tọa độ một góc alpha:



* **Code Minh Họa: sử dụng thư viện OpenGL**

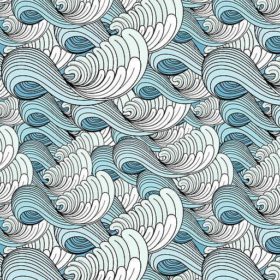




Hình 2c. Demo Phép Xoay

## **Bài 3: Texture Vật thể**

* **Texture** là một yếu tố không thể thiếu trong việc quyết định sự thành công của một thiết kế. Một texture bắt mắt, gây ấn tượng sắc nét, sẽ là điểm nhấn vô cùng nổi bật cho thiết kế
* **Texture** có thể được nhìn thấy rất nhiều trong thế giới tự nhiên: bất kì vật gì có bề mặt thì đều có texture theo một tỷ lệ nhất định. Chúng ta có thể quan sát texture ở cả các vật thể tự nhiên hay giả tự nhiên (nhân tạo) như texture trên gỗ, cây, kim loại và da.
* Nói chung, texture liên quan đến các đặc điểm bề mặt của một vật thể và có kích thước, hình dáng, mật độ, sự sắp xếp, tỷ lệ các thành phần cơ bản của vật thể. Một **texture** thường được miêu tả là mịn hay gồ ghề, mềm hay cứng, thô hay bóng vv.
* Theo thống kê chung cho thấy **texture** bao gồm 2 loại cơ bản nhất bao gồm: **Visual Texture** và **Tactile Texture**. Thông tin từng loại được cập nhật cụ thể như sau:
* **Tactile Texture**: Tactile được hiểu là chạm vào và tactile texture có nghĩa là sự gồ ghể của bất kỳ một bề mặt nào đó mà khi chúng ta chạm vào có thể cảm thấy được. Đây được xem là một trong những yếu tố quan trọng nhất trong thiết kế 3D so với 2D.
* **Visual texture**: Ấn tượng có thể nhìn thấy được mà **texture** mang lại cho người xem, như về màu sắc, xu hướng hay có thể là mật độ trong trên một bức ảnh. **texture** nó liên quan trực tiếp đến những ảo ảnh của một kết cấu bề mặt. Nó chính là hình dạng của **tactile texture**ở bề mặt 2D.



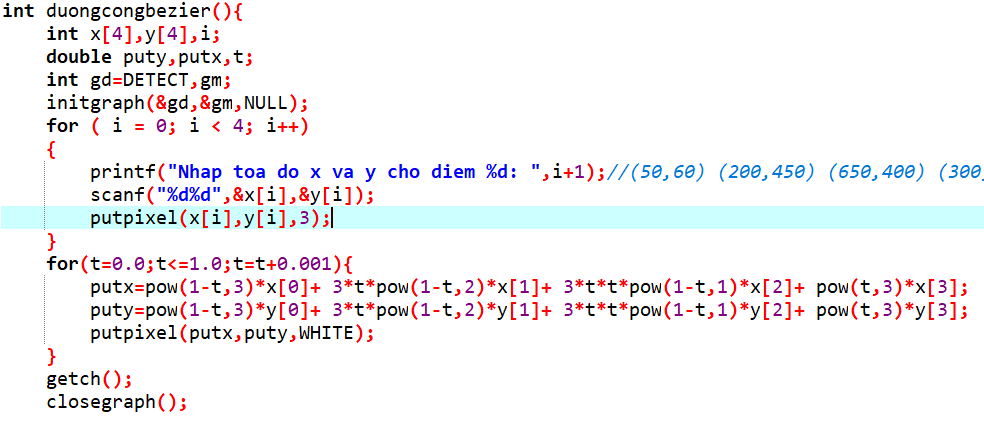
Hình 3. Mẫu Tactile và Visual Texture

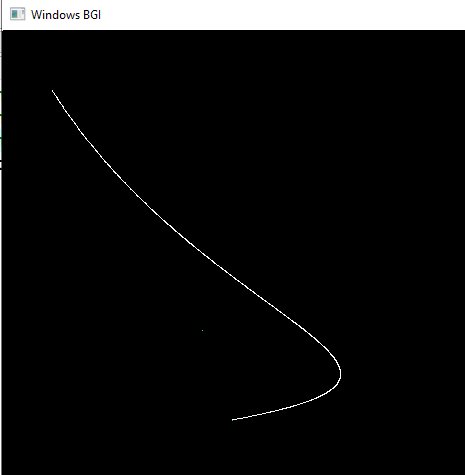
* **Demo Texture:** sử dụng OpenGL tạo hộp (tham khảo các ví dụ đồ họa 3D MS)



