ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

ĐỒ ÁN MÔN HỌC ĐỒ HỌA MÁY TÍNH VÀ XỬ LÝ ẢNH:

MÔ PHỎNG CHƯƠNG TRÌNH PAINT TRÊN MÁY TÍNH

Nhóm thực hiện: Lưu Thanh Sơn – MSSV: 14520772
Đỗ Phú An – MSSV: 14520002
Giảng viên hướng dẫn: TS Ngô Đức Thành

Lớp: Đồ họa máy tính và xử lý ảnh (CS113.G21.KHTN)

Thành phố Hồ Chí Minh, ngày 15 tháng 6 năm 2016

MỤC LỤC:

CHƯƠNG 1: TỔNG QUAN VỀ ĐỒ HỌA MÁY TÍNH 4

1. Khái niệm Đồ họa máy tính 4

2. Hệ đồ họa cơ bản 5

a. Màn hình: 5

b. Các hệ màu hiên nay: 5

c. Các thiết bị nhập: 6

d. Phần mềm: 6

3. Tổng quan về Đồ án 6

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT 7

1. Các đối tượng đồ họa cơ sở 7

a. Hệ tọa độ: 7

b. Điểm: 7

c. Đoạn thẳng: 7

d. Vùng tô 8

2. Các phép biến đổi trong đồ họa 2D 8

a. Hệ tọa độ thuần nhất (hormogenerous Coordinate): 8

b. Phép tịnh tiến: 8

c. Phép biến đổi tỉ lệ: 8

d. Phép quay: 8

e. Phép lấy đối xứng: 9

f. Phép biến dạng: 9

g. Tính chất phép biến đổi Affine: 9

3. Hiển thị đối tượng 2 chiều 9

a. Khái niệm: 9

b. Quy trình hiển thị đối tượng 2 chiều: 9

CHƯƠNG 3: CƠ SỞ THUẬT TOÁN 10

1. Thuật toán vẽ đường thẳng Bresenha 10

2. Thuật toán vẽ đường tròn Mid-Point 11

3. Thuật toán Tô màu Boundary Fill 13

4. Thuật toán xén hình Cohen – Sutherland 17

CHƯƠNG 4: CẤU TRÚC CHƯƠNG TRÌNH 21

1. Tổng quan về Lập trình MFC 21

2. Giao diện chương trình. 24

3. Cấu trúc chương trình 25

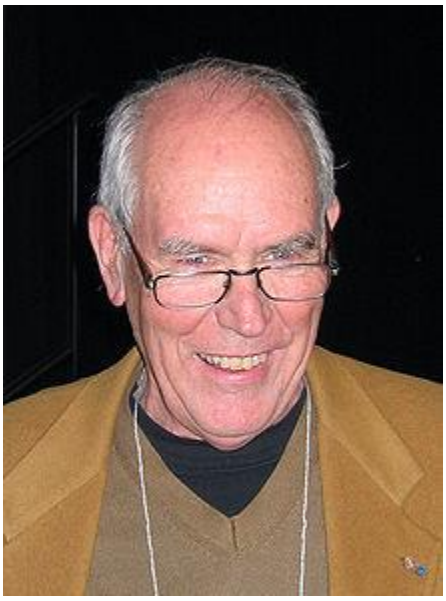
a. Lớp C_Circle: 25

b. Lớp C_Elip: 26
c. Lớp C_FillColor: 27
d. Lớp C_Line 29
e. Lớp Rectangle: 31
f. Lớp Square: 32
g. Lớp Screen: 33
e. Lớp CpaintAppView: 34
f. Lớp MainFrame: 45
g. Save và Open File: 47
4. Đánh giá chương trình 48
CHƯƠNG 5: BẢNG PHÂN CÔNG THỰC HIỆN ĐỒ ÁN 49
TÀI LIỆU THAM KHẢO 50

CHƯƠNG 1: TỔNG QUAN VỀ ĐỒ HỌA MÁY TÍNH

1. Khái niệm Đồ họa máy tính

- Đồ họa máy tính (Computer Graphic) là tất cả những gì có liên quan đến việc sử dụng máy tính để phát sinh ra hình ảnh. Các vấn đề liên quan đến đồ họa máy tính bao gồm: tạo, lưu trữ và các thao tác trên các mô hình và các ảnh.
- Cha đẻ của ngành Đồ họa máy tính là Giáo sư Ivan Sutherland. Với luận án tiến sĩ "Sketchpad. a Man machine graphical communication system" làm phương tiện để vẽ trên màn hình máy tính đầu tiên, ông được xem là nhà tiên phong trong lĩnh vực Đồ họa máy tính.



Ivan Sutherland (2006).

Thiết bị vẽ trên máy tính đầu tiên.

2. Hệ đồ họa cơ bản

Một hệ đồ họa cơ bản bao gồm phần cứng và phần mềm. Phần cứng bao gồm các thiết bị nhập/xuất dữ liệu. Phần mềm bao gồm các công cụ lập trình và các ứng dụng về đồ họa.

a. *Màn hình:*

- Màn hình dạng điểm (pixel) là dạng thường gặp nhất trong số các dạng màn hình sử dụng cho việc Đồ họa hiện nay.
- Sự bật tắt cũng như cường độ áng của một tập hợp các điểm ảnh trên màn hình là cơ sở cho việc hiển thị hình ảnh.
- Có 2 loại màn hình phổ biến là màn hình CRT và màn hình LCD
    - **Màn hình CRT (CRT – Cathode Ray Tupe):** Màn hình CRT sử dụng phần màn huỳnh quang dùng để hiển thị các điểm ảnh, để các điểm ảnh phát sáng theo đúng màu sắc cần hiển thị cần các tia điện tử tác động vào chúng để tạo ra sự phát xạ ánh sáng. Ống phóng CRT sẽ tạo ra các tia điện tử đập vào màn huỳnh quang để hiển thị các điểm ảnh theo mong muốn.
    - **Màn hình LCD (Liquid Crystal Display):** Nếu điện cực của một điểm ảnh con không được áp một điện thế, thì phần tinh thể lỏng ở nơi ấy không bị tác động gì cả, ánh sáng sau khi truyền qua chỗ ấy vẫn giữ nguyên phương phân cực, và cuối cùng bị chặn lại hoàn toàn bởi kính lọc phân cực thứ hai. Điểm ảnh con này lúc đó bị tắt và đối với mắt đây là một điểm tối. Để bật một điểm ảnh con, cần đặt một điện thế vào điện cực của nó, làm thay đổi sự định hướng của các phân tử tinh thể lỏng ở nơi ấy; kết quả là ánh sáng sau khi truyền qua phần tinh thể lỏng ở chỗ điểm ảnh con này sẽ bị xoay phương phân cực đi, có thể lọt qua lớp kính lọc phân cực thứ hai, tạo ra một điểm màu trên tấm kính trước.
- Các thông tin về hình ảnh hiển thị được lưu trữ trong một vùng nhớ gọi là vùng đệm làm tươi (refresh buffer) hoặc là vùng đệm khung (frame buffer). Vùng nhớ này lưu trữ các giá trị độ sáng trên màn hình. Do đó, vùng nhớ này cần phải được "làm tươi" liên tục để đáp ứng sự thay đp63i liên tực của các hình ảnh hiển thị. Tốc độ làm tươi của màn hình được tính bằng Hertz (Hz). Giá trị làm tươi của các màn hình hiện nay khoảng 60-80 Hz.

b. *Các hệ màu hiên nay:*

- Việc nghiên cứu màu sắc bao gồm nhiều lĩnh vực như: quang học (độ sáng, bước sóng, năng lượng,…), sinh lý học (sự cảm nhận về màu sắc của con người), tâm lý học, … và nhiều nhân tố khác. Đối với những người làm về Tin học, vấn đề chính là mối tương tác qua lại giữa sự cảm nhận màu sắc của con người với các bộ phận hiển thị màu sắc trên máy tính và các phần mềm thiết kế trên nó.
- Bảng sau sẽ cho biết mối tương quan giữa sự cảm nhận của con người về màu sắc với các bộ phận hiển thị hình ảnh và phần mềm thiết kế đồ họa:

Sự cảm nhận của con người	**Phần cứng**	**Phần mềm**
Màu sắc (Color)	Các màu gốc hiển thị (thường là Đỏ, Lục, Lam)	Thuật toán trên không gian màu
Sắc độ màu (HUE)	Bước sóng	
Độ bão hòa (Saturation)	Sự thuần nhất của màu	
Độ sáng (Lightness)	Cường độ sáng	Hiệu chỉnh Gammar
Sự rung của màn hình	Tốc độ làm tươi	

- Không gian màu được đưa ra để định các màu hiển thị trên máy tính vì chúng làm đơn giản hóa các thao tác tính toán cần thiết cho việc chuyển đổi giữa các màu. Có các không gian màu thường dùng sau:
    - Không gian **RGB**: Gồm 3 thành phần: Red (đỏ), Green (lục), Blue (lam). Không gian này được minh họa bằng một khối lập phương với các trục R,G,B.
    - Không gian **HSL**: Không gian này chú trọng hơn đến các thành phần của sự cảm nhận màu sắc ở mắt (Hue, Sarturation, Lightness). Không gian được biểu diễn trong hệ tọa độ trụ, với 2 hình nón úp vào nhau. S (Saturation) là tọa độ gốc, L (Lightness) là trục thẳng đứng và H (Hue) là tọa độ tương ứng với góc quay.
    - Không gian màu **CMYK**: Gồm 4 thành phần màu: Cyan (xanh lơ), Magneta (hồng), Yellow (vàng), Key (đen). Nguyên lý làm việc của hệ màu này dựa trên cơ sở hấp thụ ánh sáng. Do đó, đây là hệ màu tiêu chuẩn dùng cho việc in ấn hiện nay

c. Các thiết bị nhập:

- Bàn phím: Xuất hiện trong hầu hết các máy tính ngay từ buổi sơ khai. Đây là loại thiết bị nhập cơ bản nhất, nhưng sự tương tác không cao và không uyển chuyển do người dùng chỉ có thể nhập 2 loại dữ liệu: số và ký tự.
- Chuột máy tính: được phát minh bởi Douglas Engelbart, là thiết bị nhập dùng thao tác trỏ (Point) và chọn (Click) với các chức năng phù hợp với yêu cầu người dùng. Đây là cách giao tiếp thân thiện và uyển chuyển với người dùng. Cùng với bàn phím, chuột là thiết bị nhập thông dụng nhất hiện nay và được xem như là 1 thành phần không thể thiếu của một hệ thống máy tính tiêu chuẩn.
- Ngoài ra, hiện nay, chúng ta còn có một số thiết bị nhập khác phục vụ cho việc đồ họa. Điển hình là bàn vẽ (Sketchpad) dùng cho việc thiết kế và vẽ trên máy tính

d. Phần mềm:

- Phần mềm đồ họa chia làm hay loại: các công cụ hỗ trợ lập trình đồ họa và các phần mềm ứng dụng đồ họa:
    - Công cụ hỗ trợ lập trình đồ họa. Các ngôn ngữ như: C/C++, Pascal, Java,.. với các thư viện đồ họa hỗ trợ như: OpenGL, OpenCV, DirectX, Swing, Aforce, … và thường dùng cho các lập trình viên trong việc thiết kế những ứng dụng về Đồ họa (Game, giả lập, ….)
    - Phần mềm đồ họa: Là những phần mềm hỗ trợ cho người dùng các công việc liên quan về Đồ họa như: Photoshop dùng cho việc chỉnh sửa ảnh, Illustrator dùng cho việc thiết kế đồ họa 2D, AutoCAD dùng cho việc thiết kế bản vẽ 3D,….

3. Tổng quan về Đồ án

Mục tiêu của Đồ án này là áp dụng các kiến thức về Đồ họa để xây dựng chương trình mô phỏng lại chương trình Paint đã có sẵn trong hệ điều hành Windows – là một chương trình dùng để vẽ các đối tượng đồ họa cơ bản (điểm, đường thẳng, hình tròn, hình vuông,…) lên máy tính.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

1. Các đối tượng đồ họa cơ sở

a. Hệ tọa độ:

- Hệ tọa độ thế giới thực (Real Coordiante) là hệ tọa độ dùng để mô tả các đối tượng trong thế giới thực. Một trong các hệ tọa độ thường dùng là hệ tọa độ Descartes. Với hệ tọa độ này, bất kỳ 1 điểm nào trong mặt phẳng cũng được biểu diễn bằng một cặp số thực (x,y) với x là hoành độ, còn y là tung độ. Gốc tọa độ là điểm O(0,0)
- Hệ tọa độ thiết bị (Device Coordinate) là hệ tọa độ được sử dụng cho một thiết bị cụ thể như: máy in, màn hình,…. Đặc điểm chung của hệ tọa độ thiết bị là:
    - Các điểm trong hệ tọa độ thiết bị được mô tả bằng một cặp số Nguyên (x,y). Điều này được giải thích là cho các điểm trong hệ tọa độ thực được định nghĩa liên tục, trong khi trong thiết bị là các tập rời rạc
    - Các tọa độ x, y trong hệ tọa độ thiết bị không thể lớn tùy ý mà bị giới hạn trong một khoảng nào đó. Ví dụ như màn hình HD thì $x \in [0,1280]$ và $y \in [0,720]$.

b. Điểm:

- Điểm là thành phần cơ sở được định nghĩa trong một hệ tọa độ. Đối với hệ tọa độ 2 chiều, một điểm được xác định bởi một cặp tọa độ (x,y).
- Ngoài thông tin về tọa độ, điểm còn có thuộc tính là màu sắc.

c. Đoạn thẳng:

- Một đoạn thẳng được xác định bởi 2 điểm thuộc nó gọi là điểm đầu và điểm cuối.
- Phương trình đường thẳng đi qua 2 điểm A(x1,y1) và B(x2,y2) được xác định như sau:

$$\frac{x-x_1}{y-y_1}=\frac{x_2-x_1}{y_2-y_1}$$

Hay ở dạng tương đương: $(x - x_1)(y_2 - y_1) = (y - y_1)(x_2 - x_1)$

Khai triển ta có dạng: $y = mx + b$ (1)

Trong đó: $m=\frac{Dy}{Dx}$

Với $\mathbf{Dy = y_2 - y_1}$ và $\mathbf{Dx = x_2 - x_1}$

$b = y_1 - mx_1$

Nếu khai triển dưới dạng: $(y_2 - y_1)x - (x_2 - x_1)y - x_1y_2 + x_2y_1 = 0$

Và đặt $A = y_2 - y_1$; $B = -(x_2 - x_1)$

Thì phương trình trên có dạng $Ax + By + C = 0$ (2).

