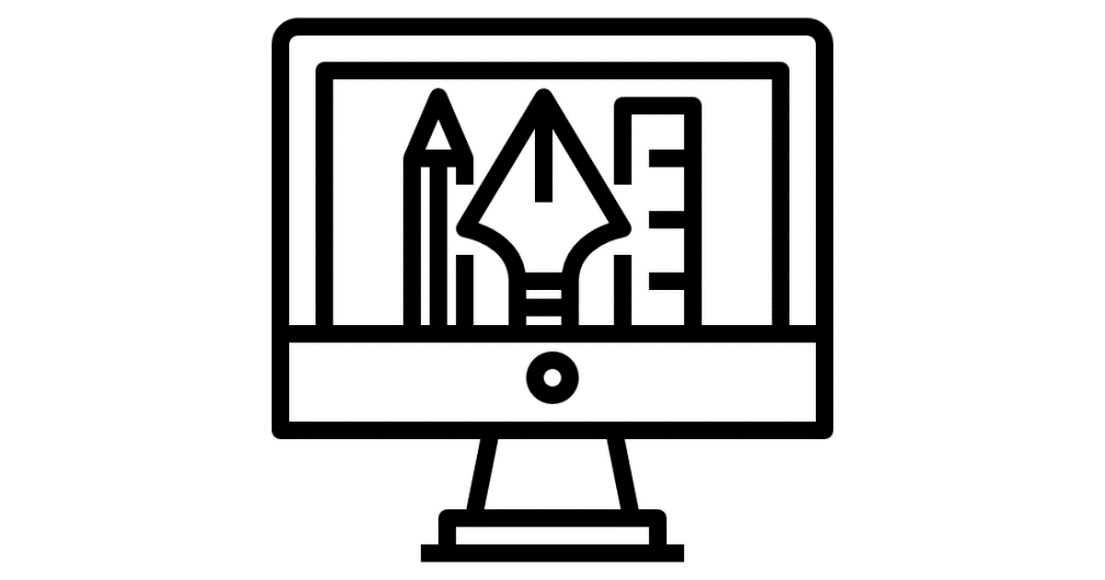
**ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

--🙢🕮🙠--

**BÁO CÁO MÔN HỌC**

**KĨ THUẬT ĐỒ HỌA**



***-*** Nha Trang  ***-***

***-*** 2021  ***-***

**PHỤ LỤC**

[1. Các thuật toán cơ bản và các phép biến đổi hình học cơ bản 4](#_Toc73239045)

[**1.1. Thuật toán DDA - Digital Differential Analyzer** 4](#_Toc73239046)

[**a.Vẽ đường thẳng** 5](#_Toc73239047)

[**b.Vẽ đường tròn** 6](#_Toc73239048)

[**1.2. Thuật toán Midpoint** 8](#_Toc73239049)

[**a.Vẽ đường thẳng** 8](#_Toc73239050)

[**b.Vẽ đường tròn** 11](#_Toc73239051)

[**1.3. Thuật toán Bresenham** 13](#_Toc73239052)

[**a.Vẽ đường thẳng** 14](#_Toc73239053)

[**b.Vẽ đường tròn** 14](#_Toc73239054)

[**1.4 Phép tịnh tiến** 16](#_Toc73239055)

[**1.5. Phép xoay** 18](#_Toc73239056)

[**1.6. Phép tỉ lệ** 20](#_Toc73239057)

[2. Chạy các chương trình demo đồ họa bằng OpenGL 21](#_Toc73239058)

[**2.1. Vẽ hình học 3D với OpenGL** 22](#_Toc73239059)

[**2.2. Mô phỏng hệ Mặt Trời với OpenGL** 23](#_Toc73239060)

[**2.3 Đọc file OBJ sử dụng OpenGL** 26](#_Toc73239061)

[3. Đường cong Bezier và mặt cong Bezier 28](#_Toc73239062)

[**3.1. Đường cong Bezier** 28](#_Toc73239063)

[**3.2. Mặt cong Bezier** 30](#_Toc73239064)

[4. Demo đồ hoạ với Java 34](#_Toc73239065)

**THÔNG TIN BÁO CÁO**

Báo cáo môn học: Kĩ Thuật Đồ Hoạ

Giáo viên bộ môn: Nguyễn Đình Cường

Họ và tên sinh viên: Nguyễn Văn Hải Long

MSSV: 60136035

**CHI TIẾT BÁO CÁO**

# 1. Các thuật toán cơ bản và các phép biến đổi hình học cơ bản

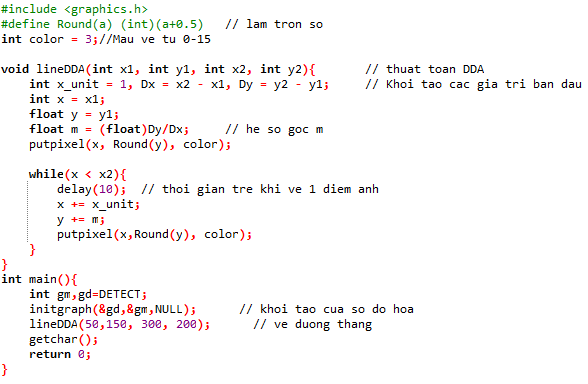
## **1.1. Thuật toán DDA - Digital Differential Analyzer**

Thuật toán DDA - Digital Differential Analyzer (bộ phân tích vi sai) có thể tóm tắt qua các bước một cách tổng quan:

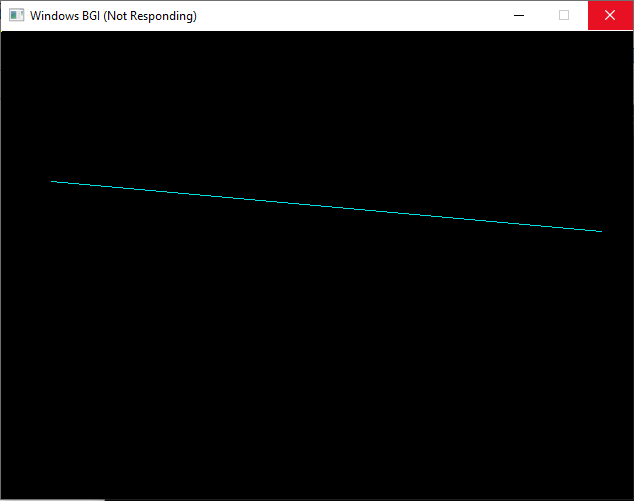
* Bước 1: Giả sử tọa độ của hai điểm A(xA, yA), B(xB, yB) không trùng nhau với điều kiện là xA, yA, xB, yB đều là số nguyên.
* Bước 2: Tính số điểm ảnh của đường thẳng được vẽ thêm trên màn hình.
* So sánh trị tuyệt đối của dx = xB - xA, dy = yB - yA và lấy trị tuyệt đối lớn nhất vì xác định càng nhiều số giao điểm thì đường thẳng càng rõ và mịn.
* Gọi là steps là số điểm ảnh được vẽ thêm, khi đó steps = max(|dx|, |dy|).
* Bước 3: Sau khi xác định được số giao điểm steps sẽ xác định được giá trị cộng vào cho x và y bắt đầu từ tọa độ A(xA, yA) cho tới tọa độ điểm B(xB, yB). Các giá trị này sẽ được tính như sau: x\_inc = dx / steps, y\_inc = dy / steps, giá trị là số thực.
* Bước 4: Sau khi có tất cả các thông số cần thiết bao gồm A(xA, yA), B(xB, yB), steps, x\_inc, y\_inc, tiến hành bước cuối cùng là tìm các tọa độ cần vẽ. Chỉ cần cộng xA, yA với x\_inc, y\_inc theo công thức xi+1 = xi + x\_inc, yi+1 = yi +y\_inc. Sau đó làm tròn kết quả về số nguyên để ra các tọa độ tiếp theo.

