ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Môn học: Đồ họa máy tính

Báo cáo đồ án 1

Vẽ đối tượng 2D

Sinh viên: Phan Văn Hoàng
Giảng viên: Trần Thái Sơn
Giảng viên hướng dẫn: Võ Hoài Việt

Mục lục

1 Mở đầu

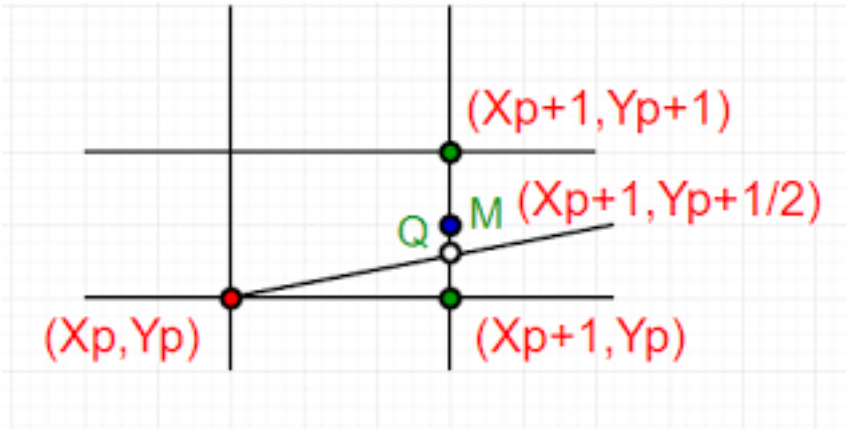
1.1 Giới thiệu

OpenGL (Open Graphics Library) là một API đa nền tảng, đa ngôn ngữ cho kết xuất đồ họa vector 2D và 3D. API thường được sử dụng để tương tác với bộ xử lý đồ họa (GPU), nhằm đạt được tốc độ kết xuất phần cứng. Cho đến nay, OpenGL (Open Graphics Library) đã trở thành một công cụ quan trọng trong phát triển ứng dụng đồ họa 2D và 3D. Đồ án này tập trung vào việc vẽ các hình 2D bằng GLUT (OpenGL Utility Toolkit) để tạo các cửa sổ đồ họa, xử lý sự kiện và tạo giao diện đồ họa cơ bản.

1.2 Mục tiêu

- Vẽ các object 2D bao gồm: đường thẳng, đường tròn, elip, parabol, hyperbol sử dụng các thuật toán như DDA, Bresenham, MidPoint thông qua thư viện đồ họa OpenGL và GLUT.
- So sánh và đánh giá với các hàm vẽ do thư viện OpenGL cung cấp về thời gian và độ chính xác.

2 Lý thuyết nền tảng của thuật toán midpoint



Hình 1: Minh họa midpoint cho đường thẳng (Nguồn: Geeksforgeeks)

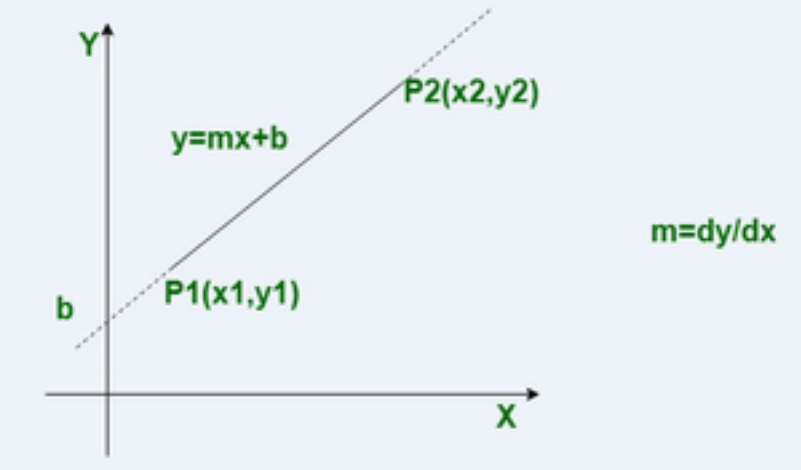
Thuật toán Midpoint là một phương pháp hiệu quả và phổ biến để vẽ nhiều hình trong trong không gian 2D và là một phần quan trọng trong các ứng dụng đồ họa và máy tính. Ý tưởng cơ bản của thuật toán Midpoint là sử dụng một điểm trung tâm

(Midpoint) để quyết định xem điểm nào nên được chọn để vẽ tiếp theo. Thuật toán này làm việc bằng cách tính toán một giá trị gọi là sai số (error) dựa trên vị trí hiện tại của điểm midpoint so với 2 điểm pixel có thể chọn tiếp theo (trên dưới hoặc trái phải). Dựa vào tham số quyết định (decision parameter) ta có thể chọn pixel tiếp theo cần vẽ. Đối với mỗi loại object sẽ có các cách xử lý khác nhau, nhưng về cơ bản thì dựa trên điểm midpoint để quyết định điểm tiếp theo, nếu điểm midpoint nằm phía dưới (trên) đường thẳng (hoặc đường cong, ...) tức là đường thẳng (hoặc đường cong, ...) gần với pixel phía trên (dưới) hơn nên ta chọn pixel phía trên (dưới) là pixel tiếp theo (xem hình minh họa 1).

Để hiểu chi tiết hơn về midpoint algorithm, phần 3 sẽ xây dựng và giải thích chi tiết các thuật toán cho các hình như đường thẳng, hình tròn, hình elip,...

3 Xây dựng và giải thích các thuật toán

3.1 Basic Incremental Algorithm (Digital Differential Analyzer)



Hình 2: Hình minh họa một đoạn thẳng (Nguồn: Geeksforgeeks)

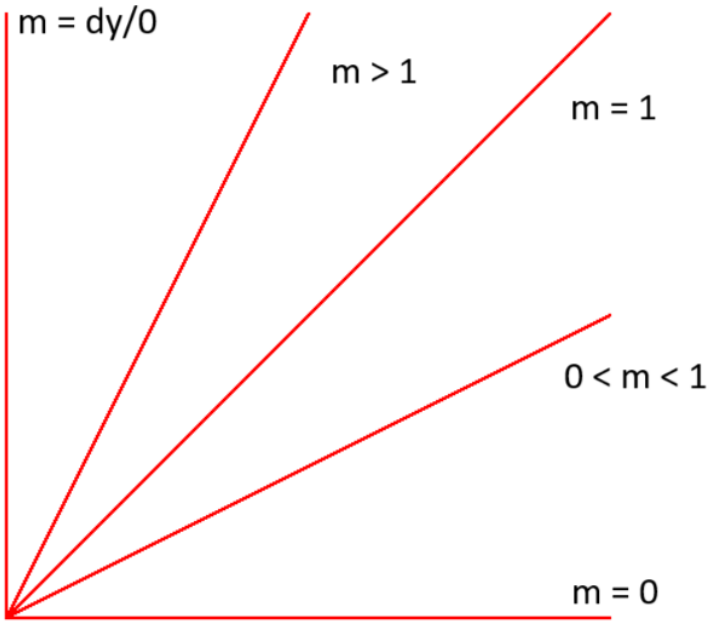
3.1.1 Công thức một đường thẳng

Để biểu diễn một đường thẳng, có nhiều cách khác nhau. Thuật toán DDA chọn cách biểu diễn đường thẳng sau để vẽ một đoạn thẳng:

$$y = mx + b$$

Trong đó:

- $m = \frac{dy}{dx} = \frac{y_2 - y_1}{x_2 - x_1} = tan(\theta)$ với θ là góc hợp bởi đường thẳng và Ox.
- $b = y_1 - mx_1$



Hình 3: Hình minh họa các trường hợp của đường thẳng với $m >= 0$ và $x_1 < x_2, y_1 < y_2$. Đối với các trường hợp m < 0 thì ta sẽ suy luận tương tự.

3.1.2 Trường hợp $0 \leq m \leq 1$

Với $0 \leq m \leq 1$ thì dy <= dx do đó ở mỗi bước x tăng một lượng Δx.

Ta có: $x_{k+1} = x_k + \Delta x \Rightarrow y_{k+1} = y_k + m\Delta x$.

Cho $\Delta x = 1$ thì ở mỗi bước, ta tính điểm tiếp theo như sau: $\begin{cases} y_{k+1} = y_k + m \\ x_{k+1} = x_k + 1 \end{cases}$

Data: (x_1, y_1), (x_2, y_2) - Hai điểm cuối của đoạn thẳng
Result: Các điểm trên đoạn thẳng
$Dx \leftarrow x_2 - x_1$, $Dy \leftarrow y_2 - y_1$;
$m \leftarrow \frac{Dy}{Dx}$;
$x \leftarrow x_1$, $y \leftarrow y_1$;
setPixel(x, round(y));
while $x < x_2$ **do**
    $x \leftarrow x + 1$;
    $y \leftarrow y + m$;
    setPixel(x, round(y));
end

Algorithm 1: Thuật toán DDA để vẽ đường thẳng với $0 \leq m \leq 1$

3.1.3 Trường hợp m > 1

Với m > 1 thì dy > dx do đó ở mỗi bước x tăng một lượng Δy.

Ta có: $y_{k+1} = y_k + \Delta y \Rightarrow x_{k+1} = x_k + \frac{1}{m}\Delta y$.