(1) Còn được gọi là Phương trình Tổng quát của đoạn thẳng.

- Phương trình tham số của đường thẳng có dạng các tọa độ x,y được mô tả qua thành phần thứ 3 là t với $t \in [0;1]$.

$$\begin{cases} x=(1-t)x_1+tx_2 \\ y=(1-t)y_1+ty_2 \end{cases}$$

- Các thuộc tính của đoạn thẳng bao gồm:
    - Màu sắc
    - Độ rộng nét vẽ
    - Kiểu vẽ

(1) Phương trình đoạn chắn của đường thẳng

d. Vùng tô

- Một vùng tô bao gồm đường biên và vùng bên trong. Đường biên là một đường khép kín (như Đa giác)
- Các thuộc tính của Vùng tô:
  - Thuộc tính đường biên: là các thuộc tính như thuộc tính của đoạn thẳng.
  - Thuộc tính vùng bên trong: bao gồm màu tô và mẫu tô.

2. Các phép biến đổi trong đồ họa 2D

a. Hệ tọa độ thuần nhất (hormogenerous Coordinate):

- Tọa độ thuần nhất của một điểm trên mặt phẳng được biểu điễn bằng bộ 3 số tỉ lệ (x_h,y_h,h) không đồng thời bằng 0 và liên hệ với tọa độ (x,y) theo công thức:

$x = \frac{x_h}{h}$ và $y = \frac{y_h}{h}$

- Nếu một điểm có tọa độ thuần nhất là (x,y,z) thì nó cũng có tọa độ thuần nhất là (h.x,h.y,h.z), trong h=đó h là số khác 0 bất kỳ.
- Để đơn giản hóa, chúng ta có thể chọn h=1. Lúc này, mỗi điểm trên hệ tọa độ thuần nhất được biểu diễn dưới dạng tọa độ thuần nhất là (x,y,1).

b. Phép tịnh tiến:

- Ta có: Q = P.$M_T(tr_x,tr_y)$ với $M_T(tr_x,tr_y) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ tr_x & tr_y & 1 \end{pmatrix}$

⇨ $(x' \; y' \; 1) = (x \; y \; 1) . \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ tr_x & tr_y & 1 \end{pmatrix}$

c. Phép biến đổi tỉ lệ:

- Ta có: Q = P.$M_T(s_x,s_y)$ với $M_T(s_x,s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$

⇨ $(x' \; y' \; 1) = (x \; y \; 1). \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$

d. Phép quay:

- Ta có: Q = P.$M_T(\alpha)$ với $M_T(\alpha) = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$

⇨ $(x' \; y' \; 1) = (x \; y \; 1) . \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$

e. Phép lấy đối xứng:

- Ta có: $Q = P.M_T$ với $M_T = \begin{cases} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, Ox \\ \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, Oy \end{cases}$ (Ox – lấy đối xứng qua Ox, Oy – lấy đối xứng qua Oy)

f. Phép biến dạng:

- Ta có: $Q = P.M_T$ với $M_T = \begin{cases} \begin{pmatrix} 1 & 0 & 0 \\ sh_{xy} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, Ox \\ \begin{pmatrix} 1 & sh_{yx} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, Oy \end{cases}$ (Ox – biến dạng theo trục Ox, Oy – biến dạng theo trục Oy)

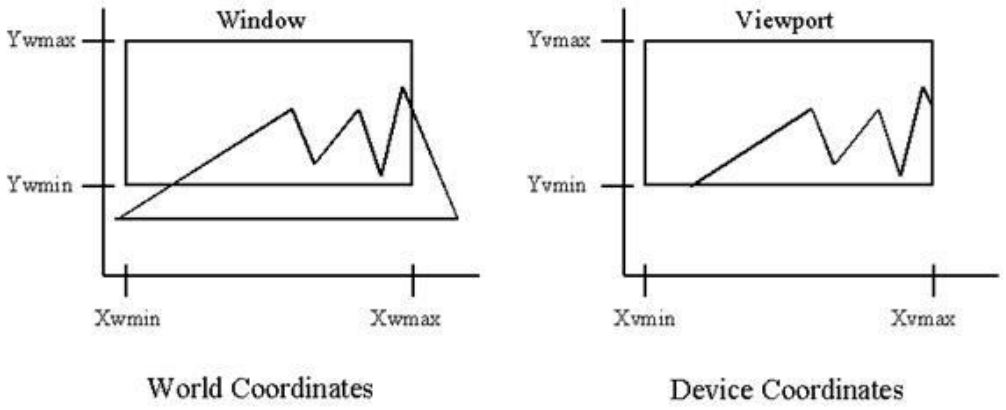
g. Tính chất phép biến đổi Affine:

- Phép biến đổi Affine bảo toàn đường thẳng.
- Tính song song của các đường thẳng được bảo toàn.
- Tính tỉ lệ về khoảng cách được bảo toàn.

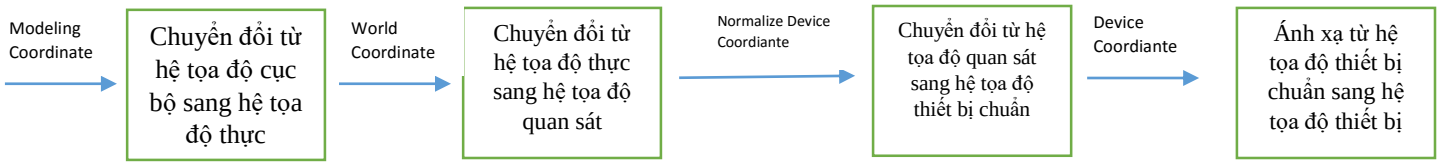
3. Hiển thị đối tượng 2 chiều

a. Khái niệm:

- Cửa sổ (Window) là vùng được chọn để hiển thị trong tọa độ thế giới thực.
- Vùng quan sát (Viewport) là vùng được chọn trên thiết bị để các đối tượng cửa sổ ánh xạ vào.

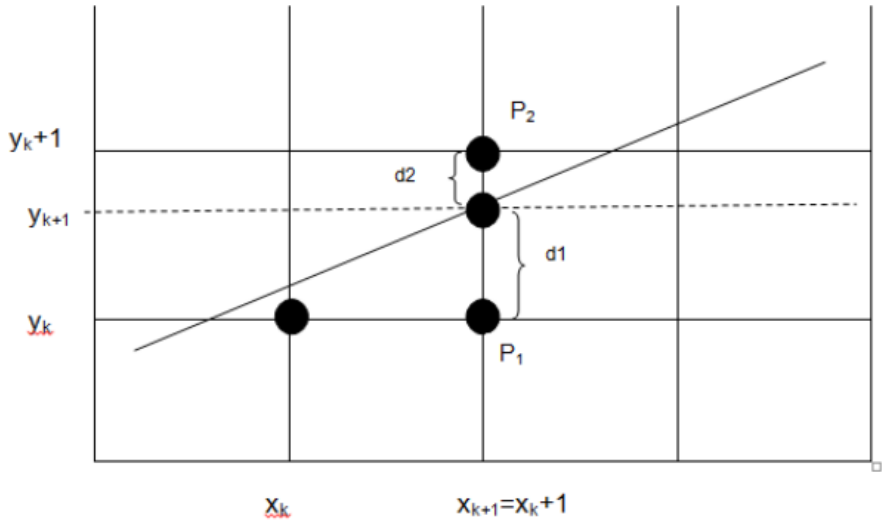


b. Quy trình hiển thị đối tượng 2 chiều:



CHƯƠNG 3: CƠ SỞ THUẬT TOÁN

1. Thuật toán vẽ đường thẳng Bresenha



- Thuật toán Bresenham đưa ra cách chọn y_{i+1} hay là y_i. Vấn đề mấu chốt là làm sao để hạn chế tối đa các phép toán trên số thực.
- Gọi $Dx = x_2 – x_1$, $Dy = y_2 – y_1$. Đặt $p_0 = 2Dy – Dx$.
- Xét dấu p_i:
  - Nếu $p < 0$: $p_{i+1} = p_i + 2Dy$.
  - Nếu $p >= 0$: $p_{i+1} = p_i + 2Dy – 2Dx$.

Thuật giải Bresenham:

```
function line(x0, x1, y0, y1)
    boolean steep:= abs(y1 - y0) > abs(x1 - x0)
    if steep then
       swap(x0, y0)
       swap(x1, y1)
    if x0 > x1 then
       swap(x0, x1)
       swap(y0, y1)
    int deltax:= x1 - x0
    int deltay:= abs(y1 - y0)
    real error:= 0
    real deltaerr:= deltay / deltax
    int ystep
    int y:= y0
    if y0 < y1 then ystep:= 1 else ystep:= -1
    for x from x0 to x1
       if steep then plot(y,x) else plot(x,y)
       error:= error + deltaerr
       if error ≥ 0.5 then
          y:= y + ystep
          error:= error - 1.0
```

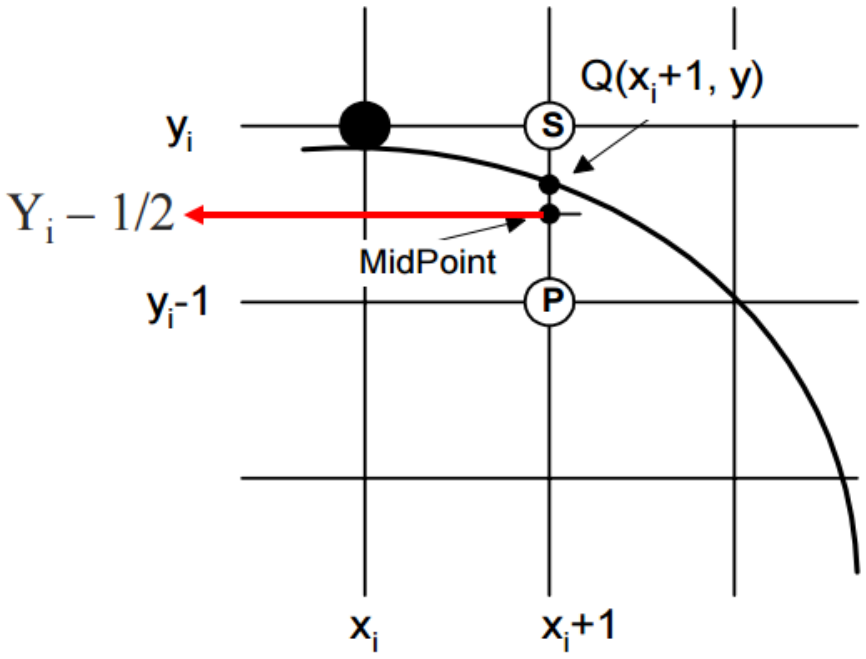
Thuật giải Bresenham đầy đủ cho đường thẳng vẽ theo mọi hướng:

```
function line(x0, y0, x1, y1)
  dx:= abs(x1-x0)
  dy:= abs(y1-y0)
  if x0 < x1 then sx:= 1 else sx:= -1
  if y0 < y1 then sy:= 1 else sy:= -1
  err:= dx-dy

  loop
    setPixel(x0,y0)
    if x0 = x1 and y0 = y1 exit loop
    e2:= 2*err
    if e2 > -dy then
      err:= err - dy
      x0:= x0 + sx
    end if
    if e2 <  dx then
      err:= err + dx
      y0:= y0 + sy
    end if
  end loop
```

Nhận xét:

- Thuật toán Bresenham chỉ làm việc trên số nguyên và các thao tác chỉ là phép cộng và dịch bit (nhân với 2) nên cho kết quả thực hiện nhanh.
- Thuật toán này cho kết quả tương tự thuật toán DDA (cũng là một thuật toán vẽ đường cơ bản nhưng độ phức tạp cao).