### **a.Vẽ đường thẳng**

Triển khai chương trình:

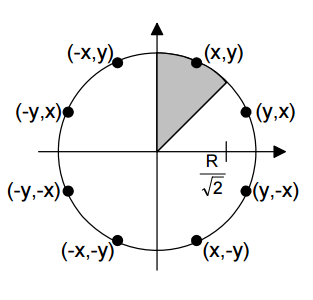


Kết quả chương trình:



### **b.Vẽ đường tròn**

Sử dụng thuật toán DDA xác định 8 điểm của đường tròn và góc giữa hai đường thẳng liền kề:



Xác định 8 điểm:

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

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

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

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

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

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

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

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

Triển khai thuật toán:

void DDA(int xc, int yc, int r)

{

int x = 0, y = r;

drawcircle(x, y, xc, yc);

while(x<=y)

{

x++;

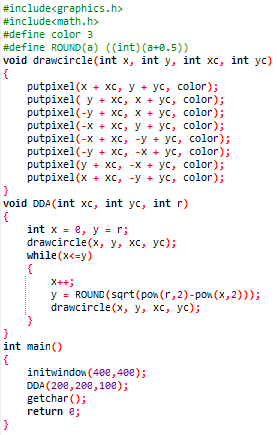
y = ROUND(sqrt(pow(r,2)-pow(x,2)));

drawcircle(x, y, xc, yc);

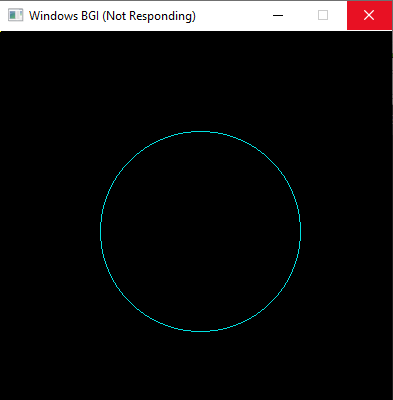
}

}

Triển khai chương trình:



Kết quả chương trình:



## **1.2. Thuật toán Midpoint**

Thuật toán Midpoint (trung điểm) có thể mô tả như sau:

Đối với việc lựa chọn điểm vẽ tiếp theo sau A, có hai lựa chọn đó là điểm P và Q với M là trung điểm giữa chúng. Việc quyết định chọn điểm nào để vẽ tiếp theo phụ thuộc vào vị trí của M so với đường thẳng đang vẽ. Nếu M nằm phía dưới đường thẳng, chọn điểm Q. Ngược lại, nếu M nằm phía trên đường thẳng ta chọn điểm P.

Các trường hợp xét dưới đây với 0 < m < 1.

### **a.Vẽ đường thẳng**

Phương trình đường thẳng là:

y = m \* x + b, m = dy / dx

⇔ y = dx / dx \* x + b

⇔ dy \* x + (-dx \* y) + b \* dx = 0 (1)

Mà:

F(x, y) = a \* x + b \* y + c = 0 (2)

(1), (2) ⇒ a = dy, b = -dx, c = b \* dx

Tìm giá trị d

F(M) = F(xi + 1, yi + 1 / 2)

= F(xi, yi) + a + b / 2 = 0 + dy + dx / 2 = dy + dx / 2

Tìm giá trị dmax

Đặt d = F(M)

* Nếu d > 0, chọn điểm Q để vẽ.
* Nếu d < 0, chọn điểm P để vẽ.

Trường hợp 1 - chọn Q

Mnew \* (xi + 2, yi + 1 / 2) ⇒ dnew = F(xi + 2, yi + 1 / 2)

(Δd)Q = dnew - dold

= F(xi + 1, yi + 1 / 2) - F(xi + 2, yi + 1 / 2) = a

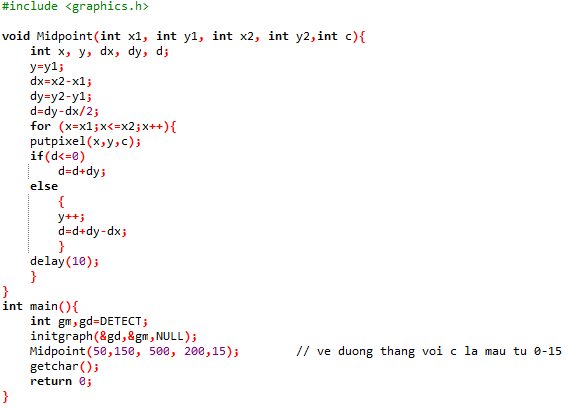
= dy

⇒ dnew = dold + dy

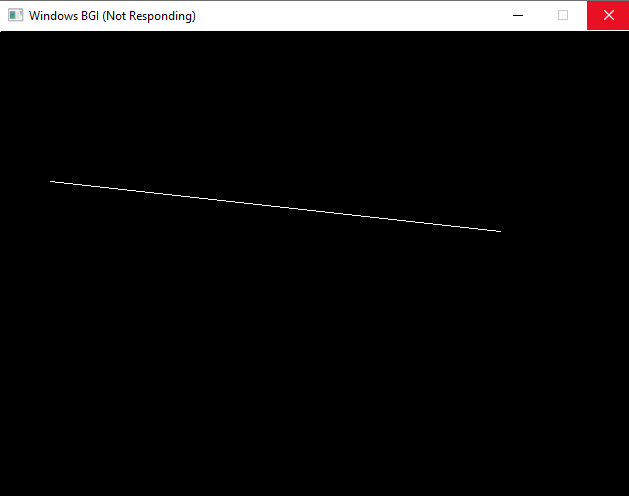
Trường hợp 2 - chọn P

Mnew \* (xi + 2, yi + 3 / 2) ⇒ dnew = F(xi + 2, yi + 3 / 2)  
(Δd)P = dnew - dold= F(xi + 1, yi + 1 / 2) - F(xi + 2, yi + 3 / 2) = a + b  
= dy - dx  
⇒ dnew = dold + dy - dx

Triển khai chương trình:



Kết quả chương trình:



### **b.Vẽ đường tròn**

Đường tròn có tâm O(xc, yc) = (0, 0), bán kính r có phương trình:

x2 + y2 = r2 => x2 + y2 - r2 = 0

Đặt f(x, y) = x2 + y2 - r2

Với mọi điểm P(x, y) nằm trong hệ tọa độ Oxy:

* P(x, y) nằm trên đường tròn O nếu f(x, y) = 0
* P(x, y) nằm ngoài đường tròn O nếu f(x, y) > 0
* P(x, y) nằm trong đường tròn O nếu f(x, y) < 0

Sau đó vẽ 8 điểm đối xứng. Giả sử đã vẽ được (Xi, Yi) ở bước thứ i, cần xác định (Xi+1, Yi+1) ở bước thứ i + 1.