Cho $\Delta y = 1$ thì ở mỗi bước, ta tính điểm tiếp theo như sau: $\begin{cases} \mathrm{y}_{k+1} = y_k + 1 \\ \mathrm{x}_{k+1} = x_k + \frac{1}{m} \end{cases}$

Data: (x_1, y_1), (x_2, y_2) - Hai điểm cuối của đoạn thẳng
Result: Các điểm trên đoạn thẳng
$Dx \leftarrow x_2 - x_1$, $Dy \leftarrow y_2 - y_1$;
$m \leftarrow \frac{Dy}{Dx}$;
$x \leftarrow x_1$, $y \leftarrow y_1$;
setPixel(x, round(y));
while $y < y_2$ **do**
    $x \leftarrow x + 1/m$;
    $y \leftarrow y + 1$;
    setPixel($round(x)$, y));
end

Algorithm 2: Thuật toán DDA để vẽ đường thẳng với m > 1

3.1.4 Tổng quát

Với $0 \leq m \leq 1$ $(dy \leq dx)$ ta có: $\begin{cases} y_{k+1} = y_k + m = y_k + \frac{dy}{dx} \\ x_{k+1} = x_k + 1 = x_k + \frac{dx}{dx} \end{cases}$

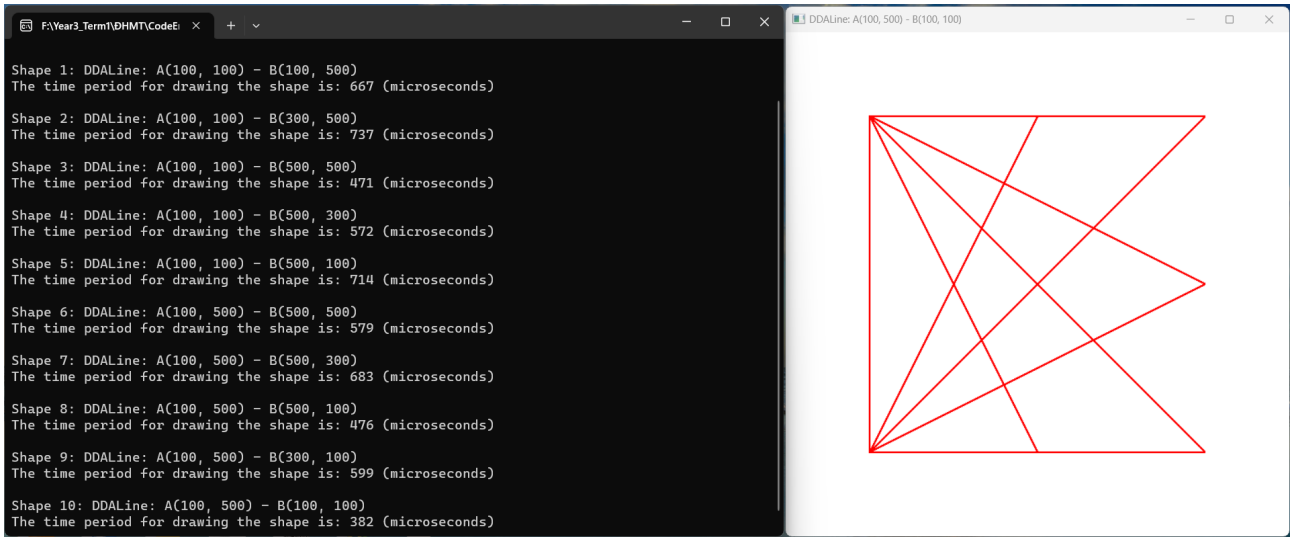
Với $m > 1$ $(dy > dx)$ ta có: $\begin{cases} y_{k+1} = y_k + 1 = y_k + \frac{dy}{dy} \\ x_{k+1} = x_k + \frac{1}{m} = x_k + \frac{dx}{dy} \end{cases}$

Giả sử ta đặt $steps = max(|dx|, |dy|)$ (dx hoặc dy khác 0) thì $\forall m$ ta có: $\begin{cases} y_{k+1} = y_k + \frac{dy}{steps} \\ x_{k+1} = x_k + \frac{dx}{steps} \end{cases}$

Data: (x_1, y_1), (x_2, y_2) - Hai điểm cuối của đoạn thẳng
Result: Các điểm trên đoạn thẳng
$Dx \leftarrow x_2 - x_1$, $Dy \leftarrow y_2 - y_1$;
$steps \leftarrow max(abs(Dx), abs(Dy))$;
$x \leftarrow x_1$, $y \leftarrow y_1$;
$x\_inc \leftarrow Dx/steps$;
$y\_inc \leftarrow Dy/steps$;
for $i \leftarrow 0$ **to** $steps - 1$ **do**
    `setPixel`(`round`(x), `round`(y));
    $x \leftarrow x + x\_inc$;
    $y \leftarrow y + y\_inc$;
end

Algorithm 3: Thuật toán DDA để vẽ đường thẳng

3.1.5 Kết quả minh họa thuật toán



Hình 4: Kết quả minh họa của thuật toán (microseconds)

Kết quả minh họa của thuật toán được thể hiện trên hình 4, ta thấy thuật toán chạy với một thời gian không chênh lệch nhiều giữa các trường hợp, đều dưới 1 milisecond cho mỗi hình.

3.2 Bresenham Algorithm for Line

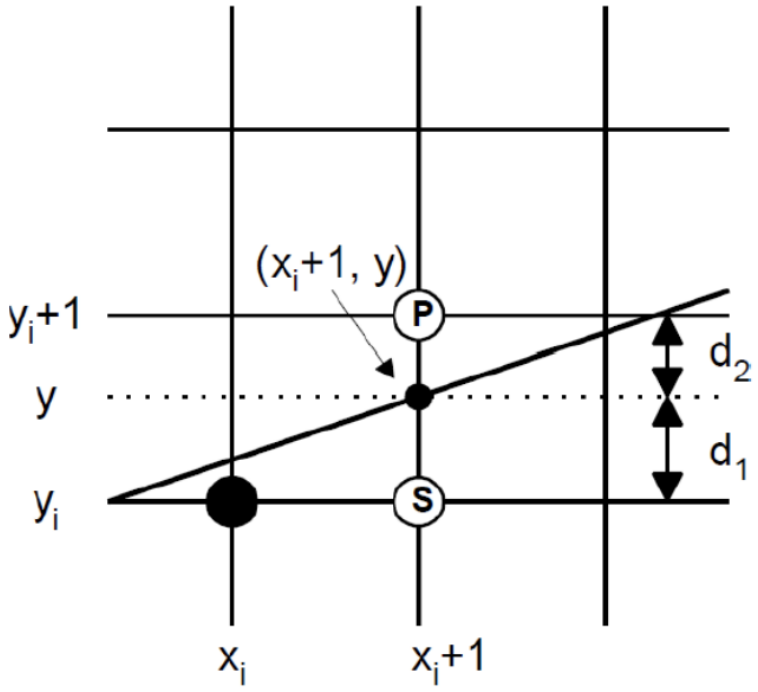
Thuật toán DDA là một thuật toán phổ biến, dễ cài đặt, tuy nhiên nó có một số điểm yếu như sau:

- Sai số lớn: DDA sử dụng phép chia để tính toán các giá trị dự đoán cho các điểm trên đường thẳng. Do đó, nó thường dẫn đến sai số lớn, đặc biệt khi vẽ đường thẳng với độ dốc lớn hoặc nếu cần vẽ đường thẳng ngắn.
- Sai số nguyên: DDA thường trả về các giá trị số thực, cần phải làm tròn thành số nguyên. Điều này tạo ra sự cố về sai số nguyên, có thể gây ra sai lệch vị trí của các điểm trên đường thẳng.
- Tính chậm: Vì DDA sử dụng phép chia và lượng tính toán số thực, nên nó thường chậm hơn so với thuật toán khác sử dụng số nguyên.

Chính vì những điểm hạn chế đó, Bresenham Algorithm đã được ra đời và phát triển với sai số nhỏ hơn, tính toán trên số nguyên, do đó cũng cải thiện được tốc độ. Vì vậy

Bresenham Algorithm thường là lựa chọn tốt hơn trong nhiều trường hợp và được ưu tiên sử dụng trong đồ họa máy tính và các ứng dụng liên quan đến xử lý hình ảnh.

3.2.1 Ý tưởng



Hình 5: Hình vẽ minh họa ý tưởng Bresenham (Nguồn: bài giảng của giảng viên)

Với $0 < \mathrm{m} < 1$, điểm tiếp theo của điểm (x_i, y_i) là $P(x_i+1, y_i)$ hoặc $Q(x_i+1, y_i+1)$. Dựa vào khoảng cách của điểm tiếp theo (x_i+1, y) so với P hoặc Q, nếu gần điểm nào hơn thì ta sẽ lấy điểm đó.

3.2.2 Xây dựng thuật toán (với $0 < \mathrm{m} < 1$)

Đặt: $\begin{cases} \mathrm{d}_1 = y - y_i \\ \mathrm{d}_2 = (y_i + 1) - y \end{cases} \Rightarrow \mathrm{y}_{i+1} = \begin{cases} \mathrm{y}_i, \text{ nếu } d_1 < d_2, \\ \mathrm{y}_i + 1, \text{ nếu } d_1 \geq d_2 \end{cases}$

Thay vì so sánh d_1 với d_2, ta so sánh $d_1 - d_2$ với 0.

Với $y = \frac{Dy}{Dx}(x_i + 1) + b$ thì

$$d_1 - d_2 = 2y - 2y_i - 1 \quad (1)$$

$$= 2\frac{Dy}{Dx}x_i - 2y_i + 2\frac{Dy}{Dx} + 2b - 1 \quad (2)$$

Đặt $p_i = Dx(d_1 - d_2) = 2Dy.x_i - 2Dx.y_i + 2Dy + Dx(2b - 1)$
Tính tham số quyết định cho pixel tiếp theo dựa trên pixel trước:

$$p_{i+1} = p_i + 2Dy - 2Dx(y_{i+1} - y_i)$$

Ta có: $y_{i+1} = \begin{cases} \mathrm{y}_i, \text{ nếu } d_1 < d_2, \\ \mathrm{y}_i + 1, \text{ nếu } d_1 \geq d_2 \end{cases}$

Suy ra, nếu $\begin{cases} \mathrm{p}_i < 0 \Rightarrow y_{i+1} = y_i \Rightarrow p_{i+1} = p_i + 2Dy, \\ \mathrm{p}_i \geq 0 \Rightarrow y_{i+1} = y_i + 1 \Rightarrow p_{i+1} = p_i + 2Dy - 2Dx \end{cases}$

Trong đó, giá trị khởi đầu của p_i là:

$$\begin{aligned} p_0 &= 2Dy.x_0 - 2Dx.y_0 + 2Dy + Dx(2b - 1) && (3) \\ &= 2Dy - Dx + 2(Dy.x_0 - Dx.y_0 + Dx.b) && (4) \\ &= 2Dy - Dx \text{ (vì } mx_0 + b = y_0) && (5) \end{aligned}$$

3.2.3 Tổng quát

Data: Hai điểm cuối của đoạn thẳng: (x_1, y_1) và (x_2, y_2)
Result: Các điểm trên đoạn thẳng