2. Thuật toán vẽ đường tròn Mid-Point



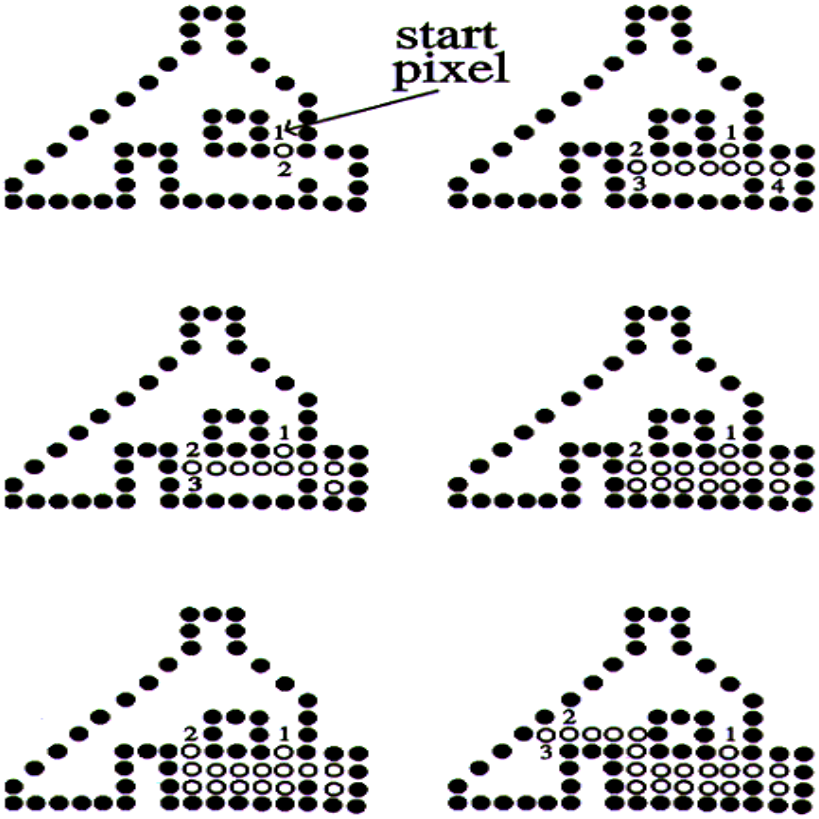
- Do tính đối xứng của đường tròn nên ta chỉ cần vẽ 1/8 đường tròn, sau đó lấy đối xứng.

  Cung 1/8 trên được mô tả như sau: $\begin{cases} 0 \le x \le R\dfrac{\sqrt{2}}{2} \\ R\dfrac{\sqrt{2}}{2} \le y \le R \end{cases}$
- Chọn điểm bắt đầu để vẽ là điểm (0,R). Đặt $p_0 = 1 - R$.
- Xét dấu p_i:
  - Nếu $p < 0$: $p_{i+1} = p_i + 2x_i + 3$.
  - Nếu $p >= 0$: $p_{i+1} = p_i + 2x_i - 2y_i + 5$.

Thuật toán Mid-Point (biểu diễn bằng mã C):

```
void drawcircle(int x0, int y0, int radius)
{
   int x = radius;
   int y = 0;
   int err = 0;

   while (x >= y)
   {
      putpixel(x0 + x, y0 + y);
      putpixel(x0 + y, y0 + x);
      putpixel(x0 - y, y0 + x);
      putpixel(x0 - x, y0 + y);
      putpixel(x0 - x, y0 - y);
      putpixel(x0 - y, y0 - x);
      putpixel(x0 + y, y0 - x);
      putpixel(x0 + x, y0 - y);

      y += 1;
      err += 1 + 2*y;
      if (2*(err-x) + 1 > 0)
      {
         x -= 1;
         err += 1 - 2*x;
      }
   }
}
```

3. Thuật toán Tô màu Boundary Fill



- Bắt đầu từ điểm nằm trong vùng tô, ta sẽ kiểm tra các điểm lân cận đã được tô màu có phải là điểm biên hay không, nếu không phải là điểm biên và không phải điểm đã tô thì ta sẽ tô mau cho nó. Quá trình này lặp lại cho tới khi nào không còn điểm nào tô màu được nữa.
- Bằng cách này, tất cả các điểm thuộc vùng tô sẽ được tô hết. Do tính chất của thuật toán này là các điểm trong vùng tô sẽ được tô loang dần ra tới biên nên nó còn được gọi là Thuật toán tô màu loang (Flood Fill).

Thuật toán tổng quát:

Flood-fill (node, target-color, replacement-color):
1. If *target-color* is equal to *replacement-color*, return.
2. If the color of *node* is not equal to *target-color*, return.
3. Set the color of *node* to *replacement-color*.
4. Perform **Flood-fill** (one step to the south of *node*, *target-color*, *replacement-color*).
   Perform **Flood-fill** (one step to the north of *node*, *target-color*, *replacement-color*).
   Perform **Flood-fill** (one step to the west of *node*, *target-color*, *replacement-color*).
   Perform **Flood-fill** (one step to the east of *node*, *target-color*, *replacement-color*).
5. Return.

- Do thuật toán này dựa trên việc gọi đệ quy để tô các điểm lân cận với nó, nên khi không gian tô màu lớn thì thuật toán này sẽ bị lỗi tràn bộ nhớ (overstack).
- Để khắc phục, ta sẽ tiến hành tô loang dần theo dòng quét ngang thay vì 4 điểm lân cận.

Thuật giải tổng quát:

Flood-fill (node, target-color, replacement-color):
1. Set *Q* to the empty queue.
2. If the color of *node* is not equal to *target-color*, return.
3. Add *node* to *Q*.
4. For each element *N* of *Q*:
5. Set *w* and *e* equal to *N*.
6. Move *w* to the west until the color of the node to the west of *w* no longer matches *target-color*.
7. Move *e* to the east until the color of the node to the east of *e* no longer matches *target-color*.
8. For each node *n* between *w* and *e*:
9. Set the color of *n* to *replacement-color*.
10. If the color of the node to the north of *n* is *target-color*, add that node to *Q*.
11. If the color of the node to the south of *n* is *target-color*, add that node to *Q*.
12. Continue looping until *Q* is exhausted.
13. Return.

Mã giả đầy đủ toàn bộ thuật toán:

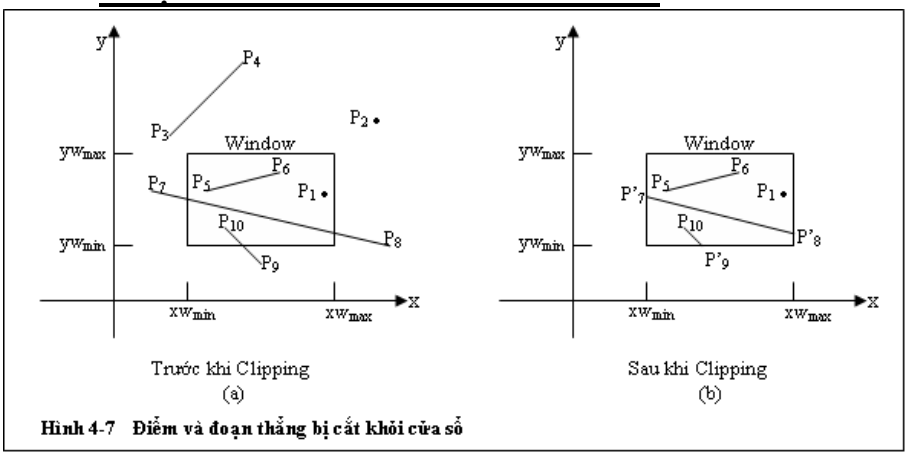
```
cur, mark, and mark2 each hold either pixel coordinates or a null value
      NOTE: when mark is set to null, do not erase its previous coordinate value.
            Keep those coordinates available to be recalled if necessary.
   cur-dir, mark-dir, and mark2-dir each hold a direction (left, right, up, or down)
   backtrack and findloop each hold boolean values
   count is an integer
```

```
set cur to starting pixel
set cur-dir to default direction
clear mark and mark2 (set values to null)
set backtrack and findloop to false

while front-pixel is empty
      move forward
end while

jump to START

MAIN LOOP:
      move forward
      if right-pixel is empty
            if backtrack is true and findloop is false and either front-pixel or left-pixel is empty
                  set findloop to true
            end if
            turn right
PAINT:
            move forward
      end if
```

```
START:
    set count to number of non-diagonally adjacent pixels filled (front/back/left/right ONLY)
    if count is not 4
        do
            turn right
        while front-pixel is empty
        do
            turn left
        while front-pixel is filled
    end if
    switch count
        case 1
            if backtrack is true
                set findloop to true
            else if findloop is true
                if mark is null
                    restore mark
                end if
            else if front-left-pixel and back-left-pixel are both empty
                clear mark
                fill cur
                jump to PAINT
            end if
        end case
        case 2
            if back-pixel is filled
                if front-left-pixel is not filled
                    clear mark
                    fill cur
                    jump to PAINT
                end if
            else if mark is not set
                set mark to cur
                set mark-dir to cur-dir
                clear mark2
                set findloop and backtrack to false
            else
                if mark2 is not set
                    if cur is at mark
                        if cur-dir is the same as mark-dir
                            clear mark
                            turn around
                            fill cur
                            jump to PAINT
                        else
                            set backtrack to true
                            set findloop to false
                            set cur-dir to mark-dir
                        end if
                    else if findloop is true
```

```
                        set mark2 to cur
                        set mark2-dir to cur-dir
                    end if
                else
                    if cur is at mark
                        set cur to mark2
                        set cur-dir to mark2-dir
                        clear mark and mark2
                        set backtrack to false
                        turn around
                        fill cur
                        jump to PAINT
                    else if cur at mark2
                        set mark to cur
                        set cur-dir and mark-dir to mark2-dir
                        clear mark2
                    end
                end if
            end if
        end case
        case 3
            clear mark
            fill cur
            jump to PAINT
        end case
        case 4
            fill cur
            done
        end case
    end switch
end MAIN LOOP
```

4. Thuật toán xén hình Cohen – Sutherland



Hình 4-7 Điểm và đoạn thẳng bị cắt khỏi cửa sổ

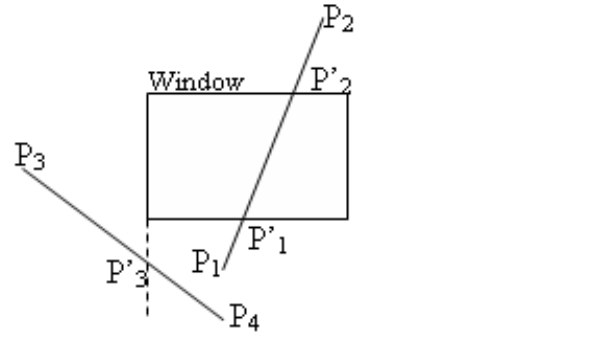
- Khái niệm Mã vùng (Area Code): một con số 4 bit nhị phân sẽ được gán cho mỗi vùng để mô tả vị trí tương đối của nó so với cửa sổ. Bằng cách đánh số từ 1 tới 4 theo thứ tự từ phải qua trái, ta quy ước vị trí của vùng so với cửa sổ (Windows) như sau:
  - Bit 1: trái (LEFT).
  - Bit 2: Phải (Right).
  - Bit 3: trên (Top).
  - Bit 4: dưới (Bottom).

Hình 4-8
Các mã vùng nhị phân cho các điểm đầu mút đoạn thẳng, được dùng để định nghĩa các vùng tọa độ liên hệ với một cửa sổ.

1001	1000	1010
0001	0000 Window	0010
0101	0100	0110

- Giá trị 1 tương ứng với vị trí bit nào trong mã vùng thì nó sẽ chỉ ra rằng điểm đó ở vị trí tương ứng, ngược lại bit đó được đặt bằng 0. Điểm nằm hoàn toàn trong cửa sổ sẽ có vị trí 0000.

Hình 4-9
Các đoạn từ một điểm này đến một điểm khác có thể cắt cửa sổ hoặc giao điểm với các biên nằm ngoài cửa sổ.



- Các bước của thuật toán:
    - Bước 1: Gán mã vùng cho điểm đầu điểm cuối P_1, P_2 của đường thẳng là c1, c2.
    - Bước 2: Các đoạn thẳng nằm hoàn toàn trong cửa sổ sẽ có c1 = c2 = 0000.
    - Bước 3: Nếu tồn tại k {1,…4} sao cho ứng với bit thứ k của c1, c2 đều có giá trị 1. Lúc này, đoạn thẳng nằm cùng phía so với bit k của cửa sổ, do đó nằm ngoài cửa sổ, Đoạn thẳng này sẽ bị loại bỏ sau khi xén.
    - Nếu c1, c2 không thuộc 2 trường hợp ở Bước 2 và Bước 3 thì đoạn thẳng có thể: không cắt cửa sổ, hoặc tồn tại một điểm nằm ngoài cửa sổ. Bằng cách xét mã vùng, ta có thể xác định được biên nào cắt nó để từ đó tìm giao điểm của đoạn thẳng đó với biên. Ví dụ trong hình trên (4-9), đoạn bị xén là P_2P_2' và P_1P_1' (với P_1' là giao điểm của P_1P_2 với biên dưới và P_2' là giao điểm P_1P_2 với biên trên).