Xi+1 = Xi+1  
Yi+1 ∈ {Yi, Yi - 1}

Tính Fi

Đặt Fi =  F(X, Y - 1/2):

F(Xi + 1, Yi - 1/2) = (Xi + 1)2 + (Yi - 1/2)2 - R2  
Fi = Xi2 + 2Xi + Yi2 - Yi + 5/4 - R2

Nếu Fi < 0 ⇔ (Xi+1, Y) gần với Yi ⇒ Yi+1 = Yi

Nếu Fi ≥ 0 ⇔ (Xi+1, Y) gần với Yi -1 ⇒ Yi+1 = Yi-1

Tính Fi +1 theo Fi

Fi+1 - Fi = 2Xi + 3 + (Yi + 12 - Yi2)+ (Yi+1 - Yi)  (\*)

Nếu Fi < 0 thì Fi+1 = Fi + 2Xi + 3, do thay thế Yi+1 = Yi  vào (\*)

Nếu Fi ≥ 0 thì Fi+1 = Fi + 2(Xi - Yi) + 5, do thay thế Yi+1 = Yi-1 vào (\*)

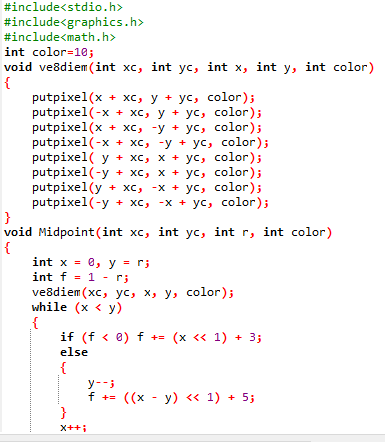
Tính giá trị F đầu tiên

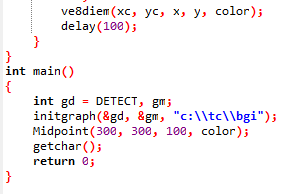
F(Xi + 1, Yi - 1/2) = (Xi + 1)2 + (Yi - 1/2)2 - R2  
Fi = Xi2 + 2Xi + Yi2 - Yi + 5/4 - R2

Thay Xi = 0 và Yi = R trong công thức trên:

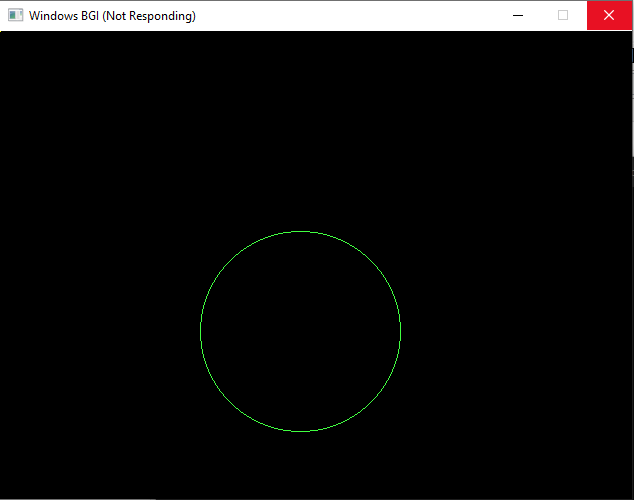
F = 5/4 - R

Triển khai chương trình:





Kết quả chương trình:



## **1.3. Thuật toán Bresenham**

Mô tả thuật toán Bresenham: thuật toán đưa ra cách chọn yi+1 là yi hay yi+1 theo một hướng khác (giảm thời gian thực hiện hơn so với DDA). 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.

Xét trường hơp 0<m<1

Gọi y là giá trị thực (giá trị chính xác) của đường thẳng tại x ở bước thứ i+1 .

y = m(xi+ 1)+b

Gọi d1 là khoảng cách từ y đến yi .

Gọi d2 là khoảng cách từ y đến yi+1 .

Ta có:

d1 = y - yi = m(xi + 1) + b - yi

d2 = yi + 1 - y = yi + 1 - [m(xi + 1) + b]

Ta xét (d1 - d2 ):

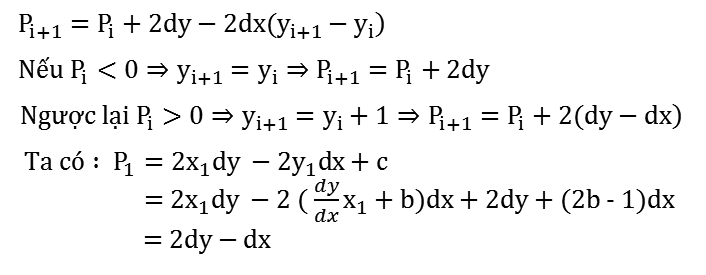
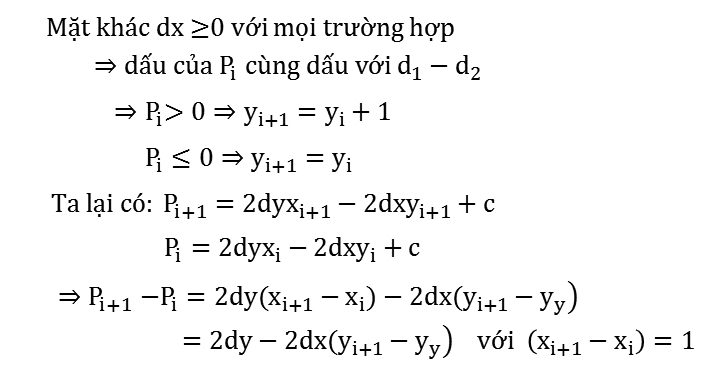
d1 - d2 > 0 => d1 > d2 => yi + 1

Ngược lại: d1 - d2 ≤ 0 => d1 ≤ d2 => yi+1 = yi

* d1 - d2 = [ m(xi +1) + b - yi] - [ yi +1 - m(xi +1) -b]

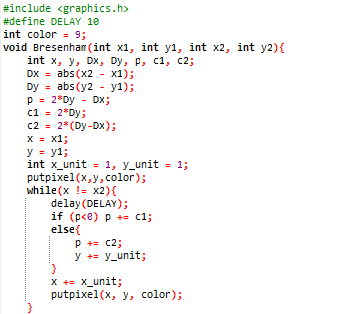
= m(xi +1) +b - yi - yi -1 + m(xi +1) +b

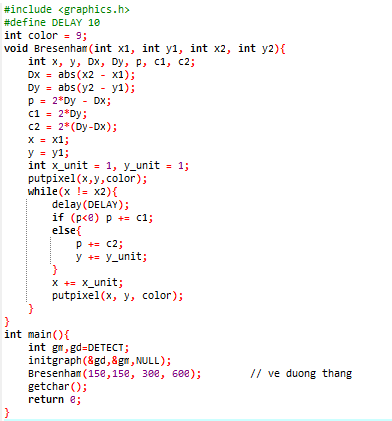
= 2m(xi +1) - 2yi + 2b - 1

Dễ thấy d1 - d2 tồn tại phép toán với số thực m = dy/dx .Và để tuân thủ theo đúng ý tưởng thuật toán chỉ thực hiện các phép toán trên số nguyên, ta khử phân số ( triệt tiêu mẫu số) bằng cách nhân 2 vế với dx:

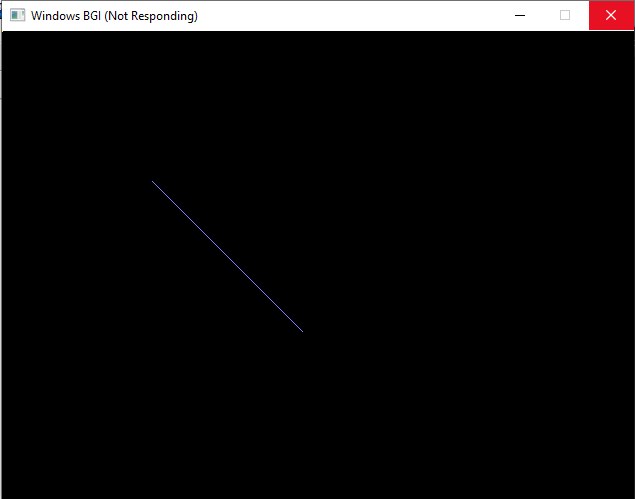
### **a.Vẽ đường thẳng**

Phát triển chương trình:





Kết quả chương trình:



### **b.Vẽ đường tròn**

Bước 1:

-       Chọn điểm đầu cần vẽ (x,y) = (0,R).