$x \leftarrow x_1$, $y \leftarrow_Y 1$;
$dx \leftarrow |x_2 - x_1|$, $dy \leftarrow |y_2 - y_1|$;
$s1 \leftarrow \text{sign}(x_2 - x_1)$, $s2 \leftarrow \text{sign}(y_2 - y_1)$;
interchange $\leftarrow$ false;
if $dy > dx$ **then**
    swap(dx, dy);
    interchange $\leftarrow$ true;
end
$p \leftarrow 2dy - dx$;
$c1 \leftarrow 2dy$;
$c2 \leftarrow 2dy - 2dx$;
`setPixel`(x, y);
for $i \leftarrow 0$ **to** $dx - 1$ **do**
    if $p < 0$ **then**
        if *interchange* **then**
            $y \leftarrow y + s2$;
        end
        else
            $x \leftarrow x + s1$;
        end
        $p \leftarrow p + c1$;
    end
    else
        $y \leftarrow y + s2$;
        $x \leftarrow x + s1$;
        $p \leftarrow p + c2$;
    end
    `setPixel`(x, y);
end

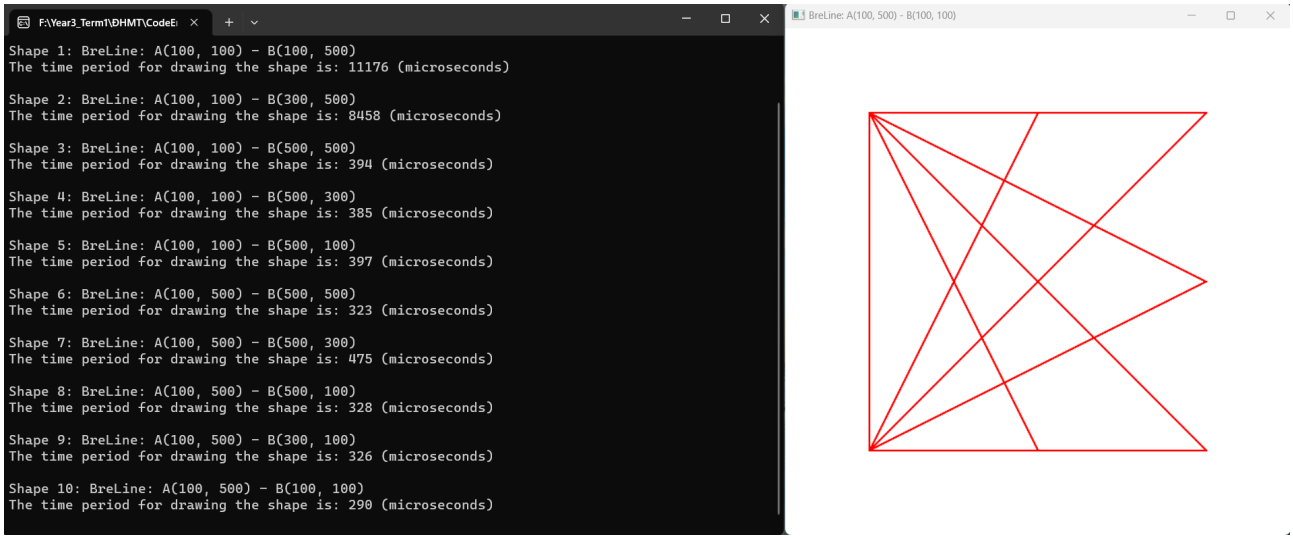
Algorithm 4: Thuật toán Bresenham

Đối với độ dốc m $>$ 1, ta chỉ cần hoán đổi vai trò của x và y trong trường hợp $0 <$

m $<$ 1. Thuật toán tổng quát được trình bày ở Algorithm 4. Trong thuật toán, biến s1 đại diện cho hướng của x, biến s2 đại diện cho hướng của y. Hàm sign(x) trả về 1 khi x $>$ 0, trả về -1 khi x $<$ 0 và trả về 0 khi x = 0. Chúng ta đã xây dựng thuật toán cho trường hợp 0 $<$ m $<$ 1 ở mục 3.2.2, theo đó x sẽ tăng 1 đơn vị qua từng vòng lặp và việc của chúng ta là tìm y. Do đó, biến **interchange** dùng để xác định xem có hoán đổi vai trò của x và y hay không. Nếu interchange = false thì sẽ không hoán đổi, ngược lại sẽ hoán đổi vai trò của x và y.

Xét trường hợp $p_i < 0$. Với 0 $<$ m $<$ 1, $y_{i+1} = y_i$ và interchange = false, do đó chỉ thay đổi x với lượng s1. Nếu $x_2 > x_1$ thì $x = x + 1$, còn nếu $x_2 < x_1$ thì $x = x - 1$, ngược lại $x = x + 0$. Với m $>$ 1, khi đó y sẽ tăng lượng s1 và x sẽ giữ nguyên. Đối với trường hợp $p_i \geq 0$. Với 0 $<$ m $<$ 1, thì $y_{i+1} = y_i + 1$, do đó y tăng lượng s2 và x tăng lượng s1. Tương tự cho trường hợp m $>$ 1.

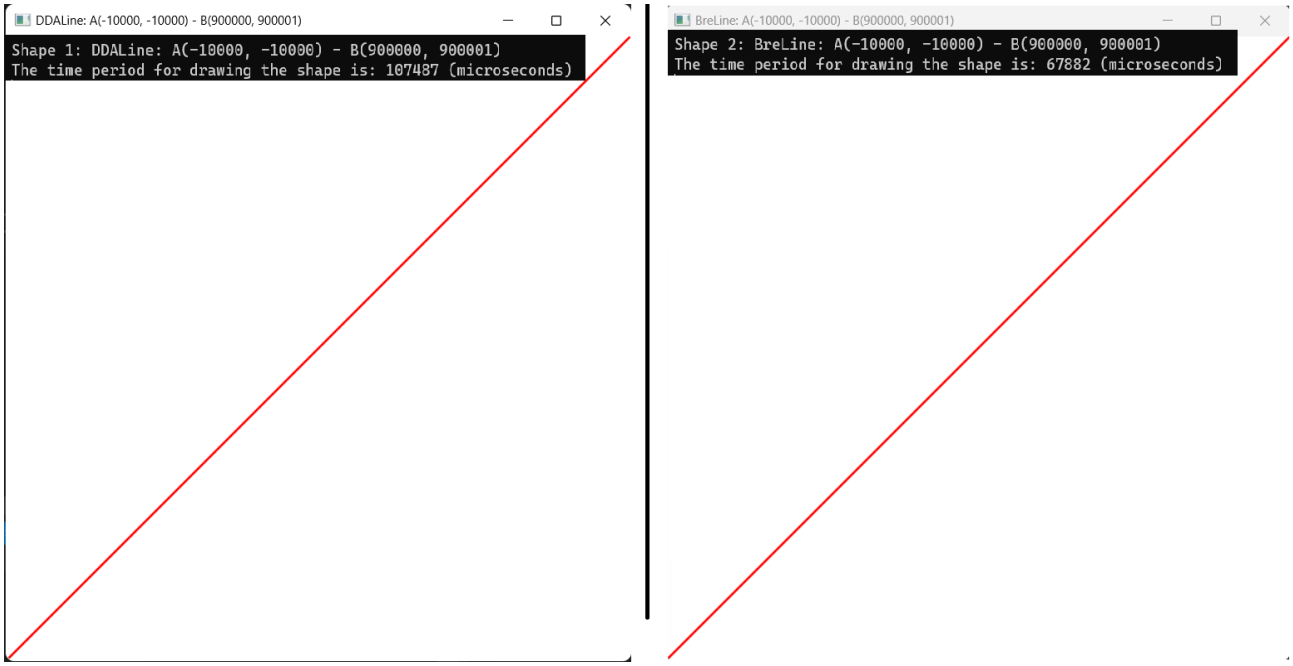
3.2.4 Kết quả minh họa thuật toán



Hình 6: Kết quả minh họa của thuật toán (microseconds)

Kết quả minh họa của thuật toán được thể hiện trên hình 6, ta thấy thuật toán chạy đều dưới 0.5 millisecond đối với các hình có $dx \geq dy$. Còn đối với 2 hình 1 và 2, có thời gian lâu hơn các hình còn lại vì $dy > dx$ do đó tốn một chi phí thời gian nhỏ cho việc swap. Nhìn chung tốc độ của thuật toán vẫn rất nhanh.

3.2.5 So sánh DDA và Bresenham

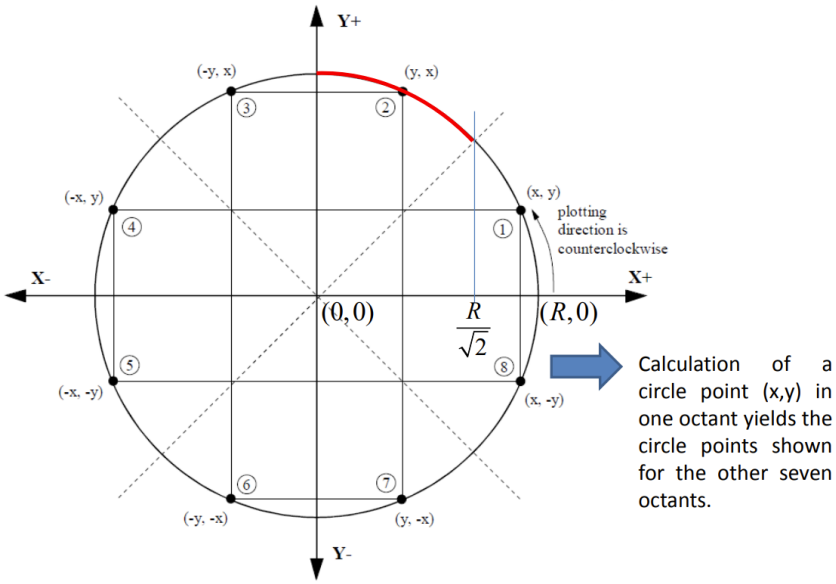


Hình 7: Kết quả của DDA và Bresenham với hình kích thước lớn

Dựa vào kết quả minh họa ở hình 4 và 6, mặc dù tốc độ của thuật toán Bresenham có chút nhỉnh hơn nhưng không có sự khác biệt lớn. Tuy nhiên, khi tăng khoảng cách của 2 điểm cuối của đoạn thẳng lên, đã có sự khác biệt của 2 thuật toán như đã được thể hiện trên hình 7, thuật toán Bresenham có tốc độ vượt trội rõ rệt, gần như gấp đôi so với thuật toán DDA.