Thuật toán Cohen – Sutherland (Mã C):

```
typedef int OutCode;

const int INSIDE = 0; // 0000
const int LEFT = 1;   // 0001
const int RIGHT = 2;  // 0010
const int BOTTOM = 4; // 0100
const int TOP = 8;    // 1000

// Compute the bit code for a point (x, y) using the clip rectangle
// bounded diagonally by (xmin, ymin), and (xmax, ymax)

// ASSUME THAT xmax, xmin, ymax and ymin are global constants.

OutCode ComputeOutCode(double x, double y)
{
                OutCode code;

                code = INSIDE;          // initialised as being inside of [[clip window]]

                if (x < xmin)           // to the left of clip window
                        code |= LEFT;
                else if (x > xmax)      // to the right of clip window
                        code |= RIGHT;
                if (y < ymin)           // below the clip window
                        code |= BOTTOM;
                else if (y > ymax)      // above the clip window
                        code |= TOP;

                return code;
}

// Cohen–Sutherland clipping algorithm clips a line from
// P0 = (x0, y0) to P1 = (x1, y1) against a rectangle with
// diagonal from (xmin, ymin) to (xmax, ymax).
```

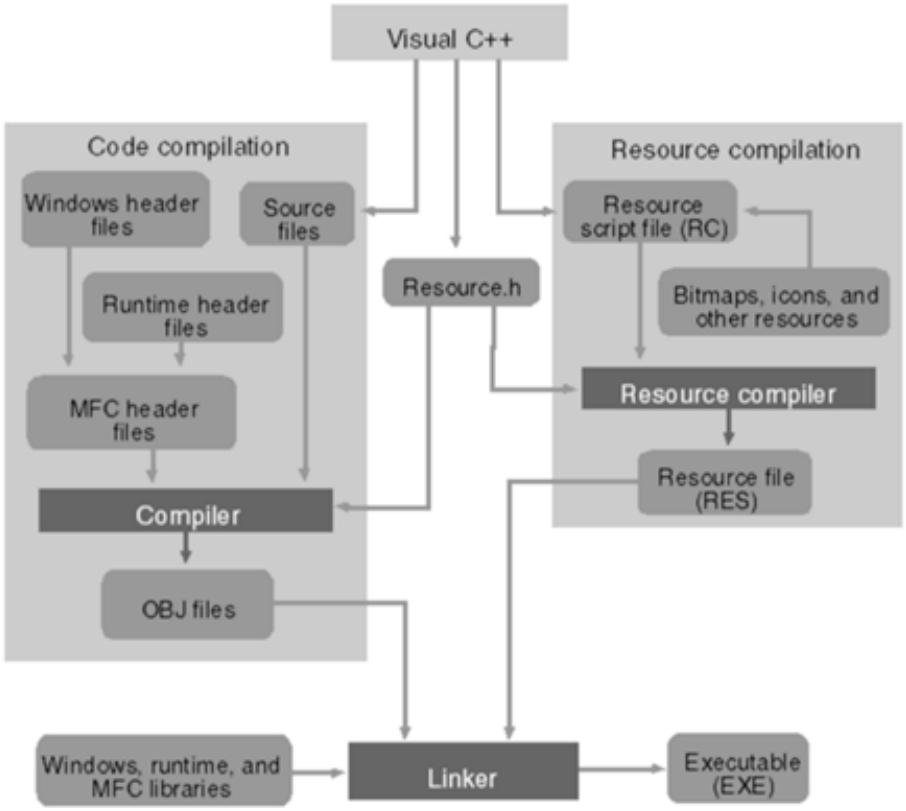
```
void CohenSutherlandLineClipAndDraw(double x0, double y0, double x1, double y1)
{
		// compute outcodes for P0, P1, and whatever point lies outside the clip rectangle
		OutCode outcode0 = ComputeOutCode(x0, y0);
		OutCode outcode1 = ComputeOutCode(x1, y1);
		bool accept = false;

		while (true) {
		if (!(outcode0 | outcode1)) { // Bitwise OR is 0. Trivially accept and get out of loop
				accept = true;
				break;
		} else if (outcode0 & outcode1) { // Bitwise AND is not 0. Trivially reject and get out of loop
				break;
			} else {
				// failed both tests, so calculate the line segment to clip
				// from an outside point to an intersection with clip edge
				double x, y;

				// At least one endpoint is outside the clip rectangle; pick it.
				OutCode outcodeOut = outcode0 ? outcode0 : outcode1;

				// Now find the intersection point;
				// use formulas y = y0 + slope * (x - x0), x = x0 + (1 / slope) * (y - y0)
				if (outcodeOut & TOP) {           // point is above the clip rectangle
					x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
					y = ymax;
				} else if (outcodeOut & BOTTOM) { // point is below the clip rectangle
					x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
					y = ymin;
				} else if (outcodeOut & RIGHT) {  // point is to the right of clip rectangle
					y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
					x = xmax;
				} else if (outcodeOut & LEFT) {   // point is to the left of clip rectangle
					y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
					x = xmin;
				}

				// Now we move outside point to intersection point to clip
				// and get ready for next pass.
				if (outcodeOut == outcode0) {
					x0 = x;
					y0 = y;
					outcode0 = ComputeOutCode(x0, y0);
				} else {
					x1 = x;
					y1 = y;
					outcode1 = ComputeOutCode(x1, y1);
				}
			}
		}
```

```
                if (accept) {
            // Following functions are left for implementation by user based on
            // their platform (OpenGL/graphics.h etc.)
            DrawRectangle(xmin, ymin, xmax, ymax);
            LineSegment(x0, y0, x1, y1);
                }
}
```

CHƯƠNG 4: CẤU TRÚC CHƯƠNG TRÌNH

1. Tổng quan về Lập trình MFC

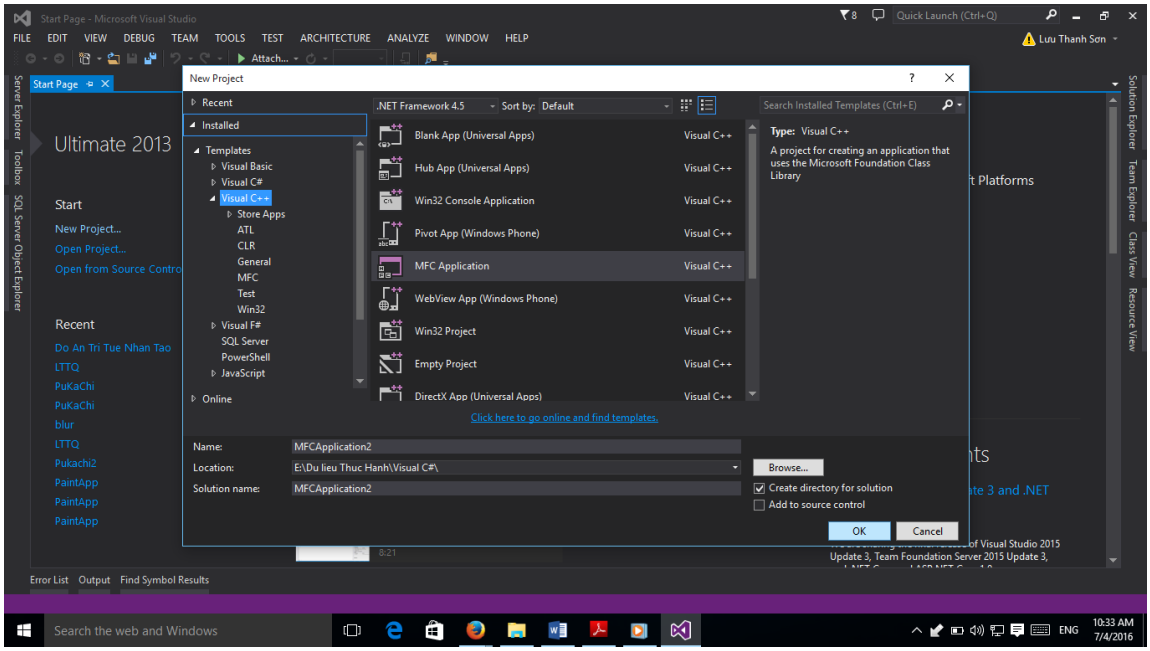
- Là một thư viện các lớp (class, OOP) trong ngôn ngữ Visual C++, dùng cho việc lập trình trên Windows. Được xây dựng trên cơ sở các hàm thư viện API của Windows.
- Giúp cho người lập trình có thể xây dựng ứng dụng nhanh và ít tốn công sức hơn so với việc sử dụng đơn thuần các hàm thư viện API của Windows
- Ta vẫn có thể gọi các hàm Windows API trong MFC
- Trong 1 ứng dụng MFC, ta thường không gọi hàm Windows API trực tiếp, mà sẽ tạo các object từ những lớp của MFC, và gọi phương thức của object đó
- Đa số các phương thức của MFC class có cùng tên với những hàm Windows API
- MFC tạo ra một Application Framework, giúp:
    - Thiết lập kiến trúc của ứng dụng một cách nhất quán và khoa học
    - Che dấu đi nhiều phần chi tiết mà Windows API đòi hỏi, giúp developer "thảnh thơi" hơn
- Các hàm thư viện API của Windows thường chứa trong file Header **stdafx.h**.



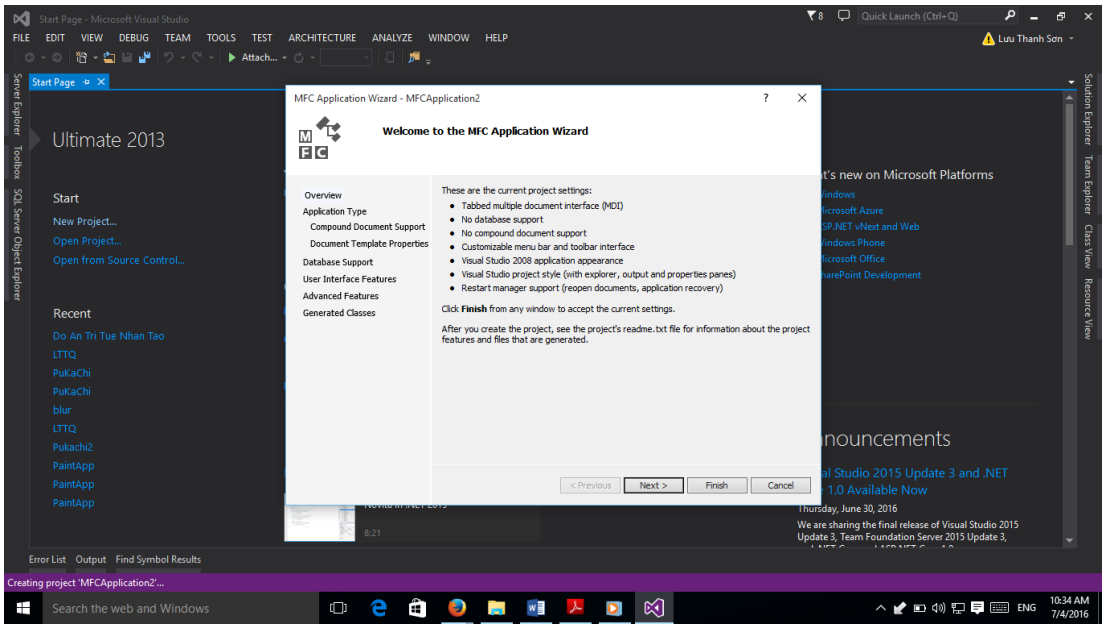
Sơ đồ biên dịch các thành phần của một ứng dụng trong VC++

- Kiến trúc Document – View: Gồm 3 thành phần: Document – Frame – View. Trong đó:
    - Document: Quản lý toàn bộ nội dung dữ liệu của ứng dụng được lưu trong bộ nhớ.
    - View: Thực hiện chức năng hiển thị.
    - Frame: Chứa view, điều phối command message từ người dùng đến các view một cách thích hợp. Các view ở đây có thể là: Toolbar, Button, Menu Items,…

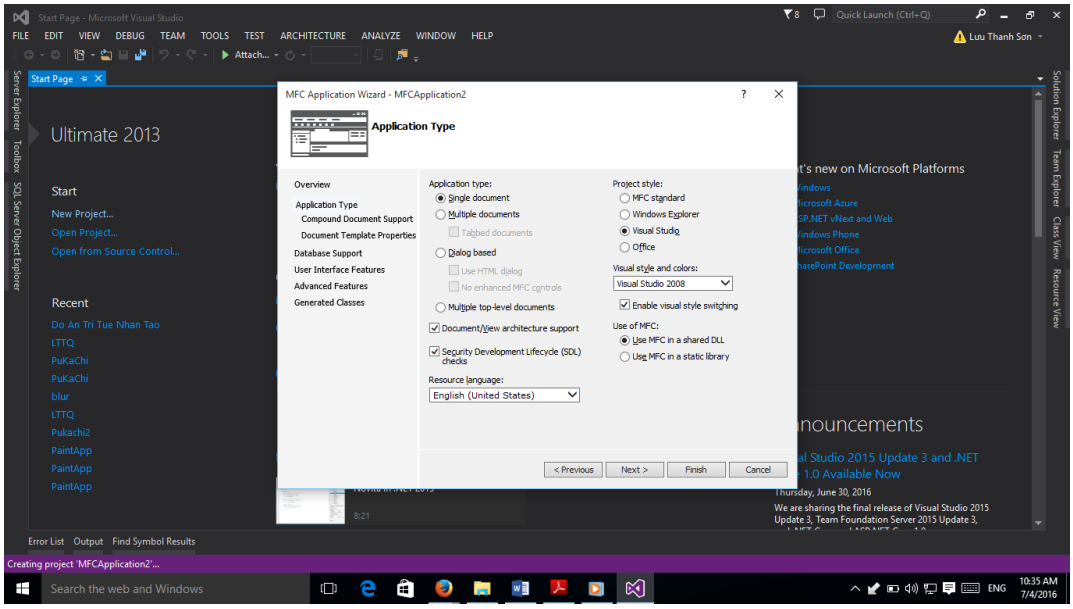
- Các lớp liên quan đến bộ khung này:
  - *CDocument*: Là lớp đối tượng quản lý mội nội dung Dữ liệu.
  - *CView*: Lớp kế thừa từ lớp CWnd, dùng để quản lý và điều phối các thành phần View của ứng dụng.
  - *CFrameWnd*: Lớp đối tượng quản lý khung cửa sổ chính của Ứng dụng.
  - *CDoc*: Lớp đối tượng quản lý tất cả bộ ba Document – Frame – View của MFC.
- Tạo một Project MFC:
  - **Bước 1**: Chọn File – New – Project – Chọn MFC Application. Chọn OK.