## **Bài 5: Vẽ đường cong và mặt cong Bezier**

* **Đường cong Bézier** là một [đường cong tham số](https://vi.wikipedia.org/w/index.php?title=%C4%90%C6%B0%E1%BB%9Dng_cong_tham_s%E1%BB%91&action=edit&redlink=1) thường được sử dụng trong [đồ họa máy tính](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_h%E1%BB%8Da_m%C3%A1y_t%C3%ADnh) và một số lĩnh vực khác. Dạng tổng quát hóa của đường cong Bézier trong [không gian nhiều chiều](https://vi.wikipedia.org/wiki/Kh%C3%B4ng_gian_nhi%E1%BB%81u_chi%E1%BB%81u) được gọi là [mặt phẳng Bézier](https://vi.wikipedia.org/w/index.php?title=M%E1%BA%B7t_ph%E1%BA%B3ng_B%C3%A9zier&action=edit&redlink=1), trong đó [tam giác Bézier](https://vi.wikipedia.org/w/index.php?title=Tam_gi%C3%A1c_B%C3%A9zier&action=edit&redlink=1) là một trường hợp đặc biệt.
* Đường cong Bézier được công bố lần đầu vào năm [1962](https://vi.wikipedia.org/wiki/1962) bởi một kỹ sư [người Pháp](https://vi.wikipedia.org/wiki/Ng%C6%B0%E1%BB%9Di_Ph%C3%A1p) [Pierre Bézier](https://vi.wikipedia.org/wiki/Pierre_B%C3%A9zier), người sử dụng nó để thiết kế thân [ôtô](https://vi.wikipedia.org/wiki/%C3%94t%C3%B4). Nhưng việc nghiên cứu những đường cong này thực tế đã bắt đầu từ năm 1959 bởi nhà toán học [Paul de Casteljau](https://vi.wikipedia.org/w/index.php?title=Paul_de_Casteljau&action=edit&redlink=1), ông sử dụng [giải thuật De Casteljau](https://vi.wikipedia.org/w/index.php?title=Gi%E1%BA%A3i_thu%E1%BA%ADt_De_Casteljau&action=edit&redlink=1) để đánh giá các đường cong đó.
* Về mặt ứng dụng, đường cong Bézier thường được sử dụng trong [đồ họa vector](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_h%E1%BB%8Da_vector) để mô hình hóa các đường cong mượt (smooth curves) và những đường cong đó có thể được phóng to hoặc thu nhỏ theo một tỉ lệ không giới hạn.
* Đường cong Bézier còn thường được sử dụng như là một công cụ để điều khiển sự chuyển động (animation).
* **Ý tưởng:**
* Một đường cong bezier được xác định dựa trên một tập các điểm kiểm soát (control point) P0 tới Pn với n được gọi là cấp bậc của đường cong. Với n = 1 thì nó chính là cấp tuyến tính, khi vẽ ra nó là các đường thẳng nối nhau. Với n = 2 thì đường cong được gọi là đường cong bậc 2.
* Giá trị n càng cao thì bậc càng cao và độ mịn càng cao, tức là số điểm control tăng lên, độ chính xác để xác định đường curve đó càng chính xác, vì vậy mà độ mịn nó sẽ tăng theo.
* **Demo đường cong Bezier:**





**Hình 5. Demo Bezier**

# TÀI LIỆU THAM KHẢO

[1]. Nguyễn Đình Cường: [computer graphics 2022\_Nha Trang University](https://elearning.ntu.edu.vn/pluginfile.php/596791/mod_forum/attachment/131669/computer%20graphics%202022_Nha%20Trang%20University.rar?forcedownload=1)

[2]. D3d12book-graphic: drive.google.com/file/d/1k2Dk2T5xbzqwlNrdXModipasbzJdqwG9/view

[3]. Learn OpenGL: <https://learnopengl.com/Getting-started/Textures>

[4]. DirectX Microsoft: https://github.com/microsoft/DirectX-Graphics-Samples