3.3 Midpoint circle algorithm

3.3.1 Công thức đường tròn



Hình 8: Hình minh họa cho một đường tròn, được chia thành 8 octants (Nguồn: bài giảng của giảng viên)

Có nhiều cách để biểu diễn đường tròn, thông thường đường tròn sẽ được biểu diễn như sau:

$$(x - x_0)^2 + (y - y_0)^2 = R^2$$
$$\Rightarrow x^2 + y^2 = R^2 \text{ với tâm là O(0, 0) và bán kính R}$$

Như được hiển thị trên hình 8, để vẽ được một đường tròn, ta chỉ cần vẽ được một octant sau đó ánh xạ qua các octant khác. Và để đơn giản, đường tròn có tâm là gốc tọa độ O(0, 0), đối với các đường tròn có tâm khác O thì ta sẽ dịch đường tròn tọa độ từ tâm O đến tâm tương ứng để được đường tròn chính xác.

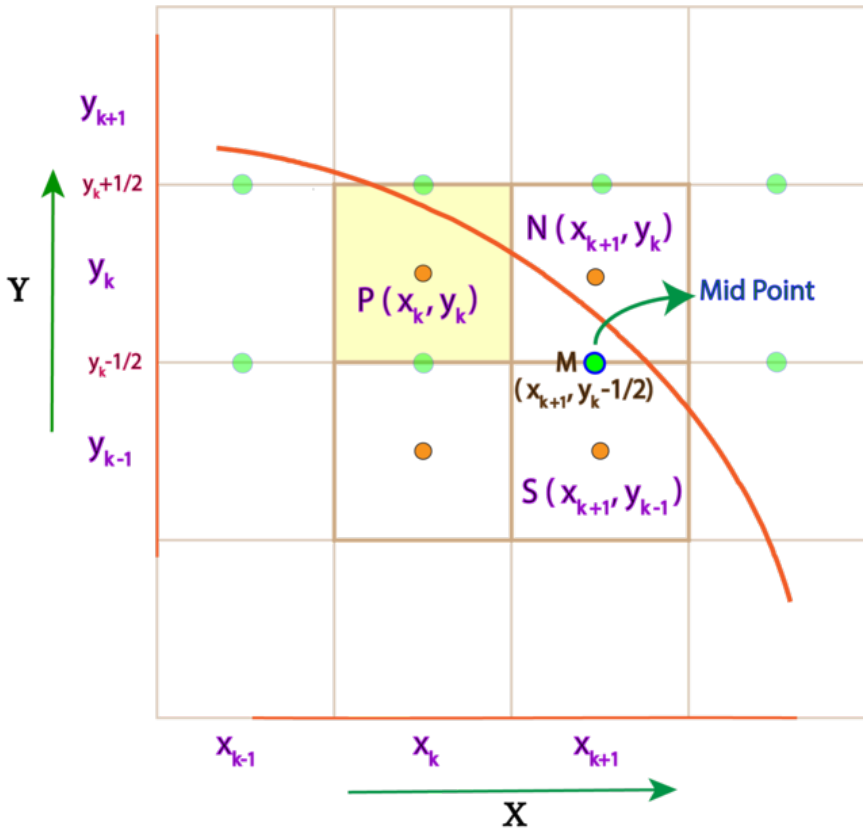
3.3.2 Xây dựng thuật toán tổng quát

Cho đường tròn tâm O(0, 0) và bán kính R và một điểm (x, y)

$$F(x, y) = x^2 + y^2 - R^2$$

$$\text{Nếu F(x, y)} \begin{cases} < 0 \text{ thì (x, y) nằm bên trong đường tròn} \\ = 0 \text{ thì (x, y) nằm trên đường tròn} \\ > 0 \text{ thì (x, y) nằm bên ngoài đường tròn} \end{cases}$$

Xét tại cung tròn số 2 trên hình 8, giả sử ta đã vẽ điểm (x_k, y_k) và cần vẽ điểm tiếp theo. Điểm tiếp theo có thể là $P(x_k + 1, y_k)$ hoặc $Q(x_k + 1, y_k - 1)$.



Hình 9: Midpoint Circle

Ý tưởng thuật toán: điểm midpoint $M(x_i + 1, y_i - \frac{1}{2})$ là điểm ở giữa hai điểm có thể là pixel tiếp theo. Nếu M nằm trong đường tròn thì y_i gần với đường tròn hơn, ngược lại nếu M nằm ngoài hoặc trên đường tròn thì $y_i - 1$ gần với đường tròn hơn.

Đặt $p_i = F(x_i + 1, y_i - \frac{1}{2}) = (x_i + 1)^2 + (y_i - \frac{1}{2})^2 - R^2$.

Nếu $\begin{cases} p_i < 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i + 1, y_i) \\ p_i \geq 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i + 1, y_i - 1) \end{cases}$

Tính tham số quyết định cho pixel tiếp theo dựa trên pixel trước:

$$p_{i+1} = p_i + (2x_i + 3) + (y_{i+1} - y_i)(y_{i+1} + y_i - 1) \tag{6}$$

Với $\begin{cases} p_i < 0 \Rightarrow y_{i+1} = y_i \Rightarrow p_{i+1} = p_i + (2x_i + 3), \\ p_i \geq 0 \Rightarrow y_{i+1} = y_i - 1 \Rightarrow p_{i+1} = p_i + (2x_i + 3) - (2y_i - 2) \end{cases}$

Trong đó, giá trị khởi đầu của p_i tại điểm (0, R) là:

$$p_0 = (0+1)^2 + (R - \frac{1}{2})^2 - R^2 \tag{7}$$

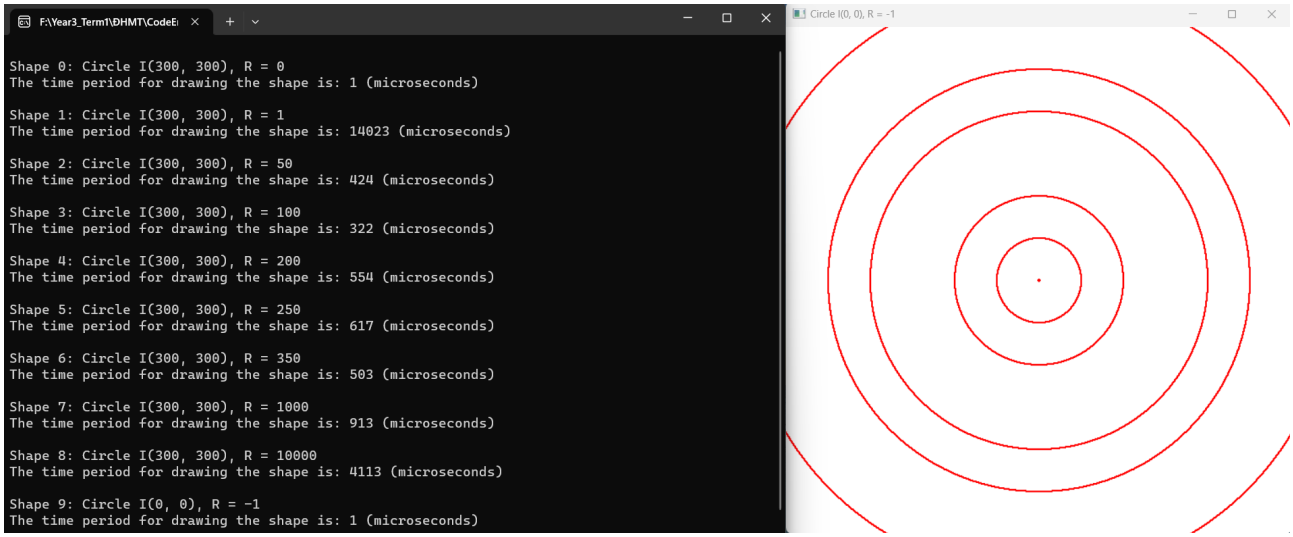
$$= \frac{5}{4} - R \tag{8}$$

Data: $(xt, yt), R$ - Tọa độ tâm và bán kính đường tròn
Result: Các điểm trên đường tròn

$x \leftarrow 0$, $y \leftarrow R$;
setPixel(xt, $yt + R$);
setPixel(xt, $yt - R$);
setPixel($xt + R$, yt);
setPixel($xt - R$, yt);
$p \leftarrow \frac{5}{4} - R$;
while $x <= y$ **do**
    if $p < 0$ **then**
        $p \leftarrow p + (2x_i + 3)$;
    end
    else
        $p \leftarrow p + (2x_i + 3) - (2y_i - 2)$;
        $y \leftarrow y - 1$;
    end
    $x \leftarrow x + 1$;
    setPixel($xt + x$, $yt + y$));
    setPixel($xt + x$, $yt - y$));
    setPixel($xt - x$, $yt + y$));
    setPixel($xt - x$, $yt - y$));
    setPixel($xt + y$, $yt + x$));
    setPixel($xt + y$, $yt - x$));
    setPixel($xt - y$, $yt + x$));
    setPixel($xt - y$, $yt - x$));
end

Algorithm 5: Thuật toán vẽ đường tròn bằng phương pháp midpoint

3.3.3 Kết quả minh họa thuật toán

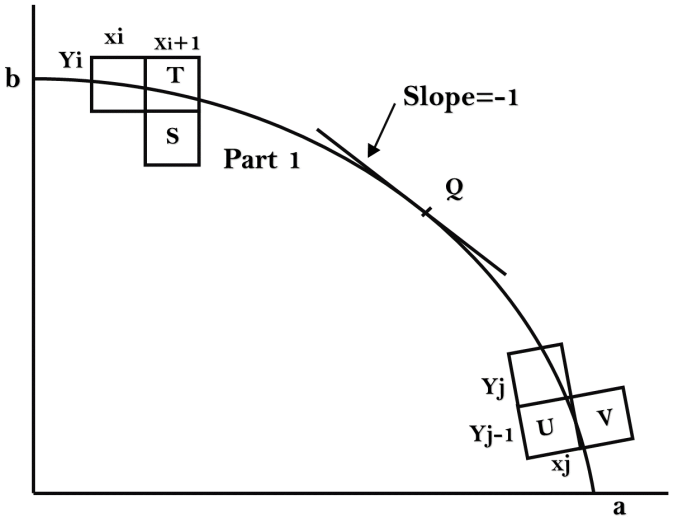


Hình 10: Kết quả minh họa của thuật toán

Nhận xét: Kết quả khá tốt, đối với những hình có kích thước vừa, tốc độ vẽ một hình xấp xỉ 0.5 millisecond. Kết quả hình vẽ không bị đứt đoạn hay không có cảm giác các pixel chênh lệch nhiều. Tuy nhiên, ở trường hợp R = 1, thuật toán vẽ có thời gian cao hơn so với R lớn hơn một chút.