-       Tính p đầu tiên: p = 3 - 2R.

-       Vẽ 8 điểm ứng với (x,y).

Bước 2:

-       Tăng x lên 1 pixel: x = x + 1.

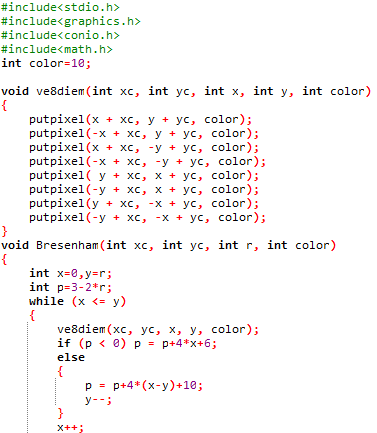
-       Nếu p < 0: p = p + 4x + 6.

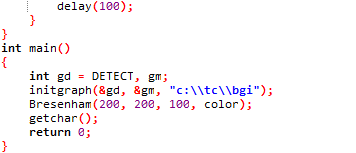
Ngược lại: p = p + 4(x - y) + 10 và y = y – 1.

-       Vẽ điểm 8 điểm ứng với (x,y) mới.

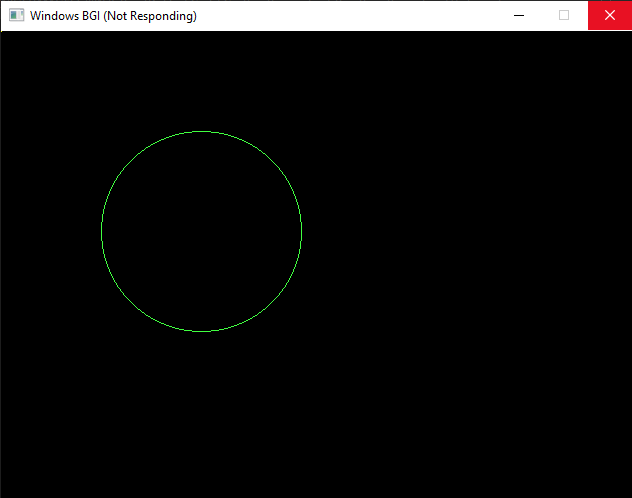
Bước 3: Lặp lại bước 2 cho đến khi x = y.

Triển khai chương trình:

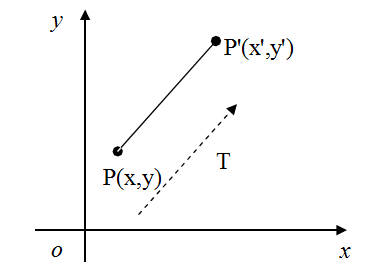




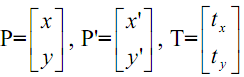
Kết quả chương trình:



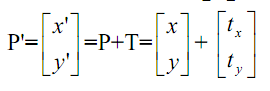
## **1.4 Phép tịnh tiến**



Nếu ta kí hiệu tọa độ các điểm và véc tơ như sau:



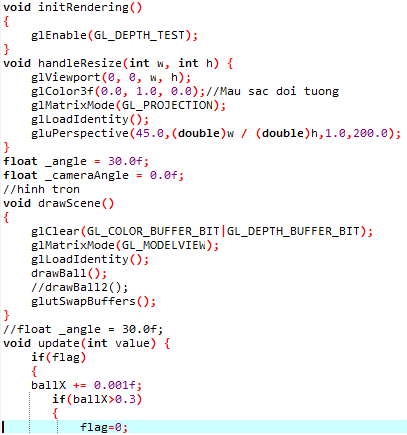
khi đó phép tịnh tiến được mô tả bởi phương trình:

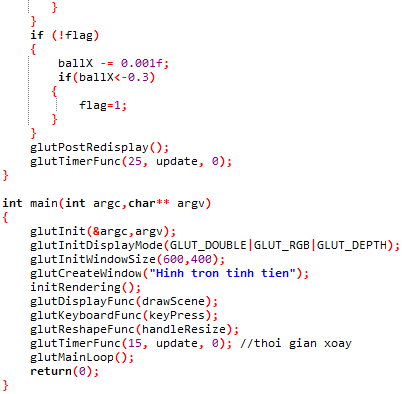


Phép tịnh tiến không làm thay đổi hình dạng của vật thể.

Triển khai chương trình:





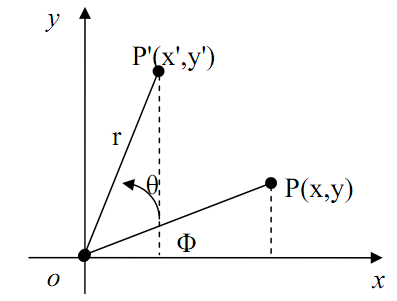


Kết quả chương trình:



## **1.5. Phép xoay**

Trong không gian hai chiều ta xét phép quay vật thể quanh tâm quay I(xr,yr) với góc quay θ (θ > 0 nếu chiều quay ngược chiều kim đồng hồ và θ < 0 nếu chiều quay cùng chiều kim đồng hồ).



Ta có các phương trình biến đổi sau:

x'= r cos(Φ+θ)= r cosΦcosθ- r sinΦsinθ

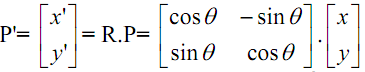
y'= r sin(Φ+θ)= r cosΦsinθ+ r sinΦcosθ

Mặt khác ta lại có x=r cosΦ và y=r sinΦ. Thay vào hai phương trình trên ta có:

x'= x cosθ- y sinθ

y'= x sinθ+ y cosθ

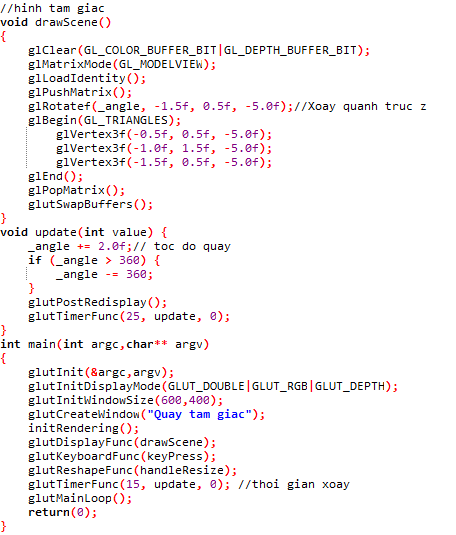
Do đó phép quay được mô tả bởi phương trình:



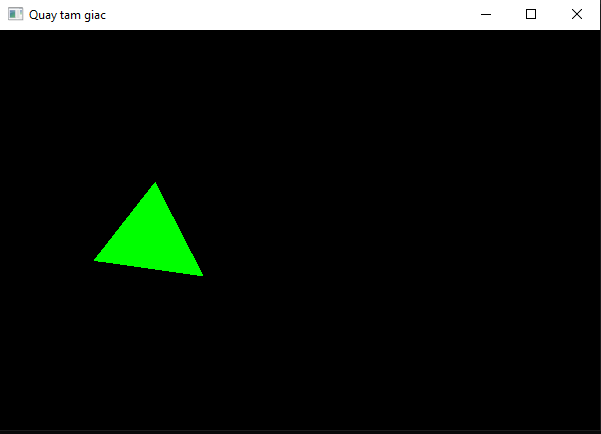
Phép quay cũng giống như phép tịnh tiến không làm thay đổi hình dáng của vật thể.