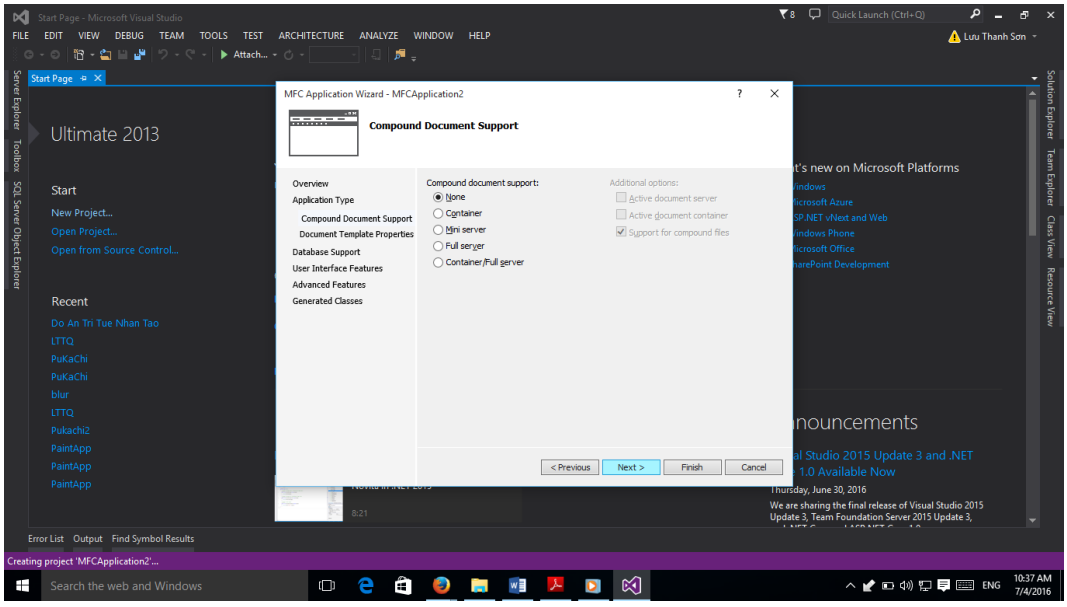
  - **Bước 2**: Chọn Next. Sau đó Chọn Single Document (là tại liệu đơn cửa sổ) hoặc Multiple Document (tài liệu đa cửa sổ).



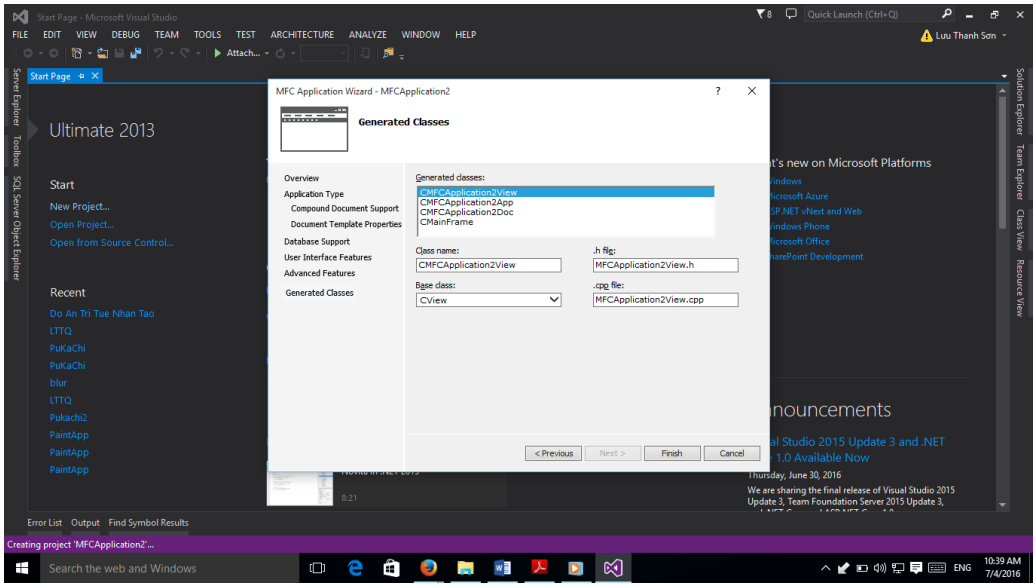
Ví dụ về Single Document là chương trình Paint, chỉ cho ta mở duy nhất 1 cửa sổ trong phiên làm việc, trong khi đó, Visual Studio là loại tài liệu Đa cửa sổ, ta có thể mở nhiều tab khác nhau trong 1 phiên làm việc.



- **Bước 3**: Chọn None. Sau đó chọn Next.

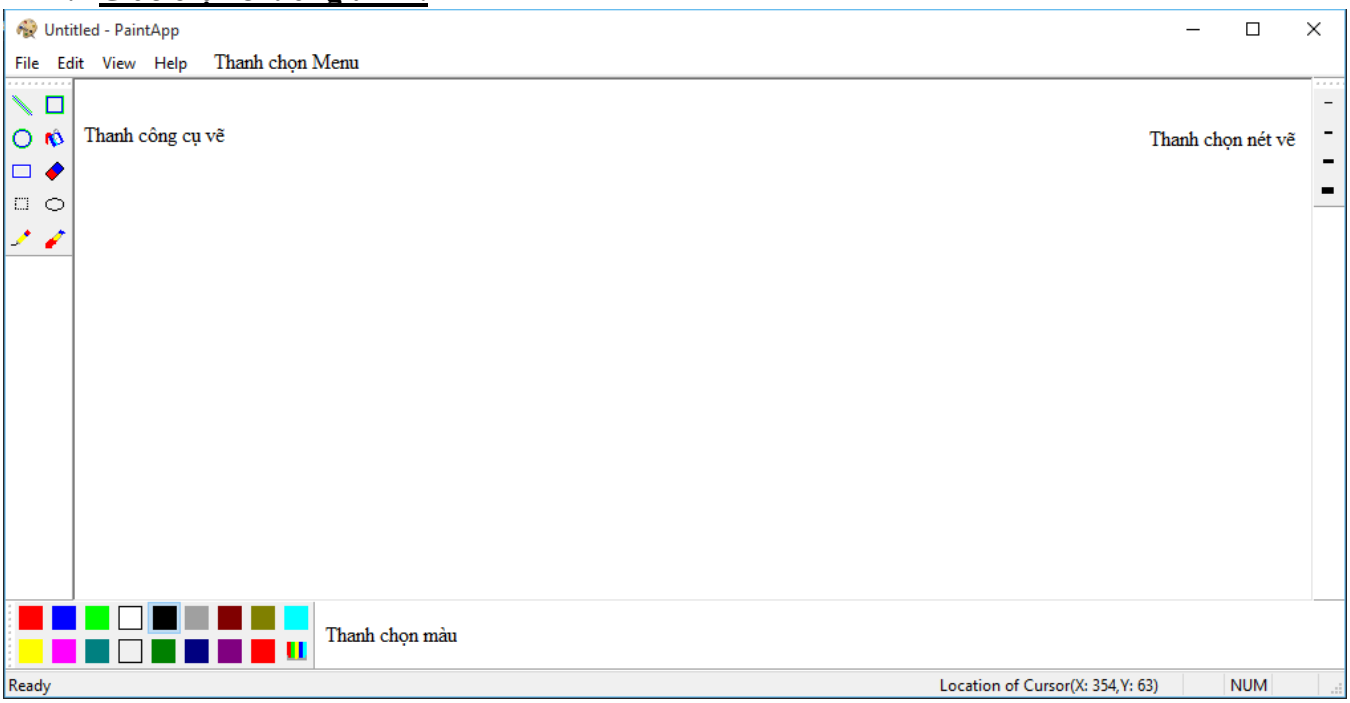


- **Bước 4:** Chọn Next 3 lần cho tới khi hộp thoại sau xuất hiện. Đặt tên cho các lớp theo ý muốn (hoặc có thể để mặc định).



- **Bước 5:** Chọn Finish để khởi tạo Project MFC.

2. Giao diện chương trình.



- Thanh công cụ vẽ: Gồm các công cụ: Vẽ đường thẳng, vẽ hình vuông, vẽ hình tròn, Tô màu, vẽ hình chữ nhật, cục tẩy, Chọn đối tượng(Select), vẽ Elip, bút chì, cọ.
- Thanh chọn màu: Gồm các màu thông dụng có sẵn và nút chọn bảng màu ở cuối cùng.
- Thanh chọn nét vẽ: Gồm 4 loại nét vẽ từ nhạt tới đậm.
- Thanh Menu: File (Save, Open, Exit), Edit (Cut, Copy, Erase All, Undo), View (Color Bar, PenStyle Bar), Help (About..).

3. Cấu trúc chương trình

a. Lớp C_Circle:

- Là lớp dùng cho việc vẽ đường tròn theo thuật toán MidPoint.

Khai báo lớp:

```
#pragma once
#ifndef _CIRCLE_H_
#define _CIRCLE_H_
class C_Circle
{
public:
	C_Circle();
	~C_Circle();
	void DrawCircle(CDC *pDC, CPoint m_point_s, float iR, COLORREF color);
	void Set8Pixel(CDC *pDC, CPoint m_point, int a, int b, COLORREF color);
};

#endif
```

Các hàm chính:

```
//hàm vẽ đường tròn
void C_Circle::DrawCircle(CDC *pDC, CPoint m_point_s, float iR, COLORREF color)
//MidPoint Algorithm
{
	int x = 0, y = iR;
	int d = 1 - iR;
	Set8Pixel(pDC, m_point_s, x, y, color);

	while (x<y)
	{
		if (d<0)
			d += 2 * x + 3;
		else
		{
			d += 2 * (x - y) + 5;
			y--;
		}
		x++;
		Set8Pixel(pDC, m_point_s, x, y, color);
	}
}
//hàm vẽ 8 pixel đối xứng (mỗi pixel tương ứng với một cung tròn do tuân theo thuật toán MidPoint chỉ vẽ 1/8 đường tròn)
void C_Circle::Set8Pixel(CDC *pDC, CPoint m_point, int a, int b, COLORREF color)
{
	pDC->SetPixel(m_point.x + a, m_point.y + b, color);
	pDC->SetPixel(m_point.x - a, m_point.y + b, color);
	pDC->SetPixel(m_point.x - a, m_point.y - b, color);
```

```
        pDC->SetPixel(m_point.x + a, m_point.y - b, color);
        pDC->SetPixel(m_point.x + b, m_point.y + a, color);
        pDC->SetPixel(m_point.x - b, m_point.y + a, color);
        pDC->SetPixel(m_point.x - b, m_point.y - a, color);
        pDC->SetPixel(m_point.x + b, m_point.y - a, color);
}
```

b. Lớp C_Elip:

- Là lớp dùng cho việc vẽ đường Elip theo thuật toán MidPoint

Khai báo lớp:

```
#pragma once
#ifndef _CIRCLE_H_
#define _CIRCLE_H_
class C_Circle
{
public:
        C_Circle();
        ~C_Circle();
        void DrawCircle(CDC *pDC, CPoint m_point_s, float iR, COLORREF color);
        void Set8Pixel(CDC *pDC, CPoint m_point, int a, int b, COLORREF color);
};

#endif
```

Các hàm chính:

```
//hàm vẽ Elip
void C_Elip::DrawElip(CDC *pDC, CPoint point_s, CPoint point_f, COLORREF color)
//MidPoint Algorithm
{
        int a, b;
        a = abs((point_f.x - point_s.x)/2);
        b = abs((point_f.y - point_s.y)/2);

        CPoint midPoint;
        midPoint.x = (int)(point_s.x + point_f.x) / 2;
        midPoint.y = (int)(point_s.y + point_f.y) / 2;

        int x, y;
        double k1, k2, P;
        x = 0;
        y = b;
        k1 = (double)(b*b) / (a*a);
        k2 = (double)1 / k1;
        P = 2 * k1 - (2 * b) + 1;
        while (k1*(double)x / y <= 1)
        {
                setpixel6(pDC, midPoint.x, midPoint.y, x, y, color);
```

```
                if (P < 0) P += 2 * k1*(2 * x + 3);
                else
                {
                        P += 2 * k1*(2 * x + 3) + 4 * (1 - y);
                        y--;
                }
                x++;
        }

        x = a;
        y = 0;
        P = 2 * k2 - 2 * a + 1;
        while (k2*(double)y / x <= 1)
        {
                setpixel6(pDC, midPoint.x, midPoint.y, x, y, color);
                if (P < 0) P += 2 * k2*(2 * y + 3);
                else
                {
                        P += 2 * k2*(2 * y + 3) + 4 * (1 - x);
                        x--;
                }
                y++;
        }



}
//lấy 6 điểm đối xứng.
void C_Elip::setpixel6(CDC *pDC, int xc, int yc, int x, int y, int color)
{
        pDC->SetPixel(xc + x, y + yc, color);
        pDC->SetPixel(xc - x, y + yc, color);
        pDC->SetPixel(xc - x, yc - y, color);
        pDC->SetPixel(xc + x, yc - y, color);
}
```

c. Lớp C_FillColor:

- Là lớp thực hiện việc tô màu cho đối tượng dựa theo thuật toán tô màu Boundary Fill (có cải tiến)

Khai báo lớp:

```
#pragma once
struct TSeed
{
        int x;
        int y;
}; //lấy tọa độ của 1 điểm lân cận
struct TStore
```

```
{
	int n;
	TSeed seed[1024];
}; //Tập tọa độ các điểm lân cận
class C_FillColor
{
public:
	C_FillColor();
	~C_FillColor();
	void BoundaryFill(CDC *pDC, int x, int y, COLORREF color, COLORREF cursor_color);
};
```

Hàm thực hiện:

```
//thực hiện thuật toán tô màu Boundary Fill
void C_FillColor::BoundaryFill(CDC *pDC, int x, int y, COLORREF color, COLORREF cursor_color)
{
	stack <CPoint> t1;
	if (pDC->GetPixel(x, y) != cursor_color)
		return;
	t1.push(CPoint(x, y));
	pDC->SetPixel(x, y, color);
	while (!t1.empty())
	{
		x = t1.top().x;
		y = t1.top().y;
		t1.pop();
		if ((pDC->GetPixel(x + 1, y) == cursor_color))
		{
			pDC->SetPixel(x + 1, y, color);
			t1.push(CPoint(x + 1, y));
		}
		//hướng bên trái
		if ((pDC->GetPixel(x - 1, y) == cursor_color))
		{
			pDC->SetPixel(x - 1, y, color);
			t1.push(CPoint(x - 1, y));
		}
		//hướng trên
		if ((pDC->GetPixel(x, y + 1) == cursor_color))
		{
			pDC->SetPixel(x, y + 1, color);
			t1.push(CPoint(x, y + 1));
		}
		//hướng dưới
		if ((pDC->GetPixel(x, y - 1) == cursor_color))
```

```
                {
                        pDC->SetPixel(x, y - 1, color);
                        t1.push(CPoint(x, y - 1));
                }
        }
}
void Push(TStore &store, int x, int y)
{
        store.seed[store.n].x = x;
        store.seed[store.n].y = y;
        store.n++;
}

void Pop(TStore &store, int &x, int &y)
{
        store.n--;
        x = store.seed[store.n].x;
        y = store.seed[store.n].y;
}

int IsEmpty(TStore &store)
{
        return (store.n == 0);
}
```

d. Lớp C_Line

- Là lớp thực hiện việc vẽ đường thẳng theo thuật toán Bresenham.

Khai báo lớp:

```
#pragma once
#define Round(a) int(a+0.5)
class C_Line
{
public:
        C_Line();
        ~C_Line();
        void DrawLine(CDC *pDC, CPoint m_point_s, CPoint m_point_f,COLORREF cCurrentColor);
};
```

Các hàm thực hiện:

```
void C_Line::DrawLine(CDC *pDC, CPoint m_point_s, CPoint m_point_f, COLORREF
cCurrentColor)  //Bresenham Algorithm
{
	int dy = abs(m_point_f.y - m_point_s.y);
	int dx = abs(m_point_f.x - m_point_s.x);
	int x, y, p;
	int c;
	int f_x, f_y;

	p = dx - dy;

	x = m_point_s.x;
	y = m_point_s.y;

	f_x = 1;
	f_y = 1;

	if (m_point_f.x - m_point_s.x < 0)
		f_x = -f_x;
	if (m_point_f.y - m_point_s.y < 0)
		f_y = -f_y;

	pDC->SetPixel(x, y, cCurrentColor);
	if (m_point_s.x == m_point_f.x)
	{
		for (int i = m_point_s.y; i < m_point_f.y; i++)
		{
			y += f_y;
			pDC->SetPixel(x, y, cCurrentColor);
		}
	}
	else
		if (m_point_s.y == m_point_f.y)
		{
			for (int i = m_point_s.x; i < m_point_f.x; i++)
			{
				x += f_x;
				pDC->SetPixel(x, y, cCurrentColor);
			}
		}

	if (m_point_s.x != m_point_f.x && m_point_s.y != m_point_f.y)
	{
		while (1)
		{
			if (x == m_point_f.x && y == m_point_f.y)
```

```
                        break;
                c = 2 * p;
                if (c > -dy)
                {
                        p = p - dy;
                        x = x + f_x;
                }
                if (c < dx)
                {
                        p = p + dx;
                        y = y + f_y;
                }
                pDC->SetPixel(x, y, cCurrentColor);
        }
    }
}
```

e. Lớp Rectangle:

- Lớp thực hiện vẽ hình chữ nhật (thật sự là vẽ 4 đường thẳng AB, BC, CD, DA) nên bản chất của nó cũng là thuật toán vẽ đường thẳng.

Khai báo lớp:

```
#pragma once
class C_Regtangle
{
public:
        C_Regtangle();
        ~C_Regtangle();
        void DrawRectangle(CDC *pDC, CPoint m_point_s, CPoint m_point_f, COLORREF color);
};
```

Các hàm thực hiện:

```
void C_Regtangle::DrawRectangle(CDC *pDC, CPoint m_point_s, CPoint m_point_f ,COLORREF color)
{
        C_Line *p1 = new C_Line();
        //vẽ 4 đường thẳng.
        p1->DrawLine(pDC, CPoint(m_point_s.x, m_point_s.y), CPoint(m_point_f.x, m_point_s.y), color);
        p1->DrawLine(pDC, CPoint(m_point_s.x, m_point_s.y), CPoint(m_point_s.x, m_point_f.y), color);
        p1->DrawLine(pDC, CPoint(m_point_s.x, m_point_f.y), CPoint(m_point_f.x, m_point_f.y), color);
        p1->DrawLine(pDC, CPoint(m_point_f.x, m_point_s.y), CPoint(m_point_f.x, m_point_f.y), color);
        delete p1;
}
```

f. Lớp Square:

- Thực hiện việc vẽ hình vuông. Thực hiện tương tự như vẽ hình chữ nhật.

Khái báo lớp:

```
#pragma once
class C_Square
{
public:
	C_Square();
	~C_Square();
	void Draw_Square(CDC *pDC, CPoint m_point_s, CPoint m_point_f, COLORREF color);
};
```

Các hàm thực hiện:

```
void C_Square::Draw_Square(CDC *pDC, CPoint m_point_s, CPoint m_point_f, COLORREF color)
{
	C_Regtangle *p1 = new C_Regtangle;
	CPoint m_point_bot;
	if (m_point_s.x < m_point_f.x)
	{
	//thực hiện giới hạn tọa độ 4 điểm vẽ sao cho khoảng cách của chúng là bằng nhau (hình vuông có 4 cạnh bằng nhau)
		if (m_point_s.y < m_point_f.y)
		{
			m_point_bot =CPoint(m_point_f.x, m_point_f.x - m_point_s.x + m_point_s.y);
			p1->DrawRectangle(pDC, m_point_s, m_point_bot, color);
		}
		else
		{
			CPoint m_point_bot(m_point_f.x, m_point_s.x - m_point_f.x + m_point_s.y);
			p1->DrawRectangle(pDC, m_point_s, m_point_bot, color);
		}
	}
	else
	{
		if (m_point_s.y < m_point_f.y)
		{
			CPoint m_point_bot(m_point_f.x, m_point_s.x - m_point_f.x + m_point_s.y);
			p1->DrawRectangle(pDC, m_point_s, m_point_bot, color);
		}
		else
		{
```

```
                        m_point_bot = CPoint(m_point_f.x, m_point_f.x - m_point_s.x + m_point_s.y);
                        p1->DrawRectangle(pDC, m_point_s, m_point_bot, color);
                }
        }
        delete p1;
}
```

g. Lớp Screen:

- Là lớp dùng để lấy tập hợp các Pixel trên màn hình để phục vụ cho công việc Undo. Tư tưởng của Undo là: chúng ta sẽ lưu lại trạng thái màn hình hiện tại trước khi vẽ thêm một đối tượng mới vào. Khi cần Undo, chúng ta chỉ việc gọi trạng thái trước của màn hình.

Khai báo lớp:

```
#pragma once
#include"stdafx.h"
class Screen
{
        COLORREF **scr;  //mỗi phần tử trên màn hình là 1 pixel với các giá trị: tọa độ x,y và màu sắc của nó.
public:
        Screen();
        void getScreen(CDC *pDC); //lấy trạng thái màn hình
        void setScreen(CDC *pDC); //đặt trạng thái cho màn hình
        void release(); //giải phóng bộ nhớ
};
```

Các hàm thực hiện:

```
void Screen::getScreen(CDC *pDC)
{
        for (int i = 0; i < width; i++)
                for (int j = 0; j < height; j++)
                        scr[i][j] = pDC->GetPixel(i, j);
}
void Screen::setScreen(CDC *pDC)
{
        for (int i = 0; i < width; i++)
                for (int j = 0; j < height; j++)
                        pDC->SetPixel(i, j, scr[i][j]);
}
void Screen::release()
{
        for (int i = 0; i < width; i++)
                        delete[] scr[i];
        delete[]scr;
}
```

e. Lớp CpaintAppView:

- Đây là lớp dùng để quản lý và kết nối các thành phần View của ứng dụng (button, Menu Items, Tollbar,…) với Resource.
- Các hàm tham chiếu:

```
ON_COMMAND(ID_EDIT_UNDO, &CPaintAppView::OnUndo)  //Menu Edit – Undo.
ON_COMMAND(ID_EDIT_CUT, &CPaintAppView::OnMove)    //Menu Edit – Move.
ON_COMMAND(ID_EDIT_COPY, &CPaintAppView::OnCopy)  //Menu Edit – Copy.
ON_COMMAND(ID_EDIT_CLEAR_ALL, &CPaintAppView::OnEraseAll)  //Menu Edit – Clear
//đường thẳng
ON_COMMAND(ID_BTN_LINE, &CPaintAppView::OnLine)  //nút vẽ đường thẳng
ON_UPDATE_COMMAND_UI(ID_BTN_LINE,  &CPaintAppView::OnLineUpdate)    //giữ trạng thái vẽ đường thẳng cho nút. Các hàm dành cho các nút khác tương tự

ON_COMMAND(ID_BTN_ERASE, &CPaintAppView::OnErase)  //dùng cục tẩy (Erase)
ON_UPDATE_COMMAND_UI(ID_BTN_ERASE, &CPaintAppView::OnEraseUpdate)

ON_COMMAND(ID_BTN_SQUARE, &CPaintAppView::OnSquare)  //vẽ hình vuông
ON_UPDATE_COMMAND_UI(ID_BTN_SQUARE, &CPaintAppView::OnSquareUpdate)
```

```
ON_COMMAND(ID_BTN_ELIP, &CPaintAppView::OnElip)   //vẽ Elip
ON_UPDATE_COMMAND_UI(ID_BTN_ELIP, &CPaintAppView::OnElipUpdate)

ON_COMMAND(ID_BTN_CIRCLE, &CPaintAppView::OnCircle)  //vẽ hình tròn
ON_UPDATE_COMMAND_UI(ID_BTN_CIRCLE, &CPaintAppView::OnCircleUpdate)

ON_COMMAND(ID_BTN_REGTANGLE, &CPaintAppView::OnRectangle)   //vẽ hình chữ nhật
ON_UPDATE_COMMAND_UI(ID_BTN_REGTANGLE,
&CPaintAppView::OnRectangleUpdate)

ON_COMMAND(ID_BTN_FILL_COLOR, &CPaintAppView::OnFillColor)  //tô màu
ON_UPDATE_COMMAND_UI(ID_BTN_FILL_COLOR,
&CPaintAppView::OnFillColorUpdate)

ON_COMMAND(ID_BTN_PENCIL, &CPaintAppView::OnPencil)  //dùng bút chì
ON_UPDATE_COMMAND_UI(ID_BTN_PENCIL, &CPaintAppView::OnPencilUpdate)

ON_COMMAND(ID_BTN_BRUSH, &CPaintAppView::OnBrush)  //dùng cọ vẽ
ON_UPDATE_COMMAND_UI(ID_BTN_BRUSH, &CPaintAppView::OnBrushUpdate)

…và các hàm dùng cho các Button còn lại tương tự
```

- Cấu trúc lớp PainAppView.h

```
//Các thiết bị đồ họa GDI
        CRect m_rect;
        CDC m_BG;
        CBitmap m_bmpBG;
        CDC m_VDC;
        CBitmap m_bmpVDC;
        void OnInitialUpdate();

//vị trí con trỏ chuột ở các sự kiện click và màu, độ đậm, chế độ vẽ  tại thời điểm hiện tại
        CPoint m_point_s;    //vị trí chuột ban đầu
        CPoint m_point_f;  //vị trí chuột lúc sau (sau khi di chuyển)
        CPoint m_point_mid;

        COLORREF cCurrentColor;  //biến cho biết màu hiện tại đang được chọn
        DRAWMODE dType;  //biến cho biết loại công cụ đang được sử dụng hiện tại (giá trị được qui định trong File Definition.h)

//dùng cho công cụ Undo: Là một tập vector các trạng thái của màn hình
        vector<Screen>vtAction;

//Chọn đối tượng: Là một ma trận XxX với chiều dài và chiều rộng thay đổi tùy theo cách người dùng chọn
        COLORREF **selectArea;
        int select_row, select_column;
        CPoint temp_s, temp_f;

        //file name
        CString FileName;
        HBITMAP hBitmap;

//vị trí con trỏ chuột
        CPoint CursorLocation;

//dùng cho cọ vẽ
        int brushStyles = 4;  //loại cọ vẽ ban đầu (đậm nhạt)
        int tag = 1;

        CPoint oldPoint = (0, 0);

        CDC *DC_Copy;
        CRect m_rect_value;
```