3.4 Midpoint elipse algorithm

3.4.1 Công thức hình elipse



Hình 11: Hình minh họa cho quarter 1, được chia thành 2 regions (Nguồn: Internet)

Có nhiều cách để biểu diễn hình elip, tuy nhiên để thuận tiện cho việc tính toán và vẽ hình, công thức sau đây được sử dụng để xây dựng thuật toán midpoint elipse algorithm:

$$F(x, y) = b^2x^2 + a^2y^2 - a^2b^2 \tag{9}$$

Trong đó:

- Tâm là O(0, 0)
- A là $\frac{1}{2}$ độ dài trục lớn
- B là $\frac{1}{2}$ độ dài trục nhỏ

$$\text{Nếu F(x, y)} \begin{cases} < 0 \text{ thì (x, y) nằm bên trong đường elipse} \\ = 0 \text{ thì (x, y) nằm trên elipse} \\ > 0 \text{ thì (x, y) nằm bên ngoài đường elipse} \end{cases}$$

Như được hiển thị trên hình 11, để vẽ được một hình elipse, ta chỉ cần vẽ được Quarter 1 sau đó ánh xạ qua các Quarter còn lại. Trong Quarter 1 chia thành 2 regions, trong đó region 1 có $|slope| < 1$ và trong region 2 thì $|slope| > 1$.

3.4.2 Trường hợp $|slope| < 1$

Giả sử ta đã vẽ điểm (x_k, y_k) và cần vẽ điểm tiếp theo. Điểm tiếp theo có thể là $P(x_k + 1, y_k)$ hoặc $Q(x_k + 1, y_k - 1)$.

Đặt $p_i = F(x_i + 1, y_i - \frac{1}{2}) = b^2(x_i + 1)^2 + a^2(y_i - \frac{1}{2})^2 - a^2b^2$.

Nếu $\begin{cases} \text{p}_i < 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i + 1, y_i) \\ \text{p}_i \geq 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i + 1, y_i - 1) \end{cases}$

Tính tham số quyết định cho pixel tiếp theo dựa trên pixel trước:

$$p_{i+1} = p_i + b^2(2x_i + 3)^2 + a^2(y_{i+1} - y_i)(y_{i+1} + y_i - 1) \tag{10}$$

Với $\begin{cases} p_i < 0 \Rightarrow y_{i+1} = y_i & \Rightarrow p_{i+1} = p_i + b^2(2x_i + 3) \\ p_i \geq 0 \Rightarrow y_{i+1} = y_i - 1 & \Rightarrow p_{i+1} = p_i + b^2(2x_i + 3) - a^2(2y_i - 2) \end{cases}$

Hay $\begin{cases} p_i < 0 \Rightarrow y_{i+1} = y_i & \Rightarrow p_{i+1} = p_i + 2b^2x_{i+1} + b^2 \\ p_i \geq 0 \Rightarrow y_{i+1} = y_i - 1 & \Rightarrow p_{i+1} = p_i + 2b^2x_{i+1} + b^2 - 2a^2y_{i+1} \end{cases}$

Trong đó, giá trị khởi đầu của p_i tại điểm (0, b) là:

$$p_0 = b^2(0 + 1)^2 + a^2(b - \frac{1}{2})^2 - a^2b^2 \tag{11}$$

$$= b^2 - a^2b + 0.25a^2 \tag{12}$$

3.4.3 Trường hợp $|slope| > 1$

Giả sử ta đã vẽ điểm (x_k, y_k) và cần vẽ điểm tiếp theo. Điểm tiếp theo có thể là $P(x_k, y_k - 1)$ hoặc $Q(x_k + 1, y_k - 1)$.

Đặt $q_i = F(x_i + \frac{1}{2}, y_i - 1) = b^2(x_i + \frac{1}{2})^2 + a^2(y_i - 1)^2 - a^2b^2$.

Nếu $\begin{cases} \text{q}_i > 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i, y_i - 1) \\ \text{q}_i \leq 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i + 1, y_i - 1) \end{cases}$

Tính tham số quyết định cho pixel tiếp theo dựa trên pixel trước:

$$q_{i+1} = q_i + b^2(x_{i+1} - x_i)(x_{i+1} + x_i + 1) - a^2(2y_i - 3) \tag{13}$$

Với $\begin{cases} q_i > 0 \Rightarrow x_{i+1} = x_i & \Rightarrow q_{i+1} = q_i - a^2(2y_i + 3) \\ q_i \leq 0 \Rightarrow x_{i+1} = x_i + 1 & \Rightarrow q_{i+1} = q_i + b^2(2x_i + 2) - a^2(2y_i + 3) \end{cases}$

Hay $\begin{cases} q_i > 0 \Rightarrow x_{i+1} = x_i & \Rightarrow q_{i+1} = q_i - 2a^2 y_{i+1} + a^2 \\ q_i \leq 0 \Rightarrow x_{i+1} = x_i + 1 & \Rightarrow q_{i+1} = q_i + 2b^2 x_{i+1} - 2a^2 y_{i+1} + a^2 \end{cases}$

Trong đó, giá trị khởi đầu của q_i tại điểm (x_k, y_k) (điểm kết thúc region 1) là:

$$q_0 = b^2(x_k + \frac{1}{2})^2 + a^2(y_k - 1)^2 - a^2b^2 \tag{14}$$

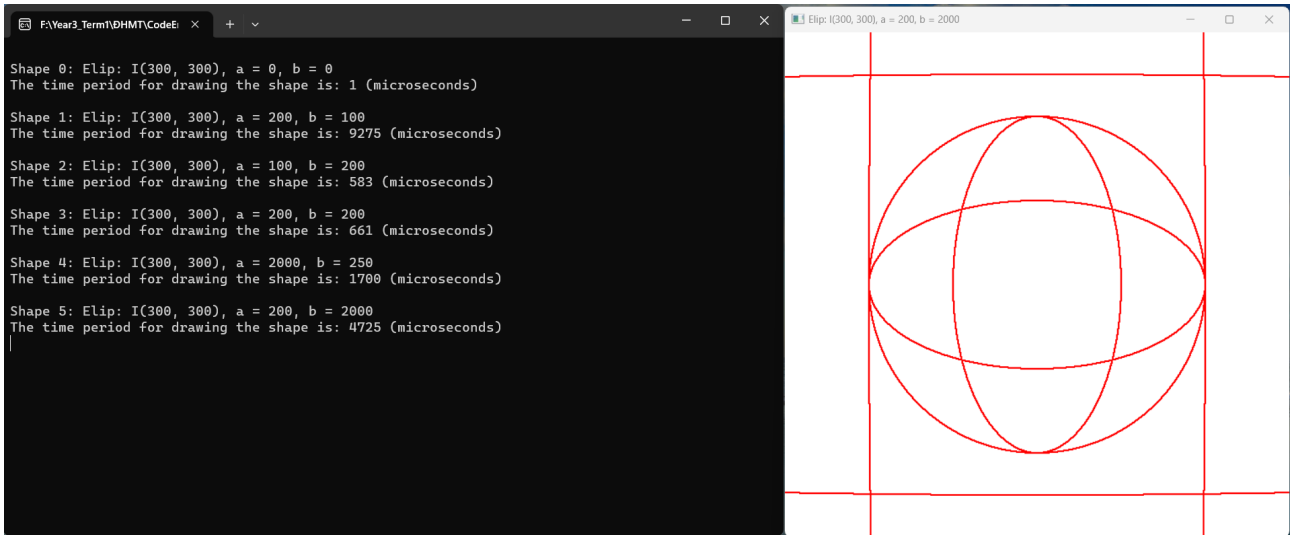
3.4.4 Thuật toán tổng quát

Data: Điểm trung tâm (xt, yt), bán trục lớn a, và bán trục nhỏ b
Result: Các điểm trên elip

if $a = b$ **then**
  plotCircle(xt, yt, a); **return**;
end
$p1 \leftarrow b^2 - a^2b + 0.25a^2$;
$x \leftarrow 0$, $y \leftarrow b$, $dx \leftarrow 2b^2x$, $dy \leftarrow 2a^2y$;
while $dx < dy$ **do**
  `setPixel`$(xt + x, yt + y)$; `setPixel`$(xt + x, yt - y)$;
  `setPixel`$(xt - x, yt + y)$; `setPixel`$(xt - x, yt - y)$;
  $x \leftarrow x + 1$, $dx \leftarrow dx + 2b^2$;
  if $p1 < 0$ **then**
    $p1 \leftarrow p1 + dx + b^2$;
  end
  else
    $y \leftarrow y - 1$; $dy \leftarrow dy - 2a^2$;
    $p1 \leftarrow p1 + dx - dy + b^2$;
  end
end
$p2 \leftarrow b^2(x + 0.5)^2 + a^2(y - 1)^2 - a^2b^2$;
while $y \geq 0$ **do**
  `setPixel`$(xt + x, yt + y)$; `setPixel`$(xt + x, yt - y)$;
  `setPixel`$(xt - x, yt + y)$; `setPixel`$(xt - x, yt - y)$;
  $y \leftarrow y - 1$; $dy \leftarrow dy - 2a^2$;
  if $p2 > 0$ **then**
    $p2 \leftarrow p2 - dy + a^2$;
  end
  else
    $x \leftarrow x + 1$; $dx \leftarrow dx + 2b^2$;
    $p2 \leftarrow p2 + dx - dy + a^2$;
  end
end