Triển khai chương trình:



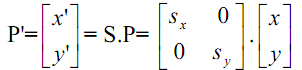


Kết quả chương trình:



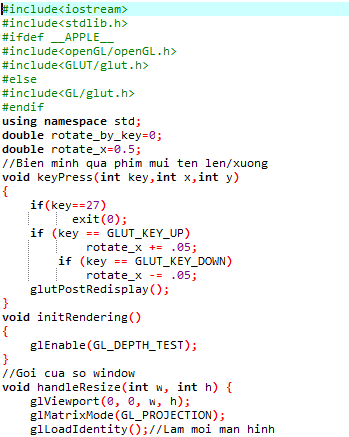
## **1.6. Phép tỉ lệ**

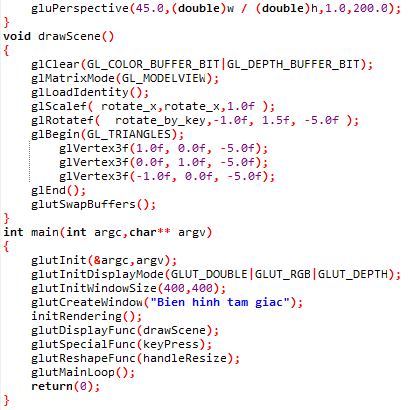
Phép tỉ lệ sẽ nhân hoành độ và tung độ ban đầu với hệ số tỉ lệ sx và sy tương ứng, cụ thể ta sẽ có x'= x.sx và y'= y.sy. Phương trình của phép biến đổi tỉ lệ có thể được mô tả như sau:



Khi (sx, sy)≠(1, 1) phép biến đổi tỉ lệ sẽ làm thay đổi hình dáng của vật thể

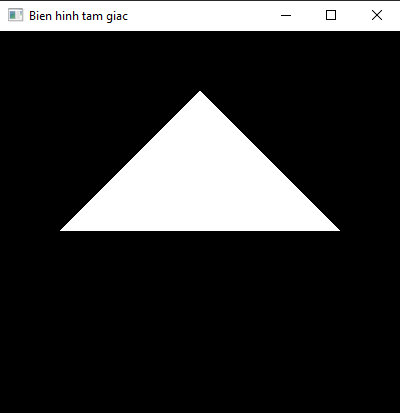
Triển khai chương trình:





Kết quả chương trình:

Ghi chú: Sử dụng phím mũi tên lên – xuống để điều chỉnh tỉ lệ hình, khi hình thu nhỏ hết mức sẽ đổi chiều hình ảnh (tương ứng là phép đối xứng)

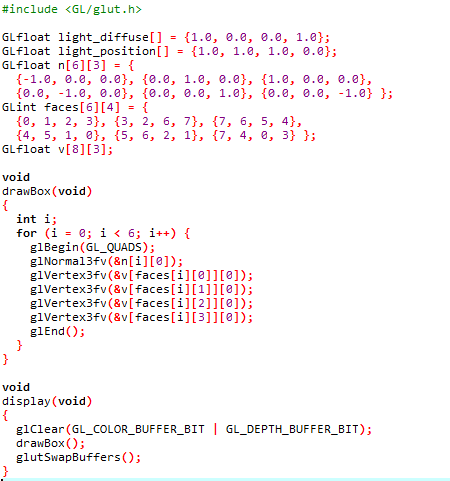


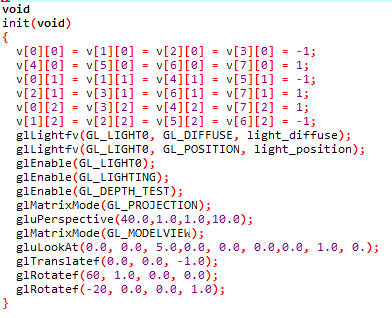
# 2. Chạy các chương trình demo đồ họa bằng OpenGL

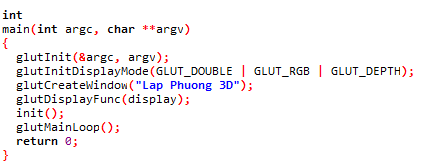
OpenGL (Thư viện đồ họa mở) là một API tiêu chuẩn đa nền tảng, tăng tốc phần cứng, không phụ thuộc vào ngôn ngữ để kết xuất đồ họa 3D (bao gồm cả 2D). Máy tính hiện đại có GPU chuyên dụng (Bộ xử lý đồ họa) với bộ nhớ riêng để tăng tốc độ kết xuất đồ họa. OpenGL là giao diện phần mềm với phần cứng đồ họa. Nói cách khác, các lệnh kết xuất đồ họa OpenGL do các ứng dụng của bạn đưa ra có thể được chuyển hướng đến phần cứng đồ họa và được tăng tốc.

## **2.1. Vẽ hình học 3D với OpenGL**

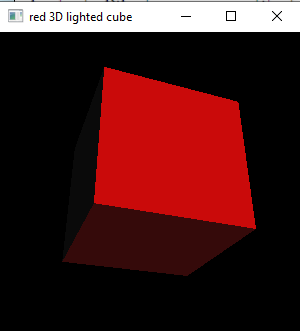
Triển khai chương trình:





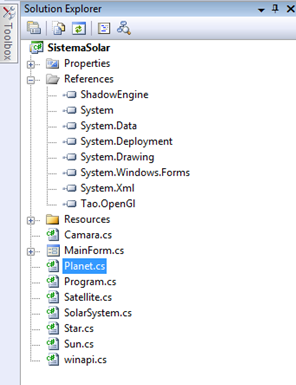


Kết quả chương trình:



## **2.2. Mô phỏng hệ Mặt Trời với OpenGL**

Sử dụng thư viện OpenGL kết hợp .Net Framework và ngôn ngữ C# mô phỏng hệ Mặt Trời (soure code tham khảo). Ứng dụng dạng Winform với các chức năng thu phóng, cùng những class C# sau:



**Camara.cs**

Đây là một máy ảnh dạng FPS (góc nhìn thứ nhất) cổ điển. Giải thích về cách hoạt động của FPS vượt ra ngoài phạm vi của bài viết này. Chúng hoạt động theo cách sau:

* + Con trỏ chuột được ghim chính giữa màn hình.
  + Khi người dùng di chuyển chuột, delta X và delta Y được tính từ điểm đầu.
  + Delta X và delta Y được chuyển thành các góc và cách quay của máy ảnh.
  + Khi người dùng muốn tiến hoặc lùi, máy ảnh sẽ di chuyển theo hướng là góc trỏ.
  + Người đọc có thể xem qua public void Update(int pressedButton) tại lớp camera để hiểu rõ hơn.