- Các hàm khai báo phục vụ cho Chương trình: Các hàm này đã được Hàm ở trên khai báo gán vào một đối tượng View cụ thể và khi đối tượng View đó phát sinh hành **ON_COMMAND** động (Click, Drag, Move, …) thì các hàm này sẽ được gọi ứng với từng đối tượng.

```
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
afx_msg void OnMouseMove(UINT nFlags, CPoint point);
afx_msg void OnUpdateIdrIndicatorPos(CCmdUI *pCmdUI);
afx_msg void OnUpdateIndicatorPos(CCmdUI *pCmdUI);

afx_msg void OnLine();   //Duong Thang
afx_msg void OnLineUpdate(CCmdUI *pCmdUI);  //

afx_msg void OnSquare();  //Square
afx_msg void OnSquareUpdate(CCmdUI *pCmdUI);

afx_msg void OnCircle();  //Circle
afx_msg void OnCircleUpdate(CCmdUI *pCmdUI);

afx_msg void OnRectangle();  //Rectangle
afx_msg void OnRectangleUpdate(CCmdUI *pCmdUI);

afx_msg void OnElip();
afx_msg void OnElipUpdate(CCmdUI *pCmdUI);

afx_msg void OnFillColor();   //Fill Color
afx_msg void OnFillColorUpdate(CCmdUI *pCmdUI);

afx_msg void OnDialogColor();

afx_msg void OnErase();   //Erase
afx_msg void OnEraseUpdate(CCmdUI *pCmdUI);

afx_msg void OnEraseAll();   //Erase All

//Colors
afx_msg void OnRed();
afx_msg void OnBlue();
afx_msg void OnGreen();
afx_msg void OnBtn0255255();
afx_msg void OnBtn0128128();
afx_msg void OnBtn12800();
afx_msg void OnBtn1281280();
afx_msg void OnBlack();
afx_msg void OnGray();
afx_msg void OnPink();
afx_msg void OnWhite();
afx_msg void OnYellow();
afx_msg void OnOrange();
afx_msg void OnOrangeUpdate(CCmdUI *pCmdUI);
afx_msg void OnPinkUpdate(CCmdUI *pCmdUI);
```

```
afx_msg void OnRedUpdate(CCmdUI *pCmdUI);
afx_msg void OnGreenUpdate(CCmdUI *pCmdUI);
afx_msg void OnBtn0128128Update(CCmdUI *pCmdUI);
afx_msg void OnBtn0255255Update(CCmdUI *pCmdUI);
afx_msg void OnBtn12800Update(CCmdUI *pCmdUI);
afx_msg void OnBtn1281280Update(CCmdUI *pCmdUI);
afx_msg void OnBlackUpdate(CCmdUI *pCmdUI);
afx_msg void OnBlueUpdate(CCmdUI *pCmdUI);
afx_msg void OnGrayUpdate(CCmdUI *pCmdUI);
afx_msg void OnWhiteUpdate(CCmdUI *pCmdUI);
afx_msg void OnYellowUpdate(CCmdUI *pCmdUI);
afx_msg void OnBtn1280128();
afx_msg void OnBtn1280128Update(CCmdUI *pCmdUI);
afx_msg void OnBtn128128128();
afx_msg void OnBtn128128128Update(CCmdUI *pCmdUI);
afx_msg void OnBtn00128();
afx_msg void OnBtn00128Update(CCmdUI *pCmdUI);
afx_msg void OnBtn01280();
afx_msg void OnBtn01280Update(CCmdUI *pCmdUI);

afx_msg void OnSelectUpdate(CCmdUI *pCmdUI);  //Select Object

afx_msg void OnMove();  //Di chuyen
afx_msg void OnCopy();

afx_msg void OnSelect();     //Chon

afx_msg void OnPencil();    //but ve
afx_msg void OnPencilUpdate(CCmdUI *pCmdUI);

afx_msg void OnBrush();    //co ve
afx_msg void OnBrushUpdate(CCmdUI *pCmdUI);

afx_msg void OnPencilSmallSize();    //net ve nho
afx_msg void OnPencilSmallSizeUpdate(CCmdUI *pCmdUI);

afx_msg void OnPencilMediumSize();    //net ve vua
afx_msg void OnPencilMediumSizeUpdate(CCmdUI *pCmdUI);

afx_msg void OnPencilLargeSize();    //net ve dam
afx_msg void OnPencilLargeSizeUpdate(CCmdUI *pCmdUI);

afx_msg void OnPencilVeryLargeSize();    //net rat ve dam
afx_msg void OnPencilVeryLargeSizeUpdate(CCmdUI *pCmdUI);

afx_msg void OnFileSave();      //Save File
afx_msg void OnFileOpen();  //Open File
```

```
afx_msg void OnUndo();       //Undo

afx_msg void OnDestroy();
```

- Các hàm xử lý sự kiện khi vẽ:

Khi ta chọn một button bất kỳ thì nó sẽ gán dType (là một kiểu ENUM) bằng một giá trị cụ thể

```
void CPaintAppView::OnFillColor()
{
    dType = FILL_COLOR;
}
void CPaintAppView::OnFillColorUpdate(CCmdUI *pCmdUI)
{
    if (dType == FILL_COLOR)
            pCmdUI->SetCheck(TRUE);
    else
            pCmdUI->SetCheck(FALSE);
}
void CPaintAppView::OnPencil()
{
    addAction();  //lưu trạng thái màn hình hiện tại trước khi thực hiện (hỗ trợ iệc undo)
    dType = PENCIL;
}
void CPaintAppView::OnPencilUpdate(CCmdUI *pCmdUI)
{
    if (dType == PENCIL)
            pCmdUI->SetCheck(TRUE);
    else
            pCmdUI->SetCheck(FALSE);
}
//trên chỉ lấy ví dụ cho 2 công cụ là PENCIL và FILL COLOR. Các công cụ còn lại tương tự
```

- Xử lý sự kiện chuột:

Khi phím trái chuột nhấn xuống:

```
void CPaintAppView::OnLButtonDown(UINT nFlags, CPoint point)
{

	m_point_s = m_point_f = point;  //lấy tọa độ chuột hiên tại khi nút trái chuột nhấn xuống.
	if (dType == FILL_COLOR) //Tô màu cho đối tượng tại vị trí chuột hiện tại.
	{
		addAction();
		C_FillColor *p1 = new C_FillColor;
		p1->BoundaryFill(&m_BG, m_point_s.x, m_point_s.y, cCurrentColor,
m_BG.GetPixel(point));
		delete p1;
	}
	if (dType == MOVE)   //Thực hiên di chuyển đối tượng đến vị trí chuột hiện tại.
	{
		addAction();
		int tempX = point.x;
		int tempY = point.y;
		for (int i = 0; i <= select_row; i++)
		{
			tempX = point.x;
			for (int j = 0; j <= select_column; j++)
			{
				m_BG.SetPixel(tempX, tempY, selectArea[i][j]);
				tempX++;
			}
			tempY++;
		}
		removeSelected(temp_s);
		temp_s = point;
	}
	if (dType == COPY)   //Tạo bảng sao đối tượng đã được chọn trước đó tới vị trí hiện tại
	{
		addAction();
		int tempX = point.x;
		int tempY = point.y;
		for (int i = 0; i <= select_row; i++)
		{
			tempX = point.x;
			for (int j = 0; j <= select_column; j++)
			{
				m_BG.SetPixel(tempX, tempY, selectArea[i][j]);
				tempX++;
			}
			tempY++;
		}
```

```
		temp_s = point;
	}
	CView::OnLButtonDown(nFlags, point);
}
```
Khi thả nút trái chuột ra:
```
void CPaintAppView::OnLButtonUp(UINT nFlags, CPoint point)
{
		m_point_f = point;  //vị trí hiện tại khi nút trái nhả ra
		switch (dType)
		{
		case (LINE) :
		//vẽ đường thẳng từ vị trí chuột ban đầu m_point_s tới vị trí hiện tại (m_point_f)
		{
			addAction();
			C_Line *p1 = new C_Line;
			p1->DrawLine(&m_BG, m_point_s, m_point_f, cCurrentColor);
			delete p1;
			break;
		}
		case (REGTANGLE) :
		{
		//tương tự như vẽ đường thẳng (2 điểm đầu – điểm cuối m_point_s, m_point_f)
			addAction();
			C_Regtangle *p1 = new C_Regtangle;
			p1->DrawRectangle(&m_BG, m_point_s, m_point_f, cCurrentColor);
			delete p1;
			break;
		}
		case(CIRCLE) : //vẽ đường tròn
		{
		//tương tự như vẽ đường thẳng (2 điểm đầu – điểm cuối m_point_s, m_point_f)
			addAction();
			m_point_mid.x = (m_point_s.x + m_point_f.x) / 2;
			m_point_mid.y = (m_point_s.y + m_point_f.y) / 2;
			iRadius = sqrt((float)(m_point_f.x - m_point_mid.x)*(m_point_f.x -
m_point_mid.x) + (float)(m_point_f.y - m_point_mid.y)*(m_point_f.y - m_point_mid.y));
			C_Circle *p1 = new C_Circle;
			p1->DrawCircle(&m_BG, m_point_mid, iRadius, cCurrentColor);
			delete p1;
			break;
		}
		case (SQUARE) :
		{
		//tương tự như vẽ đường thẳng (2 điểm đầu – điểm cuối m_point_s, m_point_f)
			addAction();
			C_Square *p1 = new C_Square;
```

```
                p1->Draw_Square(&m_BG, m_point_s, m_point_f, cCurrentColor);
                delete p1;
                break;
        }

        case(ELIP) :
        {
                addAction();
                C_Elip *el = new C_Elip;
                el->DrawElip(&m_BG, m_point_s, m_point_f, cCurrentColor);
                delete el;
                break;
        }
        case (ERASE_ALL) :  //xóa hết màn hình khi thả nút chuột trái ra. (trước đó đã chọn chức năng này)
        {
                addAction();
                m_BG.FillSolidRect(0, 0, wWidth, wHeight, RGB(255, 255, 255));
                break;
        }
        case (SELECT) :  //chọn đối tượng: Lấy tọa độ các điểm pixel trong ma trận từ m_point_s tới m_point_f.
        {
                selectArea = new COLORREF*[select_row];
                for (int i = 0; i <= select_row; i++)
                        selectArea[i] = new COLORREF[select_column];
                int tempX = temp_s.x;
                int tempY = temp_s.y;
                for (int i = 0; i <= select_row; i++)
                {
                        tempX = temp_s.x;
                        for (int j = 0; j <= select_column; j++)
                        {
                                selectArea[i][j] = m_BG.GetPixel(tempX, tempY);
                                tempX++;
                        }
                        tempY++;
                }
                break;
        }

        default:
                break;
        }

    m_VDC.BitBlt(0, 0, wWidth, wHeight, &m_BG, 0, 0, SRCCOPY);
    Invalidate(false);
```

```
	CView::OnLButtonUp(nFlags, point);
}
//khi kéo thả chuột
void CPaintAppView::OnMouseMove(UINT nFlags, CPoint point)
{
	CursorLocation = point;
	oldPoint = m_point_f;
	if (nFlags && MK_LBUTTON )
	{
		m_point_f = point;
		m_VDC.BitBlt(0, 0, wWidth, wHeight, &m_BG, 0, 0, SRCCOPY);
		switch (dType)
		{
		case (LINE):
		{
			ShowLocationMove(&m_VDC, m_point_s);
			ShowLocationMove(&m_VDC, point);
			C_Line *p1 = new C_Line;
			p1->DrawLine(&m_VDC, m_point_s, point, cCurrentColor);
			delete p1;
			break;
		}
		case (PENCIL):  //vẽ bút chì theo sự di chuyển của tọa độ chuột
		{
			ShowLocationMove(&m_VDC, point);
			C_Line *line = new C_Line;
			line->DrawLine(&m_BG, oldPoint, point, cCurrentColor);
			oldPoint = point;
			delete line;
			break;
		}
		case (BRUSH) : //tương tự như dùng bút chì
		{
			ShowLocationMove(&m_VDC, point);
			m_BG.FillSolidRect(point.x,point.y,brushStyles,brushStyles, cCurrentColor);
			break;
		}
		case (REGTANGLE): //vẽ hình chữ nhật.
//kéo thả để biết độ rộng của hình chữ nhật
		{
			ShowLocationMove(&m_VDC, m_point_s);
			ShowLocationMove(&m_VDC, point);
//VDC – hệ thiết bị đồ họa ảo, sẽ không vẽ lên màn hình khi hành động di chuyển chuột kết thúc.
			C_Regtangle *p1 = new C_Regtangle;
			p1->DrawRectangle(&m_VDC, m_point_s, point, cCurrentColor);
			delete p1;
			break;
```

```
		}
		case(CIRCLE) :  //tương tự như vẽ Hình chữ nhật.
		{
			ShowLocationMove(&m_VDC, m_point_s);
			ShowLocationMove(&m_VDC, point);
			m_point_mid.x = (m_point_s.x + m_point_f.x) / 2;
			m_point_mid.y = (m_point_s.y + m_point_f.y) / 2;
			ShowLocationMove(&m_VDC, m_point_mid);
			iRadius = sqrt((float)(m_point_f.x - m_point_mid.x)*(m_point_f.x -
m_point_mid.x) + (float)(m_point_f.y - m_point_mid.y)*(m_point_f.y - m_point_mid.y));
			C_Circle *p1 = new C_Circle;
			p1->DrawCircle(&m_VDC, m_point_mid, iRadius, cCurrentColor);
			delete p1;
			break;
		}
		case (SQUARE) :	//tương tự như vẽ Hình chữ nhật.
		{
			ShowLocationMove(&m_VDC, m_point_s);
			ShowLocationMove(&m_VDC, point);
			C_Square *p1 = new C_Square;
			p1->Draw_Square(&m_VDC, m_point_s, m_point_f, cCurrentColor);
			delete p1;
			break;
		}
		case (ELIP): //tương tự như vẽ Hình chữ nhật.
		{
			ShowLocationMove(&m_VDC, m_point_s);
			ShowLocationMove(&m_VDC, point);
			C_Elip *el = new C_Elip;
			el->DrawElip(&m_VDC, m_point_s, m_point_f, cCurrentColor);
			delete el;
			break;
		}
		case (ERASE) :
//thực hiện việc xóa (tẩy) theo sự di chuyển chuột của người dùng. Tẩy (xóa) đối tượng trên màn
hình thực chất là cho màu Pixel đó thành màu trắng (RGB(0,0,0) trùng với màu nền của khung
vẽ.
		{
			ShowLocationMove(&m_VDC, point);
			m_BG.FillSolidRect(point.x, point.y, brushStyles, brushStyles, RGB(255, 255,
255));
			break;
		}
		case (SELECT) :
		{
//kéo thả để xác định độ rộng vùng chọn.
			ShowLocationMove(&m_VDC, m_point_s);
```

```
                    ShowLocationMove(&m_VDC, point);
                    C_Regtangle *p1 = new C_Regtangle;
                    p1->DrawRectangle(&m_VDC, m_point_s, point, (0,0,0));
                    select_column = abs((point.x - m_point_s.x));
                    select_row = abs((point.y - m_point_s.y));
                    temp_s = m_point_s;
                    temp_f = m_point_f;
                    delete p1;
                    break;
            }
            default:
                    break;
            }

            Invalidate(false);
    }
    CView::OnMouseMove(nFlags, point);
}
```