Algorithm 6: Midpoint elipse algorithm

3.4.5 Kết quả minh họa thuật toán

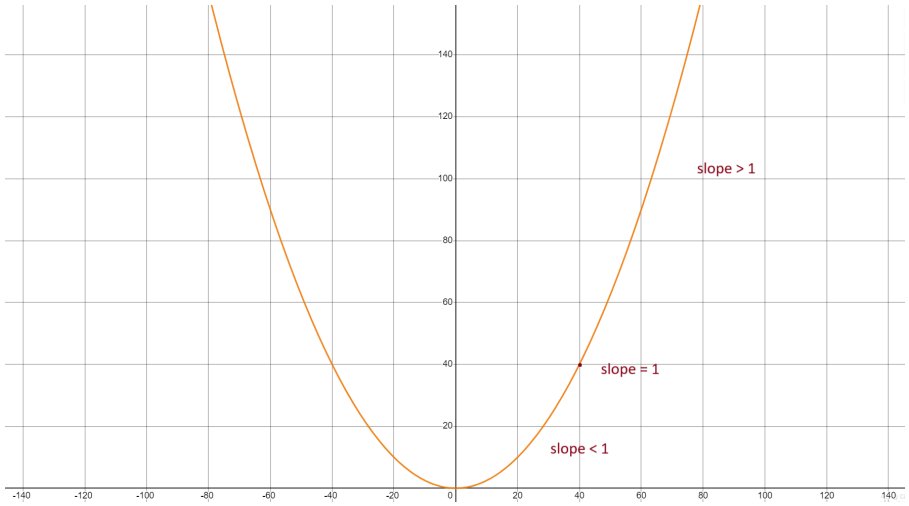


Hình 12: Kết quả minh họa của thuật toán

Nhận xét: Kết quả khá tốt, đối với những hình có kích thước vừa, tốc độ vẽ một hình xấp xỉ 1.5 milliseconds. Kết quả hình vẽ không bị đứt đoạn hay không có cảm giác các pixel chênh lệch nhiều. Thuật toán giải quyết được trường hợp đặc biệt là hình elipse trở thành hình tròn (A = B).

3.5 Midpoint parabol algorithm

3.5.1 Công thức đường parabol



Hình 13: Hình minh họa cho một parabol

Có nhiều cách để biểu diễn một parabol, tuy nhiên để thuận tiện cho việc tính toán và vẽ hình, công thức sau đây được sử dụng để xây dựng thuật toán midpoint parabol algorithm:

$$F(x, y) = x^4 - 4fy \tag{15}$$

Trong đó:

- Tọa độ đỉnh là O(0, 0)
- f là khoảng cách từ đỉnh tới tiêu cự

Nếu F(x, y) $\begin{cases} < 0 \text{ thì (x, y) nằm bên trên đường parabol} \\ = 0 \text{ thì (x, y) nằm trên parabol} \\ > 0 \text{ thì (x, y) nằm bên dưới đường parabol} \end{cases}$

Như được trên hình minh họa 13, để vẽ được một hình parabol, ta cần vẽ được đường cong bên phải trục Oy, sau đó ánh xạ qua phía bên kia. Đường cong này chia thành 2 phần, trong đó phần 1 có $slope < 1$ và phần 2 có $slope > 1$.

3.5.2 Trường hợp $slope < 1$

Giả sử ta đã vẽ điểm (x_k, y_k) và cần vẽ điểm tiếp theo. Điểm tiếp theo có thể là $P(x_k + 1, y_k)$ hoặc $Q(x_k + 1, y_k + 1)$.

Đặt $p_i = F(x_i + 1, y_i + \frac{1}{2}) = (x_i + 1)^2 - 4f(y_i + \frac{1}{2})$.

Nếu $\begin{cases} \text{p}_i < 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i + 1, y_i) \\ \text{p}_i \geq 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i + 1, y_i + 1) \end{cases}$

Tính tham số quyết định cho pixel tiếp theo dựa trên pixel trước:

$$p_{i+1} = p_i + (2x_i + 3) - 4f(y_{i+1} - y_i) \tag{16}$$

Với $\begin{cases} p_i < 0 \Rightarrow y_{i+1} = y_i & \Rightarrow p_{i+1} = p_i + (2x_i + 3) \\ p_i \geq 0 \Rightarrow y_{i+1} = y_i + 1 & \Rightarrow p_{i+1} = (2x_i + 3) - 4f \end{cases}$

Hay $\begin{cases} p_i < 0 \Rightarrow y_{i+1} = y_i & \Rightarrow p_{i+1} = p_i + 2x_{i+1} + 1 \\ p_i \geq 0 \Rightarrow y_{i+1} = y_i - 1 & \Rightarrow p_{i+1} = p_i + 2x_{i+1} + 1 - 4f \end{cases}$

Trong đó, giá trị khởi đầu của p_i tại điểm (0, 0) là:

$$p_0 = (0+1)^2 - 4f(0+\frac{1}{2}) \tag{17}$$

$$= 1 - 2f \tag{18}$$

3.5.3 Trường hợp *slope* > 1

Giả sử ta đã vẽ điểm (x_k, y_k) và cần vẽ điểm tiếp theo. Điểm tiếp theo có thể là $P(x_k, y_k + 1)$ hoặc $Q(x_k + 1, y_k + 1)$.

Đặt $q_i = F(x_i + \frac{1}{2}, y_i + 1) = (x_i + \frac{1}{2})^2 - 4f(y_i + 1)$.

Nếu $\begin{cases} q_i > 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i, y_i + 1) \\ q_i \leq 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i + 1, y_i + 1) \end{cases}$

Tính tham số quyết định cho pixel tiếp theo dựa trên pixel trước:

$$q_{i+1} = q_i + (x_{i+1} - x_i)(x_{i+1} + x_i + 1) - 4f \tag{19}$$

Với $\begin{cases} q_i > 0 \Rightarrow x_{i+1} = x_i & \Rightarrow q_{i+1} = q_i - 4f \\ q_i \leq 0 \Rightarrow x_{i+1} = x_i + 1 & \Rightarrow q_{i+1} = q_i + 2x_i + 2 - 4f \end{cases}$

Hay $\begin{cases} q_i > 0 \Rightarrow x_{i+1} = x_i & \Rightarrow q_{i+1} = q_i - 4f \\ q_i \leq 0 \Rightarrow x_{i+1} = x_i + 1 & \Rightarrow q_{i+1} = q_i + 2x_{i+1} - 4f \end{cases}$

Trong đó, giá trị khởi đầu của q_i tại điểm (x_k, y_k) (điểm kết thúc đường cong 1) là:

$$q_0 = (x_k + \frac{1}{2})^2 - 4f(y_k + 1) \tag{20}$$

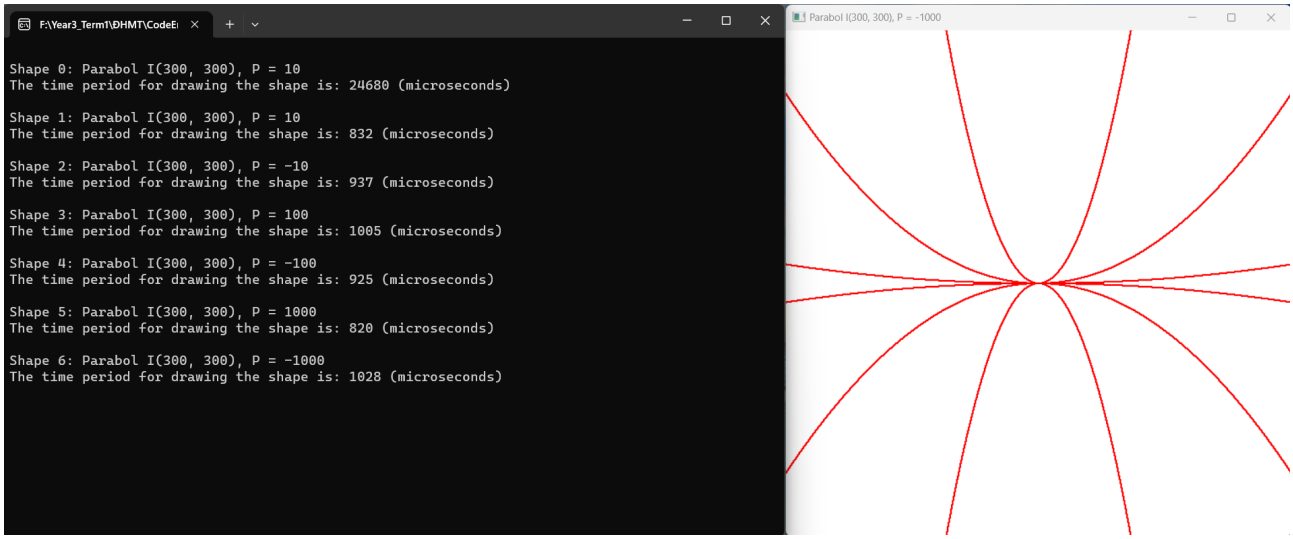
3.5.4 Thuật toán tổng quát

Data: Đỉnh (xt, yt) và khoảng cách từ đỉnh tới tiêu cự f
Result: Các điểm trên parabol

If $f = 0$ return;
$sign \leftarrow \begin{cases} 1 & \text{nếu } f > 0 \\ -1 & \text{nếu } f < 0 \end{cases}$;
$f \leftarrow abs(f)$; $x \leftarrow 0$, $y \leftarrow 0$; $dx \leftarrow 2x$, $dy \leftarrow -4f$;
$p1 \leftarrow 1 - 2f$;
while $dx < dy$ **do**
    `setPixel`$(xt + x,\ yt + \text{sign} \cdot y)$; `setPixel`$(xt - x,\ yt + \text{sign} \cdot y)$;
    $x \leftarrow x + 1$, $dx \leftarrow dx + 2$;
    if $p1 < 0$ **then**
        $p1 \leftarrow p1 + dx + 1$;
    end
    else
        $y \leftarrow y + 1$;
        $p1 \leftarrow p1 + dx + 1 + dy$;
    end
end
$p2 \leftarrow (x + 0.5)^2 - 4f(y + 1)$;
while $y < UPPER\_LIMIT$ **do**
    `setPixel`$(xt + x,\ yt + \text{sign} \cdot y)$; `setPixel`$(xt - x,\ yt + \text{sign} \cdot y)$;
    $y \leftarrow y + 1$;
    if $p2 > 0$ **then**
        $p2 \leftarrow p2 + dy$;
    end
    else
        $x \leftarrow x + 1$, $dx \leftarrow dx + 2$;
        $p2 \leftarrow p2 + dx + dy$;
    end
end

Algorithm 7: Midpoint parabol algorithm

3.5.5 Kết quả minh họa thuật toán

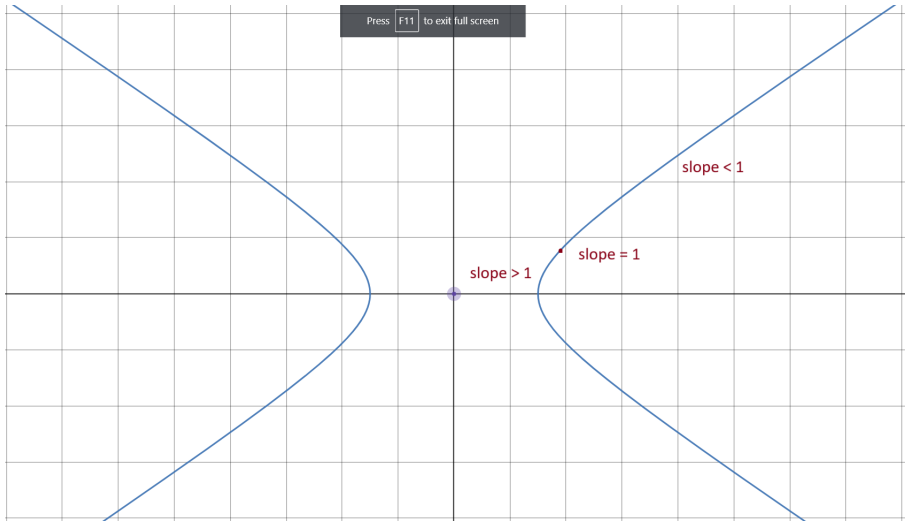


Hình 14: Kết quả minh họa của thuật toán

Nhận xét: Kết quả khá tốt, đối với những hình có kích thước vừa, tốc độ vẽ một hình xấp xỉ 1 millisecond. Kết quả hình vẽ không bị đứt đoạn hay không có cảm giác các pixel chênh lệch nhiều. Giải quyết được nhiều trường hợp của đường parabol.

3.6 Midpoint hyperbol algorithm

3.6.1 Công thức đường hyperbol



Hình 15: Hình minh họa cho một hyperbol

Có nhiều cách để biểu diễn một hyperbol, tuy nhiên để thuận tiện cho việc tính toán và vẽ hình, công thức sau đây được sử dụng để xây dựng thuật toán midpoint hyperbol algorithm:

$$F(x, y) = b^2x^2 - a^2y^2 - a^2b^2 \tag{21}$$

Trong đó:

- Tọa độ đỉnh là O(0, 0)
- 2a là độ dài trục thực
- 2b là độ dài trục ảo

Nếu F(x, y) $\begin{cases} < 0 \text{ thì (x, y) nằm bên trên đường hyperbol} \\ = 0 \text{ thì (x, y) nằm trên hyperbol} \\ > 0 \text{ thì (x, y) nằm bên dưới đường hyperbol} \end{cases}$

Như được trên hình minh họa 15, để vẽ được một hình hyperbol, ta cần vẽ được đường cong ở góc phần tư thứ nhất, sau đó ánh xạ qua các góc phần tư còn lại. Đường cong này chia thành 2 phần, trong đó phần 1 có $slope > 1$ và phần 2 có $slope < 1$.

3.6.2 Trường hợp $slope > 1$

Giả sử ta đã vẽ điểm (x_i, y_i) và cần vẽ điểm tiếp theo. Điểm tiếp theo có thể là $P(x_i, y_i + 1)$ hoặc $Q(x_i + 1, y_i + 1)$.

Đặt $q_i = F(x_i + \frac{1}{2}, y_i + 1) = b^2(x_i + \frac{1}{2})^2 - a^2(y_i + 1)^2 - a^2b^2$.

Nếu $\begin{cases} q_i > 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i, y_i + 1) \\ q_i \leq 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i + 1, y_i + 1) \end{cases}$

Tính tham số quyết định cho pixel tiếp theo dựa trên pixel trước:

$$q_{i+1} = q_i + b^2(x_{i+1} - x_i)(x_{i+1} + x_i + 1) - a^2(2y_i + 3) \tag{22}$$

Với $\begin{cases} q_i > 0 \Rightarrow x_{i+1} = x_i & \Rightarrow q_{i+1} = q_i - a^2(2y_i + 3) \\ q_i \leq 0 \Rightarrow x_{i+1} = x_i + 1 & \Rightarrow q_{i+1} = q_i + b^2(2x_i + 2) - a^2(2y_i + 3) \end{cases}$

Hay $\begin{cases} q_i > 0 \Rightarrow x_{i+1} = x_i & \Rightarrow q_{i+1} = q_i - 2a^2y_{i+1} - a^2 \\ q_i \leq 0 \Rightarrow x_{i+1} = x_i + 1 & \Rightarrow q_{i+1} = q_i + 2b^2x_{x+1} - 2a^2y_{i+1} - a^2 \end{cases}$

Trong đó, giá trị khởi đầu của q_i tại điểm $(a, 0)$ là:

$$q_0 = b^2(a + \frac{1}{2})^2 - a^2(0 + 1)^2 - a^2b^2 \tag{23}$$

$$= ab^2 - a^2 + 0.25b^2 \tag{24}$$

3.6.3 Trường hợp *slope* < 1

Giả sử ta đã vẽ điểm (x_i, y_i) và cần vẽ điểm tiếp theo. Điểm tiếp theo có thể là $P(x_i + 1, y_i)$ hoặc $Q(x_i + 1, y_i + 1)$.

Đặt $p_i = F(x_i + 1, y_i + \frac{1}{2}) = b^2(x_i + 1)^2 - a^2(y_i + \frac{1}{2})^2 - a^2b^2$.

Nếu $\begin{cases} \mathrm{p}_i < 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i + 1, y_i) \\ \mathrm{p}_i \geq 0 \Rightarrow \text{Điểm tiếp theo là: } (x_i + 1, y_i + 1) \end{cases}$

Tính tham số quyết định cho pixel tiếp theo dựa trên pixel trước:

$$p_{i+1} = p_i + b^2(2x_i + 3) - a^2(y_{i+1} - y_i)(y_{i+1} + y_i + 1) \tag{25}$$

Với $\begin{cases} p_i < 0 \Rightarrow y_{i+1} = y_i & \Rightarrow p_{i+1} = p_i + b^2(2x_i + 3) \\ p_i \geq 0 \Rightarrow y_{i+1} = y_i + 1 & \Rightarrow p_{i+1} = p_i + b^2(2x_i + 3) - a^2(2y_i + 2) \end{cases}$

Hay $\begin{cases} p_i < 0 \Rightarrow y_{i+1} = y_i & \Rightarrow p_{i+1} = p_i + 2b^2x_{i+1} + b^2 \\ p_i \geq 0 \Rightarrow y_{i+1} = y_i + 1 & \Rightarrow p_{i+1} = p_i + 2b^2x_{i+1} + b^2 - 2a^2y_{i+1} \end{cases}$

Trong đó, giá trị khởi đầu của p_i tại điểm (x_k, y_k) là:

$$p_0 = b^2(x_k + 1)^2 - a^2(y_k + \frac{1}{2})^2 - a^2b^2 \tag{26}$$

3.6.4 Thuật toán tổng quát

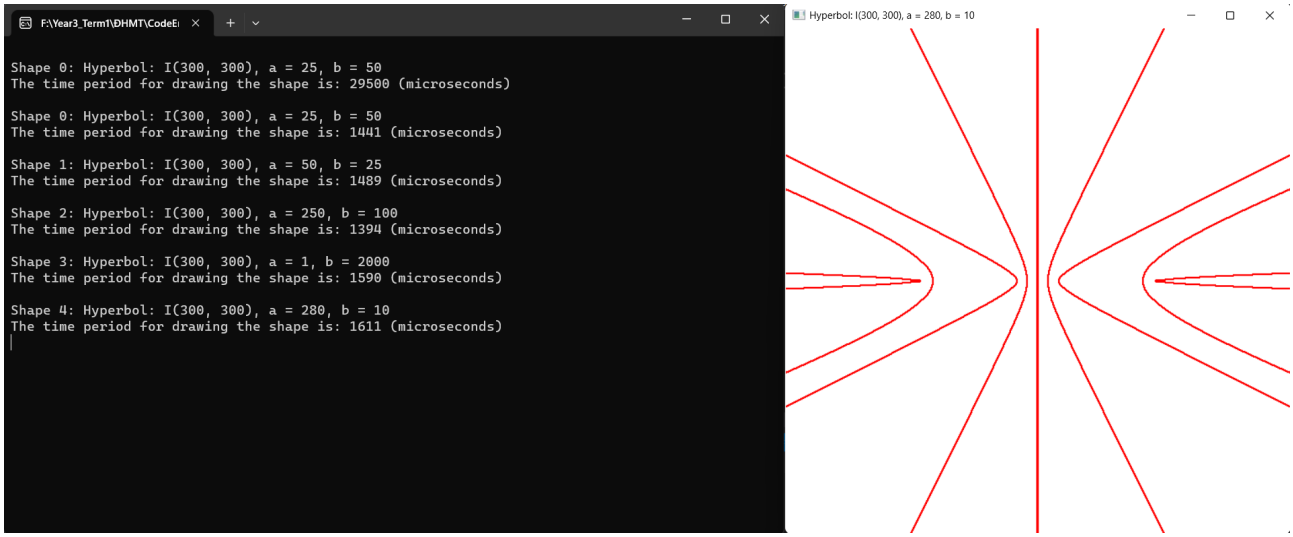
Data: Đỉnh (xt, yt) và bán trục thực a và bán trục ảo b
Result: Các điểm trên hyperbol

$upper\_bound \leftarrow \begin{cases} MAX\_LEN & \text{nếu } b \geq a \\ \infty & \text{ngược lại} \end{cases}$;
$x \leftarrow a$, $y \leftarrow 0$; $dx \leftarrow 2b^2x$; $dy \leftarrow -2a^2y$;
$p1 \leftarrow ab^2 - a^2 + 0.25b^2$;
while $dy < dx$ ***and*** $y < upper\_bound$ **do**
    setPixel$(xt + x, yt + y)$; setPixel$(xt + x, yt - y)$;
    setPixel$(xt - x, yt + y)$; setPixel$(xt - x, yt - y)$;
    $y \leftarrow y + 1$; $dy \leftarrow dy - 2a^2$;
    if $p1 > 0$ **then**
        $p1 \leftarrow p1 + dy - a^2$;
    end
    else
        $x \leftarrow x + 1$; $dx \leftarrow dx + 2b^2$;
        $p1 \leftarrow p1 + dx + dy - a^2$;
    end
end
$p2 \leftarrow b^2(x + 1)^2 - a^2(y + 0.5)^2 - a^2b^2$;
while $x < MAX\_LEN$ **do**
    setPixel$(xt + x, yt + y)$; setPixel$(xt + x, yt - y)$;
    setPixel$(xt - x, yt + y)$; setPixel$(xt - x, yt - y)$;
    $x \leftarrow x + 1$; $dx \leftarrow dx + 2b^2$;
    if $p2 < 0$ **then**
        $p2 \leftarrow p2 + dx + b^2$;
    end
    else
        $y \leftarrow y + 1$; $dy \leftarrow dy - 2a^2$;
        $p2 \leftarrow p2 + dx + b^2 + dy$;
    end
end

Algorithm 8: Midpoint hyperbol algorithm

Trong thuật toán 8, với $b \geq a$ thì dy sẽ luôn nhỏ hơn dx nên vòng lặp sẽ vô hạn, vì vậy ta đặt một giới hạn trên cho y để có thể dừng. Tương tự đối với x, ta cũng cần đặt một giới hạn trên để dừng vòng lặp.

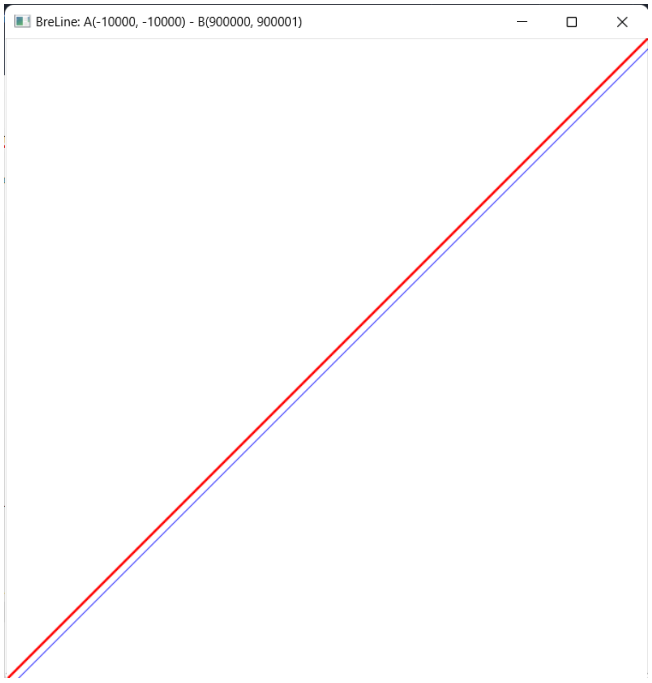
3.6.5 Kết quả minh họa thuật toán



Hình 16: Kết quả minh họa của thuật toán

Nhận xét: Kết quả khá tốt, đối với những hình có kích thước vừa, tốc độ vẽ một hình xấp xỉ 1 millisecond. Kết quả hình vẽ không bị đứt đoạn hay không có cảm giác các pixel chênh lệch nhiều.

3.7 So sánh thuật toán vẽ đoạn thẳng giữa DDA, Bresenham và OpenGL



Hình 17: Hình vẽ so sánh kết quả hiển thị của các thuật toán. Trong đó đường màu đỏ là đường được vẽ bằng thuật toán Bresenham và đường màu xanh là sử dụng hàm của OpenGL để vẽ.

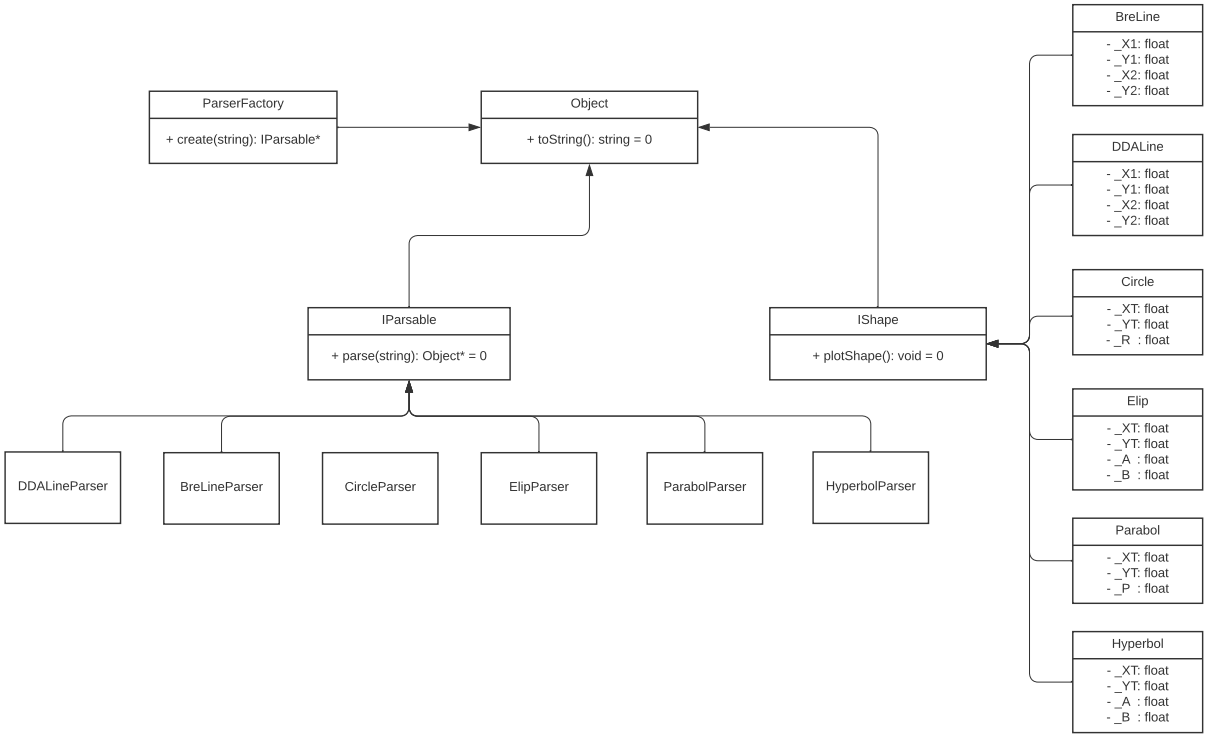
X1	Y1	X2	Y2	DDA	Bresenham	OpenGL
100	100	100	500	127	147	1
100	100	100	500	149	117	4
100	100	500	500	118	110	4
100	100	500	300	199	94	4
100	100	500	100	198	138	6
-10000	-10000	900000	900001	114104	69601	4

Bảng 1: Bảng so sánh thời gian (microseconds) chạy thuật toán vẽ đoạn thẳng giữa DDA, Bresenham và OpenGL

Nhận xét: Dựa vào kết quả được thể hiện ở bảng trên ta thấy sự khác biệt rõ rệt về tốc độ giữa thuật toán tự triển khai và thuật toán của thư viện OpenGL. Đối với các

hình có kích thước nhỏ, chênh lệch là không đáng kể. Tuy nhiên, đối với các hình có kích thước lớn, ta thấy tốc độ vẽ của hàm trong thư viện OpenGL là gấp rất nhiều lần so với hàm tự cài đặt.

4 Cấu trúc chương trình



Chương trình được thiết kế dựa trên Factory Method Design Pattern. Factory Method là một trong những mẫu thiết kế (design pattern) trong lập trình hướng đối tượng, thuộc nhóm Creational Design Pattern. Mẫu thiết kế này được sử dụng để tạo ra các đối tượng mà concreteness của chúng được quyết định bởi các lớp con (subclasses), trong khi việc tạo ra đối tượng được kiểm soát bởi một phương thức gọi là "Factory Method". Factory Method cho phép tạo ra các đối tượng mà không cần biết cụ thể lớp con nào sẽ được tạo.

Chương trình được chia thành hai phần chính để quản lý và tạo ra các đối tượng. Phần thứ nhất là các thực thể (Entity) như DDALine, BreLine, Circle, Elip, Parabol, Hyperbol. Những lớp này đảm nhiệm nhiệm vụ riêng là vẽ hình.

Phần thứ hai của chương trình là các Parsers. Các Parsers có nhiệm vụ tạo ra các thực thể (Entity) dựa trên loại (type) được đưa vào. Các Parsers này sử dụng Factory

Method Pattern để tạo ra đối tượng cụ thể mà chúng cần, dựa trên thông tin về loại (type) mà người dùng cung cấp. Việc này tạo ra sự linh hoạt và giúp dễ dàng mở rộng chương trình bằng cách thêm các lớp mới mà không cần sửa đổi mã nguồn chính của chương trình.

```
// Ví dụ dữ liệu: 0 0 0 1 1
ParserFactory factory;                          // Sử dụng để tạo Parsers
IParsable* parser = factory.create(type);       // Type là kiểu để xác định Entity sẽ được tạo (Type: 0)
IShape* shape = (IShape*)parser->parse(data);   // Data là là dữ liệu của Entity,
                                                // có cấu trúc nhất định để đưa vào constructor
                                                // giúp tạo Entity (data: 0 0 1 1)
```

Hình 18: Ví dụ sử dụng ParserFactory để tạo một thực thể

5 Các nguồn tham khảo

[1] Wikipedia/OpenGL
[2] JavaTpoint
[3] uobabylon