**MainForm.cs**

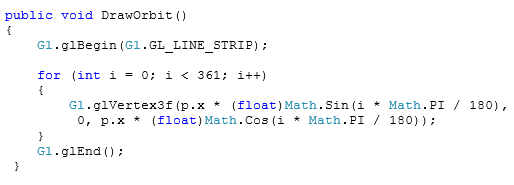
Lớp này là code chính và duy nhất của dự án. Nó chứa lệnh gọi tải kết cấu, khởi tạo bối cảnh 3D, bản vẽ nội dung 3D, v.v. Nó cũng xử lý phím nhập vào từ người dùng và đầu vào của chuột. Vì nội dung 3D yêu cầu vẽ ít nhất 30 khung hình / giây nên tác giả đã sử dụng bộ đếm thời gian và đặt tất cả câu lệnh của bản vẽ bên trong lệnh đó.

**Planet.cs**

Một hành tinh chứa các biến sau:

* Toạ độ
* Kết cấu (lưu dạng ảnh trong \bin\Debug\texturas\)
* Quỹ đạo quay (khoảng cách hiện tại tính từ mặt trời)
* Góc quay hiện tại
* Tốc độ di chuyển trên quỹ đạo hiện tại

Tác giả đã sử dụng OpenGL mặt cong bậc 2 để vẽ hình cầu mô phỏng hành tinh. Quadrics(mặt cong bậc 2) là các hình dạng được xác định trước OpenGL để trợ giúp trong các tác vụ vẽ nhỏ. Trong mỗi khung, hành tinh chuyển động trên quỹ đạo của nó theo tốc độ di chuyển. Ngoài ra, có một biến bool được gọi là hasMoon để chỉ định xem người dùng có muốn xem bản vẽ mặt trăng cho hành tinh đó hay không. Tác giả chỉ có mặt trăng của Trái Đất nhưng nếu người dùng thích, ví dụ, để vẽ mặt trăng sao Hỏa là Phobos và Deimos, họ có thể sử dụng mã đó. Một chức năng thú vị khác có chứa trong lớp hành tinh là chức năng được sử dụng để vẽ quỹ đạo của nó. Đầu tiên, tác giả tạo các điểm bằng hàm sin và sau đó kết nối chúng bằng GL\_LINE\_STRIP. Đây là code:



**Satellite.cs**

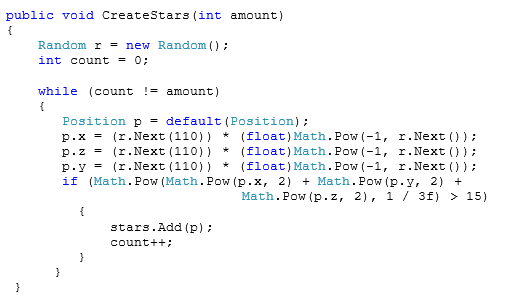
Lớp vệ tinh chứa mọi biến giống với lớp khai báo hành tinh. Điểm khác biệt duy nhất là tâm quay của nó không phải là mặt trời mà là hành tinh chủ của nó.

**SolarSystem.cs**

Đây là lớp chứa danh sách các hành tinh, các ngôi sao và vệ tinh. Lớp này để khai báo và vẽ các đối tượng đồ hoạ đó.

**Star.cs**

Đây là lớp vẽ các ngôi sao. Các ngôi sao vẽ bằng GL\_POINTS đơn được vẽ ở các vị trí ngẫu nhiên (vùng ngoài). Đây là mã lệnh để vẽ chúng:

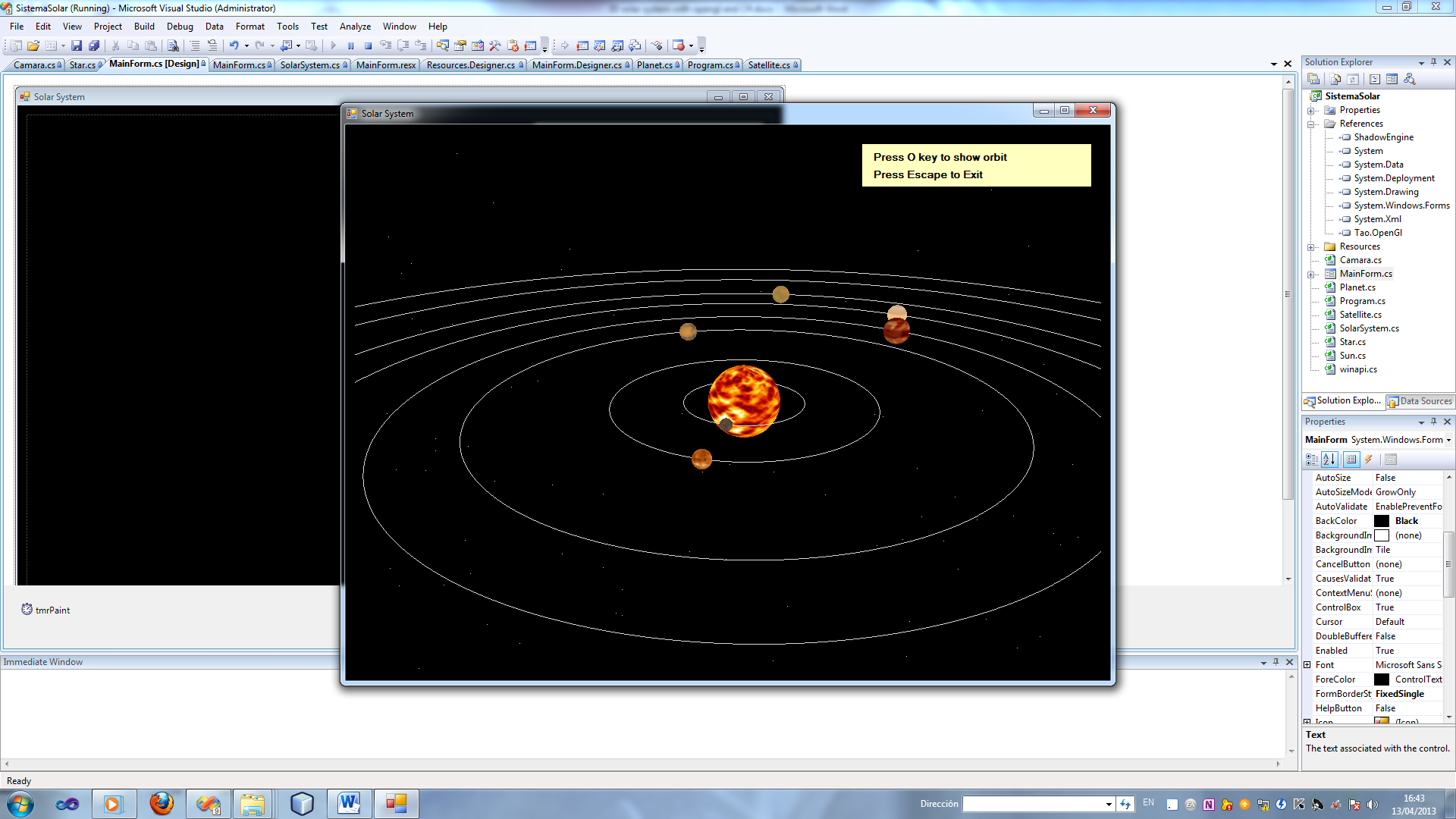


**Sun.cs**

Lớp mặt trời là lớp đơn giản nhất, nó giống như lớp hành tinh chỉ là nó không có quỹ đạo. Nó chỉ có một chuyển động quay quanh trục của nó.