f. Lớp MainFrame:

- Dùng để khai báo các đối tượng xuất hiện trên màn hình chính của Ứng dụng.

```
class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)
public:
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);  //cửa sổ chính
        void SetColumns(int nCols, CToolBar &tbToolBox);
// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:
        CStatusBar      m_wndStatusBar;             //Thanh Status
        CToolBar                m_wndDrawToolBar;  //Thanh Draw
        CToolBar                m_wndColorToolBar;  //Thanh chọn màu
        CToolBar                m_wndPenToolBar;  //Thanh nét vẽ
protected:
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        DECLARE_MESSAGE_MAP()
public:
        afx_msg void OnViewBardraw();
};
```

- Khai báo các thanh công cụ (trong hàm MainFrame.cpp)

```
        //tạo status bar        .
        if (!m_wndStatusBar.Create(this))
        {
                TRACE0("Failed to create status bar\n");
                return -1;      // fail to create
        }
        m_wndStatusBar.SetIndicators(indicators, sizeof(indicators) / sizeof(UINT));
        m_wndStatusBar.SetPaneInfo(1, ID_INDICATOR_POS, SBPS_NORMAL, 180);

        //tạo color bar ở dưới (thanh chọn màu).
        if (!m_wndColorToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_FIXED)
                || !m_wndColorToolBar.LoadToolBar(IDR_COLOR_TOOLBAR))
        {
                return -1;
```

```
	}
	m_wndColorToolBar.SetBarStyle(m_wndColorToolBar.GetBarStyle() | CBRS_TOOLTIPS |
CBRS_FLYBY | CBRS_SIZE_FIXED);
	m_wndColorToolBar.SetButtonStyle(1,	m_wndColorToolBar.GetButtonStyle(1)	|
TBBS_WRAPPED);

	m_wndColorToolBar.EnableDocking(CBRS_ALIGN_ANY);
	EnableDocking(CBRS_ALIGN_BOTTOM);
	DockControlBar(&m_wndColorToolBar, AFX_IDW_DOCKBAR_BOTTOM);
	SetColumns(9, m_wndColorToolBar);

	//tạo Draw toolbar bên tay trái.
	if (!m_wndDrawToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_FIXED)
		|| !m_wndDrawToolBar.LoadToolBar(IDR_DRAW_TOOLBAR))
	{
		return -1;
	}
	m_wndDrawToolBar.SetBarStyle(m_wndDrawToolBar.GetBarStyle() | CBRS_TOOLTIPS |
CBRS_FLYBY | CBRS_SIZE_FIXED);
	m_wndDrawToolBar.EnableDocking(CBRS_ALIGN_ANY);
	EnableDocking(CBRS_ALIGN_LEFT);
	DockControlBar(&m_wndDrawToolBar, AFX_IDW_DOCKBAR_LEFT);
	SetColumns(2, m_wndDrawToolBar);

	// tạo thanh Nét vẽ bên phải.
	if (!m_wndPenToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_FIXED)
		|| !m_wndPenToolBar.LoadToolBar(IDR_PEN_TOOLBAR))
	{
		return -1;
	}
	m_wndPenToolBar.SetBarStyle(m_wndPenToolBar.GetBarStyle()	|	CBRS_TOOLTIPS	|
CBRS_FLYBY | CBRS_SIZE_FIXED);
	m_wndPenToolBar.SetButtonStyle(1,	m_wndPenToolBar.GetButtonStyle(1)	|
TBBS_WRAPPED);
	m_wndPenToolBar.EnableDocking(CBRS_ALIGN_ANY);
	EnableDocking(CBRS_ALIGN_RIGHT);
	DockControlBar(&m_wndPenToolBar, AFX_IDW_DOCKBAR_RIGHT);
	SetColumns(1, m_wndPenToolBar);
```

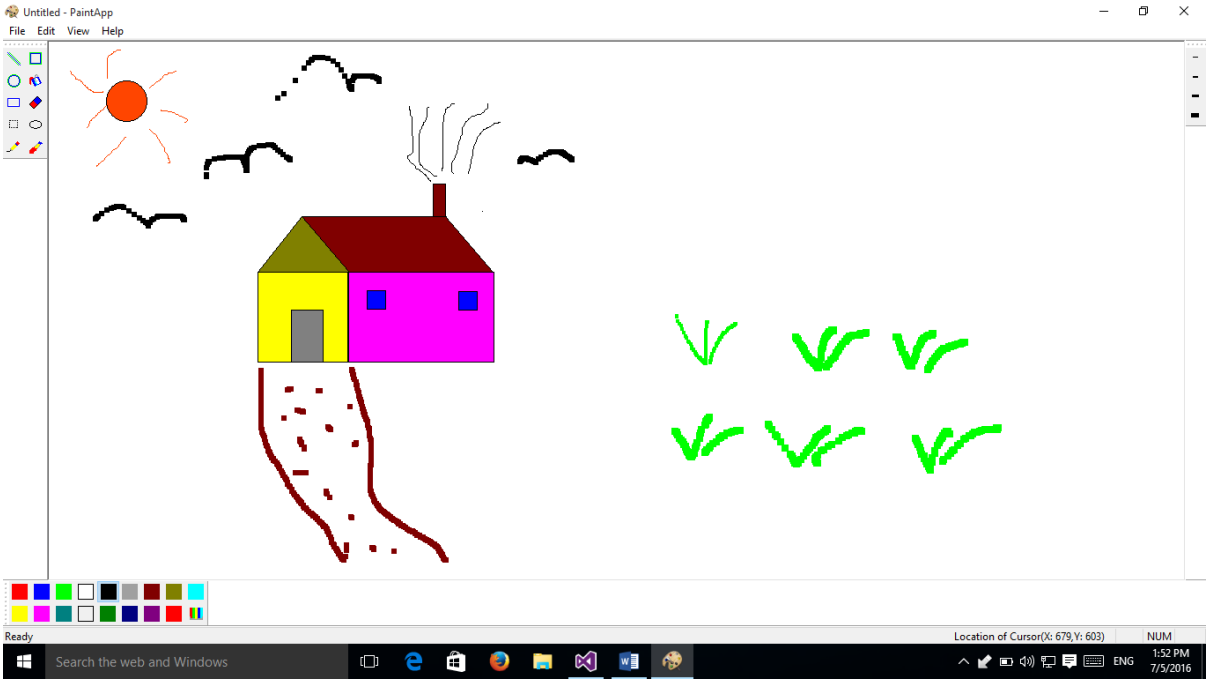
g. Save và Open File:

- Hai chức năng này được khai báo trong lớp PaintAppView.h, dùng để lưu File vẽ vào ổ cứng hoặc Load File vẽ từ ổ cứng lên màn hình.

```
//xử lý Lưu file vào ổ cứng
void CPaintAppView::OnFileSave()
{
	CFileDialog saveFile(FALSE, L"*.bmp", NULL, OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT | OFN_PATHMUSTEXIST, L"bitmap(*.bmp)|*.bmp||");
	if (saveFile.DoModal() == IDOK)
	{
		FileName = saveFile.GetPathName();
		CImage image;
		image.Create(wWidth,wHeight, 32);

		for (int i = 0; i < wWidth; i++)
			for (int j = 0; j < wHeight; j++)
				image.SetPixel(i, j, m_BG.GetPixel(i, j));
		image.Save(_T("" + FileName));
	}
}
//Xử lý Load File từ ổ cứng vào Chương trình
void CPaintAppView::OnFileOpen()
{
	CFileDialog openFile(TRUE, L"*.bmp", NULL, OFN_FILEMUSTEXIST | OFN_PATHMUSTEXIST, L"bitmap(*.bmp)|*.bmp||");
	if (openFile.DoModal() == IDOK)
	{
		FileName = openFile.GetPathName();

		CImage image;
		image.Load(_T("" + FileName));
		for (int i = 0; i < image.GetWidth(); i++)
			for (int j = 0; j < image.GetHeight(); j++)
				m_BG.SetPixel(i, j, image.GetPixel(i, j));
	}
}
```

4. Đánh giá chương trình

- Chương trình đã thực hiện được các chức năng vẽ cơ bản như: Đường thẳng, Hình chữ nhật, hình tròn, hình vuông, vẽ bằng bút, vẽ bằng cọ, xóa hình, copy, move, Undo.



- Chương trình còn có chức năng Lưu trữ file vẽ lên ổ cứng và Load file từ ổ cứng vào Chương trình.
- Tuy nhiên, chương trình còn có một số hạn chế: Chưa vẽ được đường cong, giải thuật Tô màu còn thực hiện chậm khi tô màu trong vùng không gian lớn, Chưa có các phím tắt cho các công cụ vẽ. Đó là một số hạn chế của Chương trình. Nhìn chung, chương trình cơ bản là đáp ứng được mục tiêu đặt ra của đồ án: Xây dựng chương trình vẽ các đối tượng đồ họa cơ bản.

CHƯƠNG 5: BẢNG PHÂN CÔNG THỰC HIỆN ĐỒ ÁN

Bảng phân công đây sẽ dựa vào những tiêu chí sau với đánh giá dựa vào phần trăm thực hiện của từng thành viên:

- **Ý tưởng Đề tài:** Lưu Thanh Sơn (50%), Đỗ Phú An (50%).
- **Design cho các button của chương trình:** Lưu thanh Sơn (30%), Đỗ Phú An (60%), 10% còn lại là tham khảo source code trên mạng (tại trang sharecode.vn)
- **Code chương trình:**
    - Lưu Thanh Sơn (50%): gồm các phần: đường thẳng, hình vuông, hình chữ nhật, Lệnh Undo, Lệnh Move, Lệnh Copy. Tô Màu.
    - Đỗ Phú An (50%): gồm các phần: đường tròn, hình Elip, bút chì vẽ Pencil, cọ vẽ Brush, cải tiến lệnh Undo.
- **Báo cáo:**
    - Lưu Thanh Sơn (50%): xây dựng dàn bài nội dung cho báo cáo
    - Đỗ Phú An (50%): hiệu chỉnh cho báo cáo (chỉnh sửa nội dung, căn lề, chính tả, font chữ,…) cho báo cáo

TÀI LIỆU THAM KHẢO

1. Hoàng Kiếm, Dương Anh Đức, Vũ Hải Quân, Lê Đình Duy – Giáo trình *Đồ họa máy tính* – Trường Đại học Công nghệ Thông tin – ĐHQG.TPHCM, Nhà Xuất bản Đại học Quốc gia TPHCM – 2010.
2. Hoàng Kiếm (Cb), Dương Anh Đức, Vũ Hải Quân, Lê Đình Duy – Giáo trình *Cơ sở Đồ họa máy tính* – Nhà Xuất bản Giáo dục – 2012.
3. Nguyễn Đình Quyền, Mai Xuân Hùng, Giáo trình *Lập trình C trên Windows* – Trường Đại học Công nghệ Thông tin – ĐHQG.TPHCM, Nhà Xuất bản Đại học Quốc gia TPHCM – 2008.
4. Website:https://vi.wikipedia.org/wiki/Gi%E1%BA%A3i_thu%E1%BA%ADt_Bresenham_v%E1%BA%BD_%C4%91o%E1%BA%A1n_th%E1%BA%B3ng.
5. Website: https://en.wikipedia.org/wiki/Cohen%E2%80%93Sutherland_algorithm.
6. Website: https://en.wikipedia.org/wiki/Flood_fill.
7. Website: https://**msdn**.microsoft.com/vi-vn.