Thông tin các lớp được dịch sang Tiếng Việt bởi Nguyễn Văn Hải Long. Blog cá nhân của tác giả có kèm nhiều chương trình 3D: <http://vasilydev.blogspot.com>.

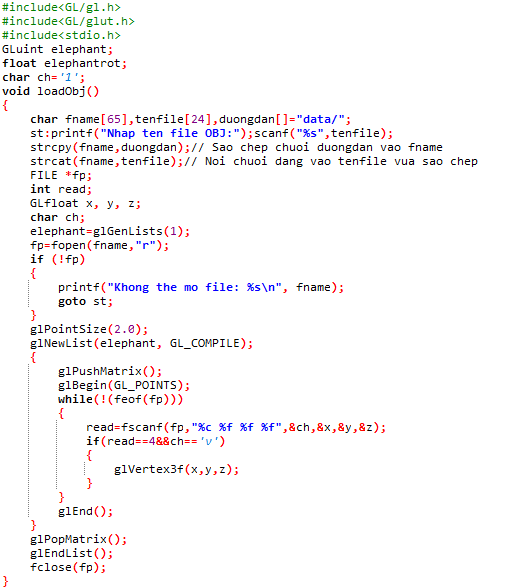
Kết quả chương trình( Sử dụng Visual Studio):

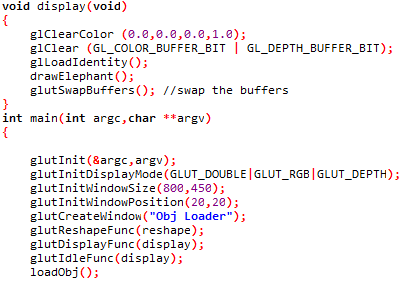


## **2.3 Đọc file OBJ sử dụng OpenGL**

Tệp OBJ là một định dạng tệp mô hình 3D. Một tiêu chuẩn được phát triển bởi Alias Wavefront cho phần mềm hoạt hình và mô hình 3D "Advanced Visualizer", phù hợp để tương tác lẫn nhau giữa các mô hình và phần mềm 3D, nó cũng có thể được đọc và lập trình thông qua Maya.

Triển khai chương trình:



Kết quả chương trình:



# 3. Đường cong Bezier và mặt cong Bezier

## **3.1. Đường cong Bezier**

Đường cong Bezier là một đường cong tham số thường được sử dụng trong đồ họa máy tính và một số lĩnh vực khác. Dạng tổng quát hóa của đường cong Bezier trong không gian nhiều chiều được gọi là mặt phẳng Bezier, trong đó tam giác Bezier là một trường hợp đặc biệt.

Đường cong Bezier được công bố lần đầu vào năm 1962 bởi một kỹ sư người Pháp Pierre Bezier, người sử dụng nó để thiết kế thân ôtô. Nhưng việc nghiên cứu những đường cong này thực tế đã bắt đầu từ năm 1959 bởi nhà toán học Paul de Casteljau, ông sử dụng giải thuật De Casteljau để đánh giá các đường cong đó.

Về mặt ứng dụng, đường cong Bezier thường được sử dụng trong đồ họa 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 dẫn" (path), một khái niệm được sử dụng trong các chương trình xử lý ảnh, được tạo ra bằng cách liên kết các đường cong Bezier với nhau. Đường cong Bezier còn thường được sử dụng như là một công cụ để điều khiển sự chuyển động (animation).

Mô tả thuật toán vẽ đường cong Bezier lập phương hay bậc ba (cubic). Với 4 điểm P0, P1, P2 và P3 trên mặt phẳng hoặc trong không gian nhiều chiều có thể định nghĩa một đường cong Bezier bậc 3. Đường cong này bắt đầu từ điểm P0, đi theo hướng của điểm P1 và P2 trước khi kết thúc tại p3. Đường cong được hình thành thường không trực tiếp đi qua điểm P1 và P2 và 2 điểm này chỉ mang tính định hướng cho đường cong.

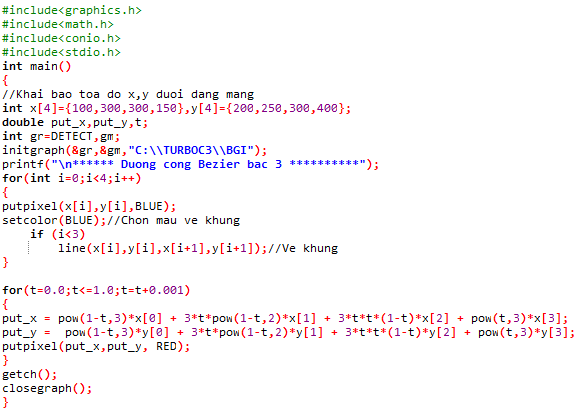
Phương trình đường cong Bezier bậc 3 có thể được định nghĩa bằng cách kết hợp 2 đường cong Bezier bậc 2 với nhau, với phương trình cho đường cong Bezier bậc 2 là BPi,Pj,Pk, trong đó Pi, Pj, Pk là các điểm của đường cong đó:



Dạng tường minh cho phương trình này là:



Triển khai chương trình vẽ Bezier bậc 3:



Kết quả chương trình:

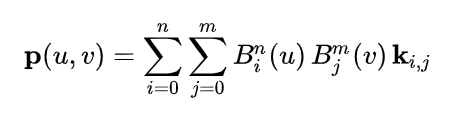


## **3.2. Mặt cong Bezier**

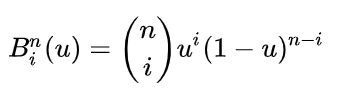
Một bề mặt Bezier cho trước có bậc ( *n* ,  *m* ) được xác định bởi tập hợp ( *n*  + 1) ( *m*  + 1) điểm điều khiển k *i*,*j* trong đó *i* = 0, ..., *n* và *j* = 0, .. ., *m* . Nó ánh xạ theo đơn vị vuông (là một hình vuông có độ dài các cạnh là 1) thành một bề mặt liên tục được đặt trong không gian chứa k *i*,*j* .

Ví dụ, nếu k *i*,*j*  là tất cả các điểm trong không gian bốn chiều, thì bề mặt đó sẽ nằm trong 1 không gian bốn chiều.

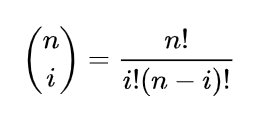
Một bề mặt Bezier hai chiều có thể được định nghĩa là một bề mặt tham số trong đó vị trí của một điểm p là một hàm của các tọa độ tham số *u* ,  *v* được cho bởi:



được xét trên bình phương đơn vị, trong đó

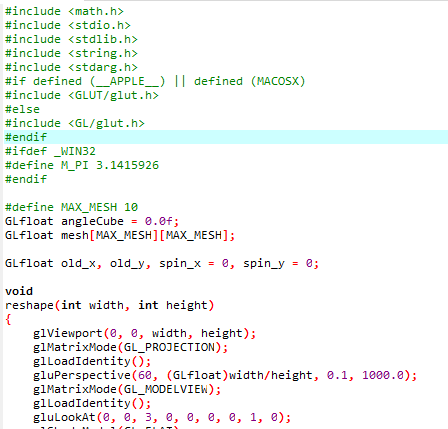


là một đa thức Bernstein cơ sở , và

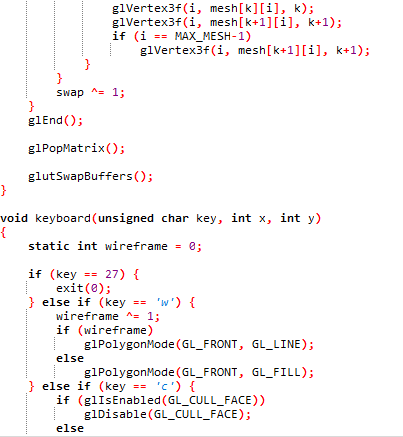


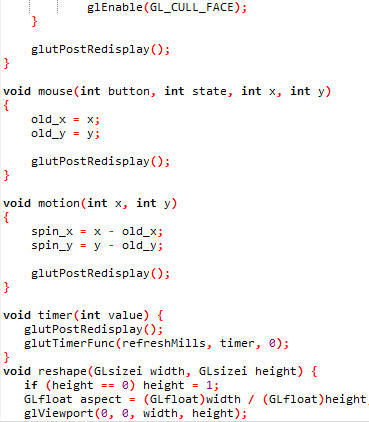
là một hệ số nhị thức.

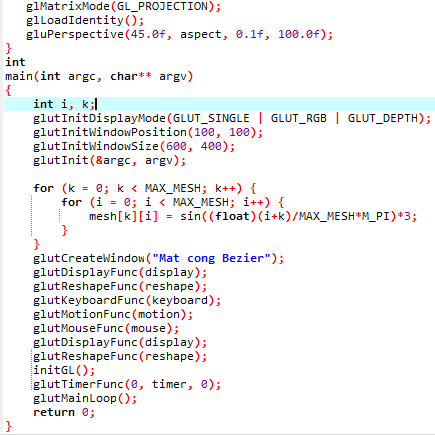
Triển khai chương trình:



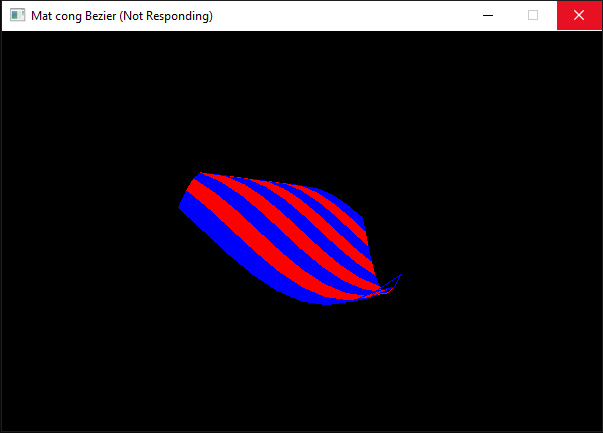








Kết quả chương trình:



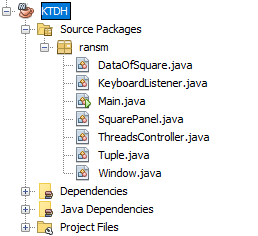
# 4. Demo đồ hoạ với Java

Demo những chương trình đồ hoạ đơn giản trên kênh Youtube: [https:/www.youtube.com/channel/UC4SVo0Ue36XCfOyb5Lh1viQ](https://www.youtube.com/channel/UC4SVo0Ue36XCfOyb5Lh1viQ)

Chạy code trên môi trường Apache NetBeans IDE:

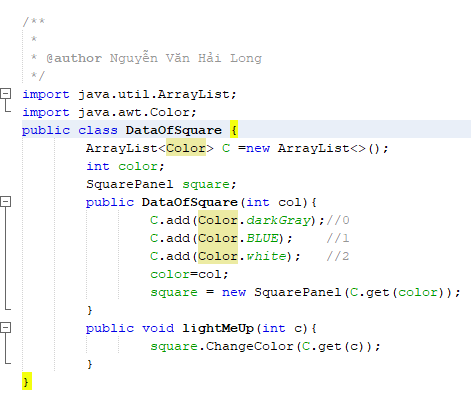
Tên chương trình: Rắn săn mồi

Cấu trúc:

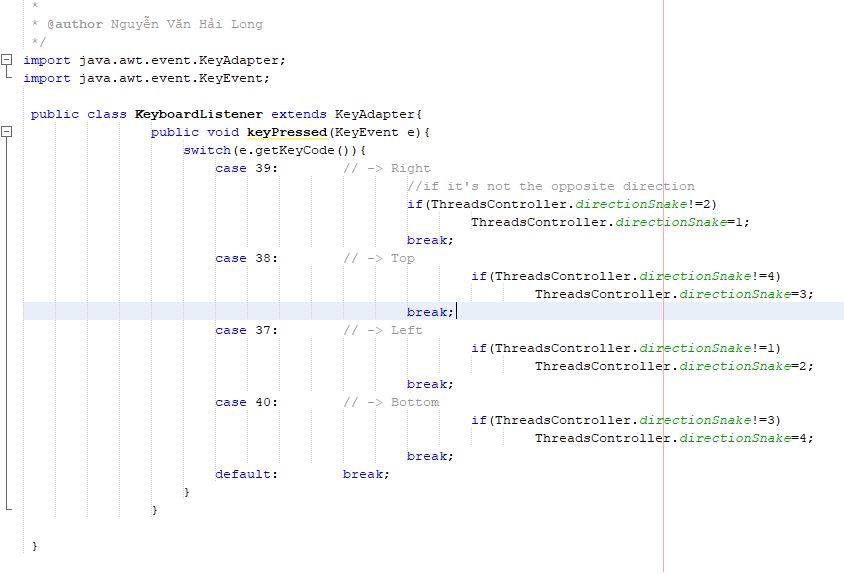


**DataOfSquare.java**

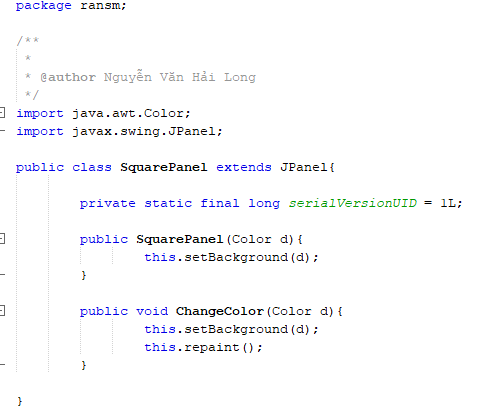
Khai báo màu sắc giao diện



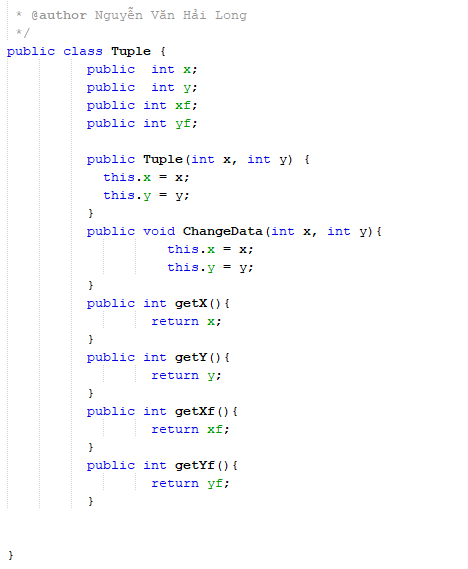
**KeyboardListener.java**

Gán phím sự kiện:

**SquarePanel.java**



**Tuple.java**



Kết quả chương trình:

