

Kỹ thuật đồ họa

Computer Graphics

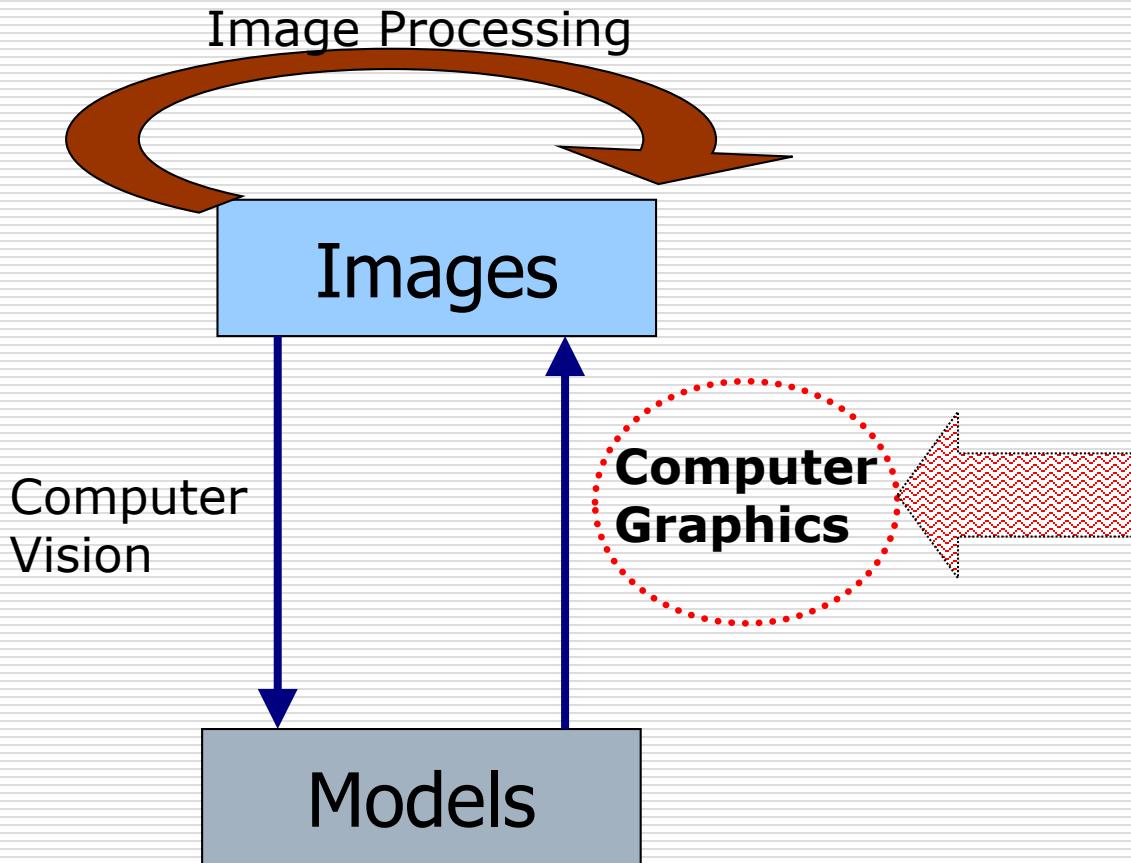
Sinh viên làm việc với ai?

- Giáo viên: Trần Nguyên Ngọc
 - Nơi công tác: Bộ môn KHMT, khoa CNTT, Tầng 2 nhà A1 HVKTQS.
 - Email: tnn1999@mail.ru
 - Giờ học:
 - Phòng học:
 - Giải đáp câu hỏi: 8h30-9h45, thứ 6 hàng tuần, tại bộ môn
-

Đồ họa máy tính (ĐHMT) là gì?

- Đây là một hướng quan trọng của khoa học máy tính
- Đối tượng nghiên cứu của ĐHMT là việc thiết lập, lưu trữ và xử lý các mô hình dưới dạng hình ảnh của chúng trên máy tính.
- Môn học **ĐHMT là một môn tin học**

Tổng quan về ĐHMT&XLA



Nội dung khóa học

- Môn học thuộc nhóm: Đồ họa máy tính và xử lý ảnh
- Tổng số tiết:
- Kiến thức gồm 4 chương:
 - Chương I. Nhập môn kỹ thuật đồ họa
 - Chương II. Đồ họa hai chiều
 - Chương III. Đồ họa 3 chiều
 - Chương IV. Giới thiệu một số kỹ thuật xử lý đồ họa và các hướng ứng dụng

Để trả thi tốt sinh viên cần:

- Nắm các nguyên lý cơ bản của ĐHMT
 - Hiểu được các kiến thức hình học khi xây dựng các mô hình hai, ba chiều
 - Biết cách lập trình đồ họa trong môi trường Windows
 - Có khái niệm tổng quan về các ứng dụng của ĐHMT trong thực tế hiện nay
-

Sẽ học ĐHMT thế nào?

- Tự ôn tập các kiến thức cơ bản về đại số tuyến tính & hình học không gian
 - Ôn tập các kiến thức về lập trình hướng đối tượng
 - Học cách biểu diễn đồ họa và các thuật toán lập trình đồ họa
 - Học cách hiện thực thuật toán trên các môi trường lập trình Windows
 - Tham gia nghiên cứu, thảo luận các đề tài khoa học về ĐHMT
-

Tài liệu tham khảo

Số TT	Tên tài liệu	Tác giả	Năm xuất bản	Nhà xuất bản	Nước xuất bản
1	Đồ họa máy tính (2 tập + CD)	Trần Giang Sơn	2008	KHKT	Việt Nam
2	Компьютерная графика	Залогова Л.А.	2006	ЛБЗ- Москв а	Nga
3	Computer graphics	Donald Hearn, M. Pauline Baker	1996	Prentice- Hall, Inc	Mỹ
4	Graphics Gems	Andrew S. Glassner	1995	AP Profes sinal	Mỹ

Kết luận

- ✓ Learning instead of Teaching
- ✓ Learning by Doing

Ứng dụng đồ họa và các thiết bị đồ họa máy tính

Nội dung

- Ứng dụng của đồ họa**
 - Các thiết bị hiển thị**
 - Màn hình CRT
 - Màn hình tinh thể lỏng
 - Plasma
 - Các thiết bị in**
 - Máy in kim
 - Máy in laser
 - Máy in nhiệt
 - Hệ thống đồ họa trên PC**
 - chẽ độ màn hình
 - Kiến trúc VIDEO RAM
-

Computer Graphics & Image processing

Đồ họa máy tính	Xử lý ảnh
<ul style="list-style-type: none"><input type="checkbox"/> Tạo ra các hình ảnh<input type="checkbox"/> Tổng hợp các hình ảnh bằng máy tính	<ul style="list-style-type: none"><input type="checkbox"/> Biến đổi ảnh, xóa nhiễu, làm tốt ảnh<input type="checkbox"/> Trích chọn các đặc trưng của ảnh<input type="checkbox"/> Phân tích ảnh

Ưu điểm của tương tác đồ họa

- Sử dụng đồ họa trong giao diện với người dùng
 - Sự phát triển của đồ họa máy tính ngày càng rộng rãi
 - các chế độ đồ họa 2D và 3D phục vụ trong các lĩnh vực xã hội khác nhau như KH, giáo dục, y học, kỹ thuật, thương mại và giải trí.
 - Tính năng ảnh 2D, 3D phát triển cao cho phép xử lý các dữ liệu ảnh một cách nhanh chóng.
 - Nhấn mạnh những ưu điểm: bằng máy tính chúng ta có thể tạo được không những hình ảnh của thế giới thực (real world) mà còn cả những vật trừu tượng và các ảnh tổng hợp.
 - Cho phép hiển thị những hình ảnh động.
 - Đồ họa có thể tạo ra những kết quả hay sản phẩm có chất lượng cao hơn và chính xác hơn, nồng suất hơn, và giảm chi phí thiết kế.
-

Ứng dụng của đồ họa

arts, publicity - Đây là động lực chính trong ĐHMT hiện nay

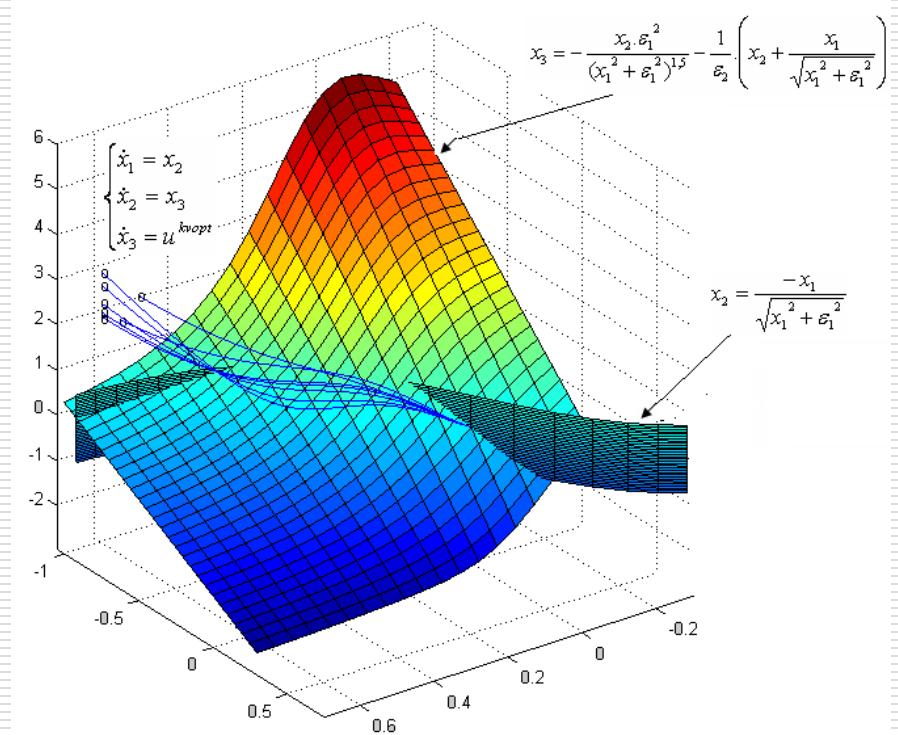
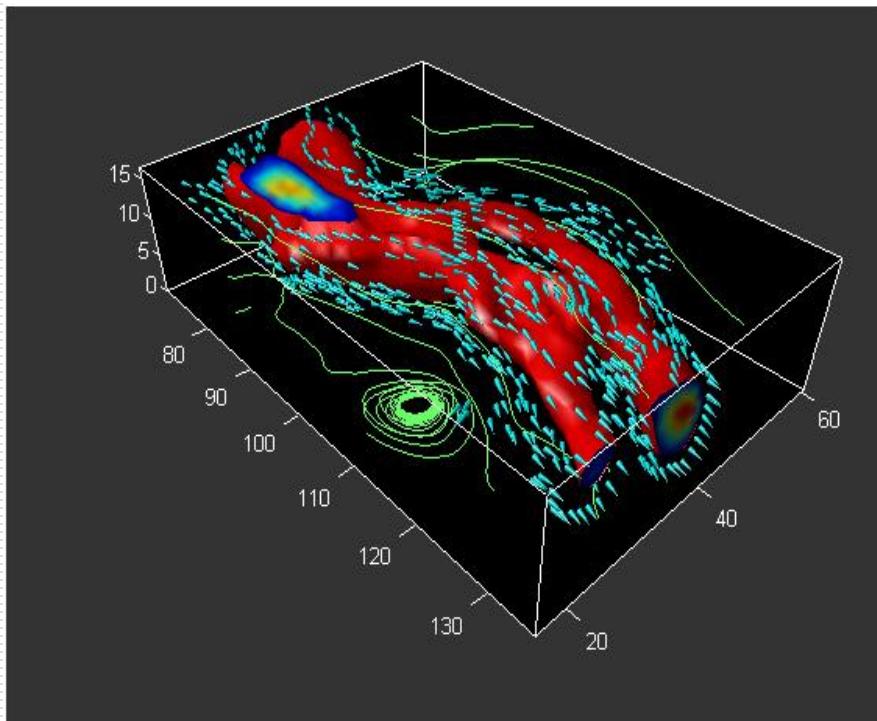


- ❑ Phần mềm hỗ trợ PaintShop Pro, Adobe Photoshop,...tạo cảm giác y như đang làm việc ngoài đời thực.
- ❑ Chương trình trò chơi, kỹ xảo điện ảnh



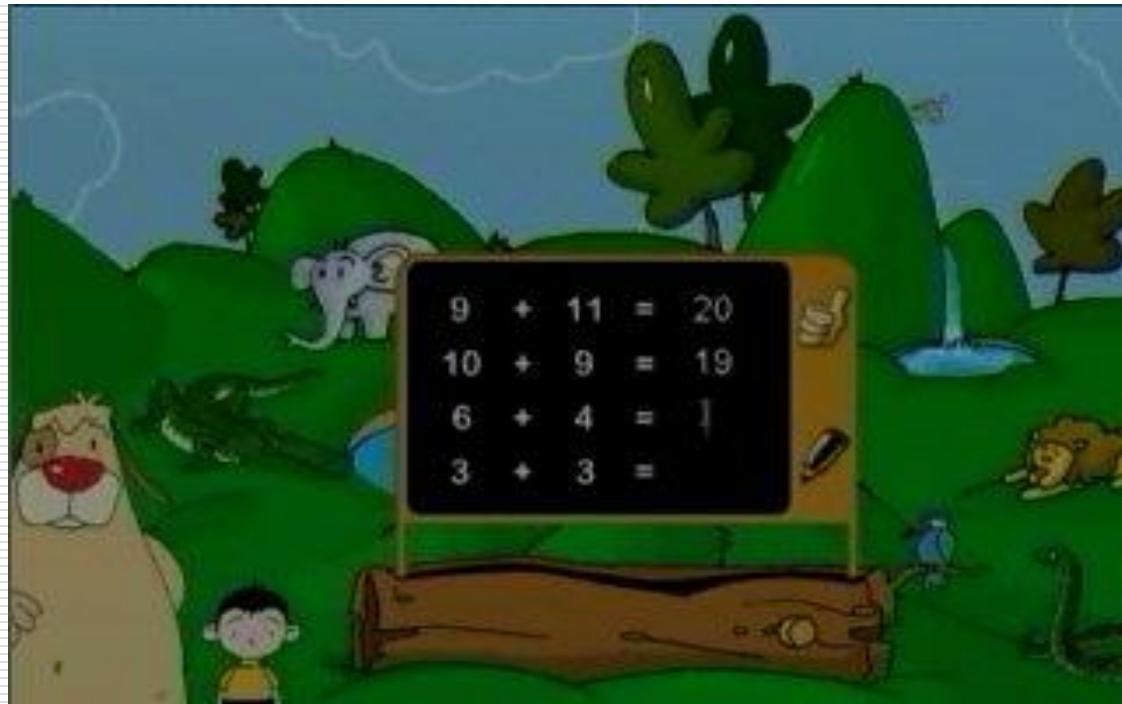
Ứng dụng của đồ họa

Scientific visualizations



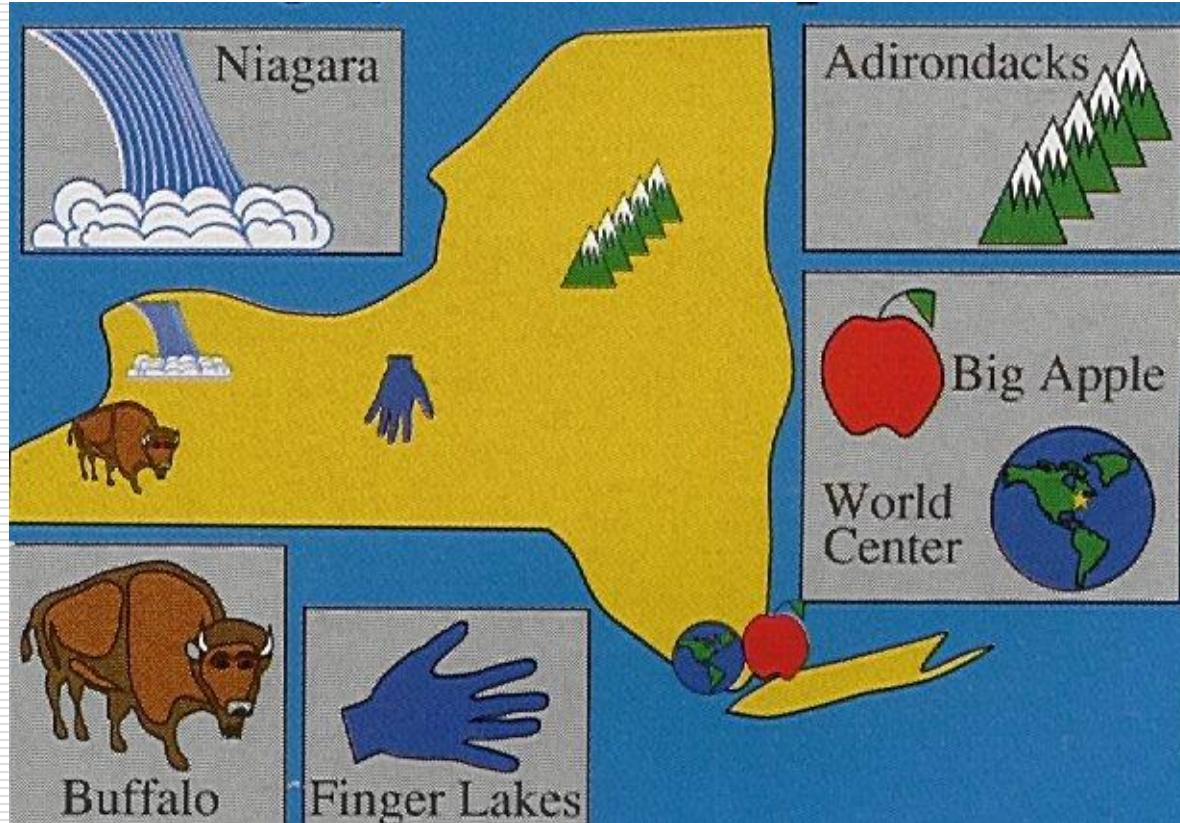
Ứng dụng của đồ họa

Education, training



Ứng dụng của đồ họa

User Interfaces



-
- Dùng hệ thống cửa sổ quản lý các hoạt động diễn ra đồng thời và các đối tượng trên màn hình. Nhấn chuột, chọn menu, biểu tượng, MS Windows.

Ứng dụng của đồ họa

Computer-Aided Design



-
- Ứng dụng chính của đồ họa tương tác đối tượng không gian.
 - Thiết kế các thành phần và hệ thống cơ khí, điện, các thiết bị điện tử, xây dựng, thiết kế thân ô tô, thân máy bay và tàu thủy, chip giao diện rộng, hệ thống cáp quang, mạng điện thoại và máy tính...
 - Phác thảo khung, hiệu chỉnh theo chiều bất kỳ, kết hợp mô hình chiếu sáng, tô màu, tạo bóng...tạo kết quả cuối cùng rất gần với thế giới thực.
-

Ứng dụng của đồ họa

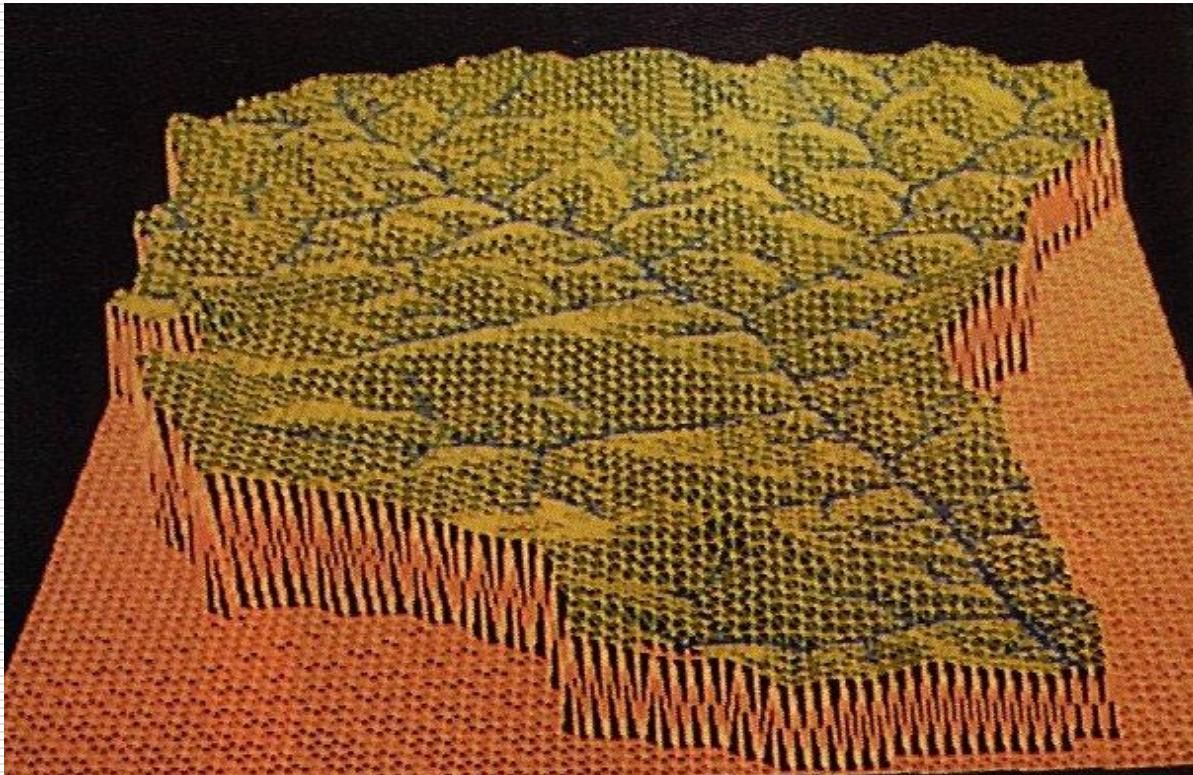
Geographical Information



Sơ đồ vị trí địa lý và các hiện tượng tự nhiên chính xác

Ứng dụng của đồ họa

Terrain Modeling



Ứng dụng của đồ họa

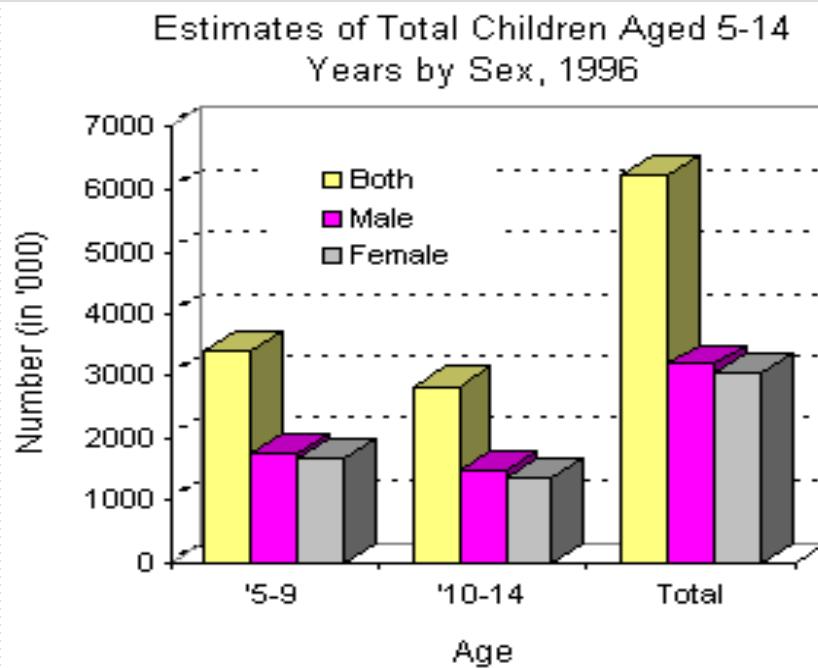
Xử lý những ảnh y tế
cũng là một động lực
khác của ĐHMT

Thu được nhiều đầu tư
Xúc tiến việc kết nối
ĐHMT với video,
máy quét, v.v.



Ứng dụng của đồ họa

Business



And many
others....

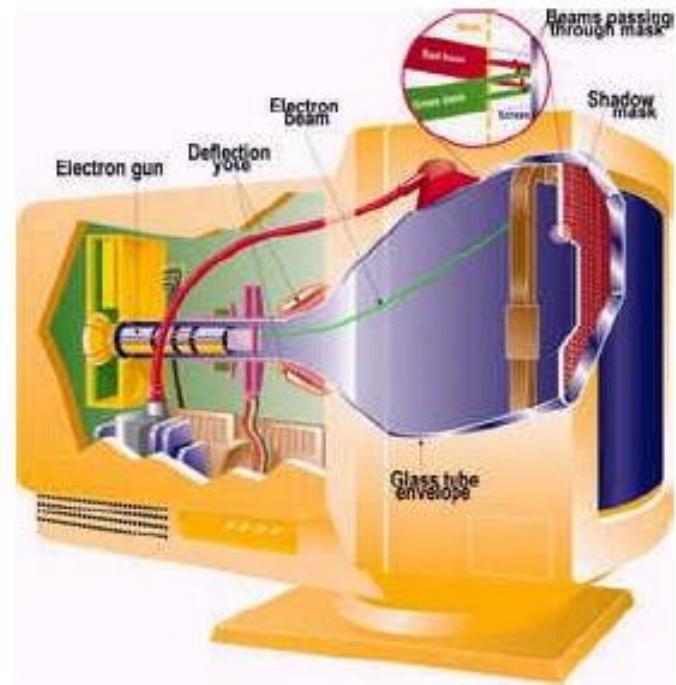
Các thiết bị hiển thị

□ Màn hình đồ họa

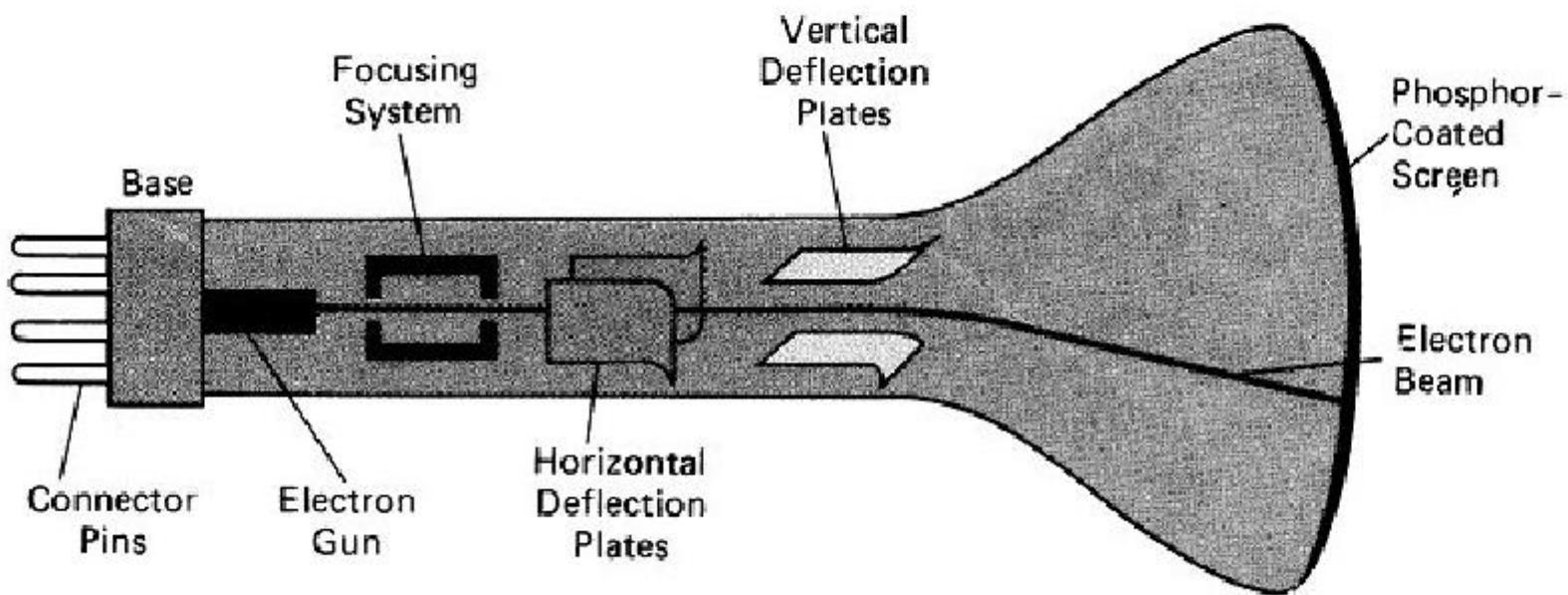
- CRT (Cathode Ray Tubes)
- LCD (Liquid crystal display)
- Plasma

CRT (cathod ray tubes)

- Là công nghệ của hầu hết các màn hình ngày nay.
- Ống thuỷ tinh chân không.
 - Là ống tạo ra chùm tia điện tử ở một đầu rồi tăng tốc các điện tử đó để chúng bị phóng về phía trước,
 - gắn một màn hình thủy tinh mà bên trong được phủ một lớp phốt pho
 - chùm tia điện tử đập vào thì lóe sáng lên.



Công nghệ màn hình CRT



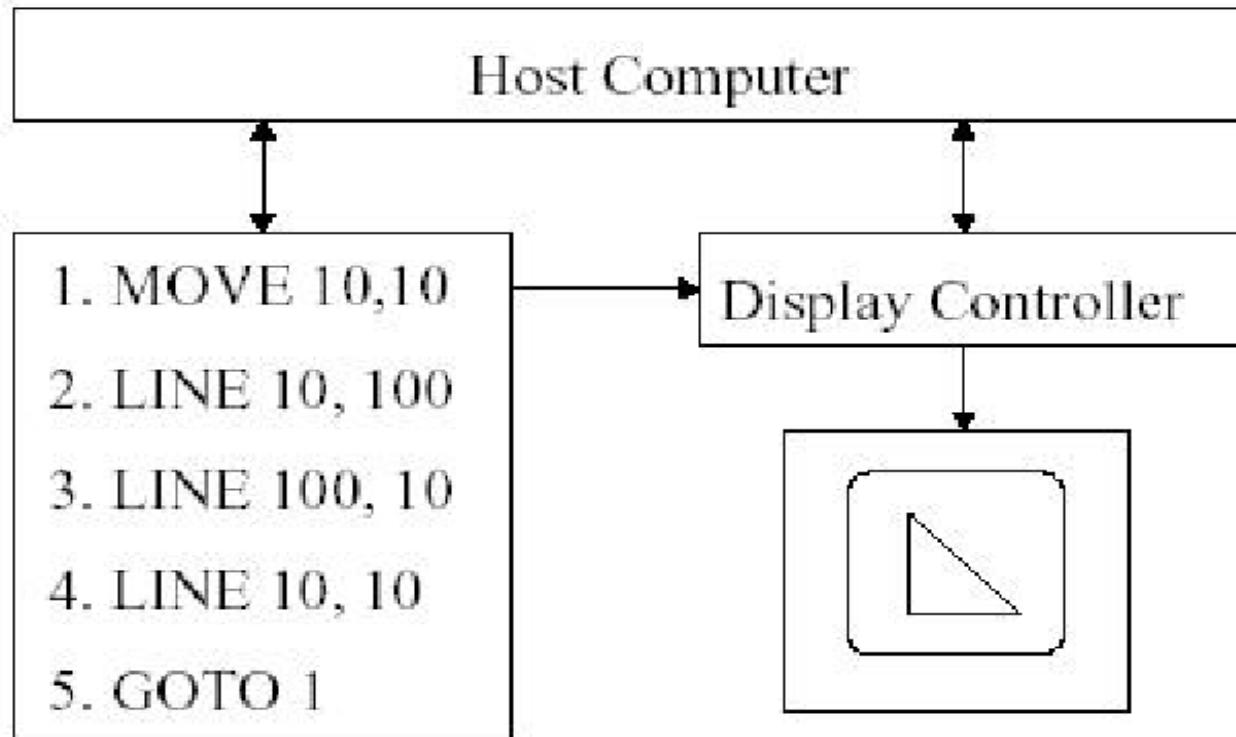
CRT

- Catot: khi được nung nóng lên thì phát ra các điện tử
- Lưới điều khiển:
 - là một cái chén bằng kim loại
 - nối với một điện áp âm thay đổi để làm thay đổi lực đẩy của nó đối với các điện tử.
 - Khi lực đẩy này cân bằng với lực hút của anot thì dòng điện tử bị ngừng, không gây ra chấm sáng trên màn hình, còn khi cường độ yếu thì gây ra chấm sáng yếu.
- Anot: luôn được duy trì ở một điện áp dương cường độ cao để hút và tăng tốc dòng điện tử về phía màn hình.
- Bộ phận hội tụ (focusing): có tập trung các hạt điện tử thành dòng sao cho khi đạt tới màn hình thì dòng này hội tụ thành một chấm nhỏ.
- bộ phận lái tia: gồm hai cặp
 - một cặp lái tia theo phương x để lái chùm tia điện tử theo chiều ngang màn hình
 - cặp kia lái theo phương y để lái chùm tia điện tử theo chiều thẳng đứng.

Hiển thị vector

- Vẽ đoạn thẳng bằng cách di chuyển chùm tia điện tử từ điểm đầu tới điểm cuối (hoặc ngược lại)
 - Không bị hiệu ứng bậc thang
 - Tốn ít bộ nhớ
 - Hoạt hình tốt hơn
 - Giá thành cao
 - ảnh phức tạp tối hơn

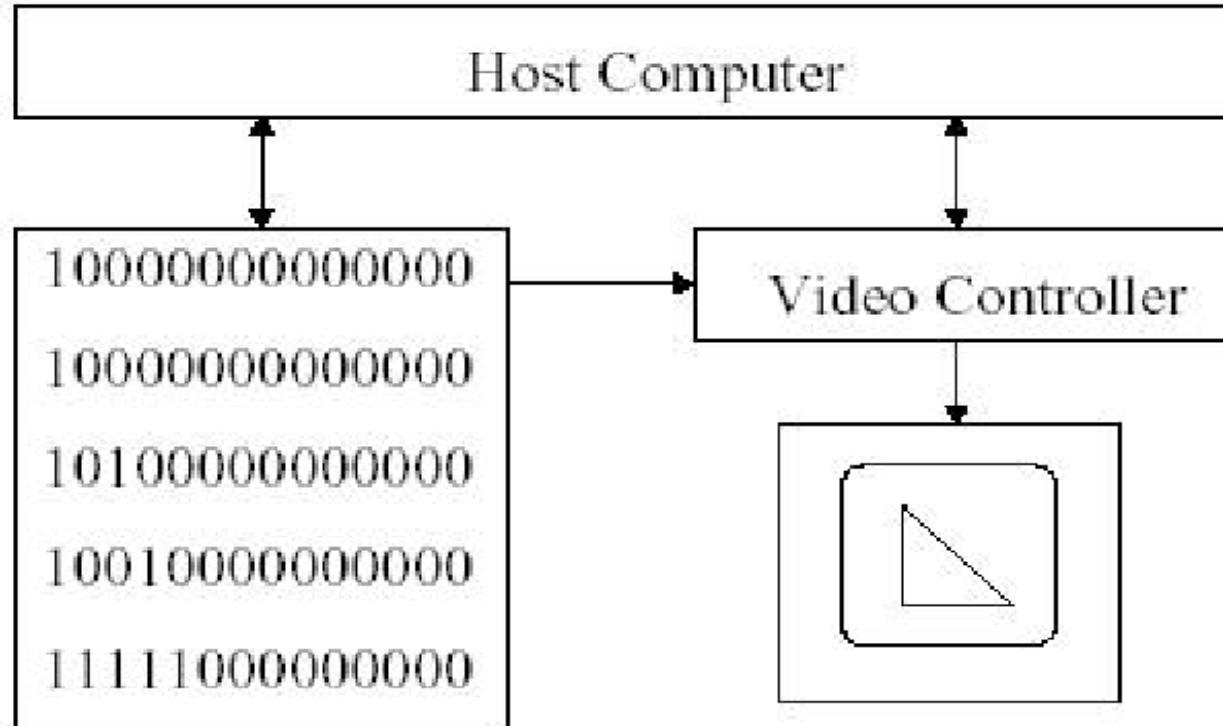
Kiến trúc màn hình vector



Raster display

- Xuất hiện vào đầu những năm 70.
- Tương tự như TV, quét mọi pixel (mẫu đều)
 - sử dụng Video RAM (Frame buffer) để giải quyết đồng bộ.
 - 256 kb RAM giá 2 triệu \$ vào năm 1971
 - Màn hình đơn sắc cẩn 160 Kb
- Màn hình màu độ phân giải cao cẩn đến 5.2 Mb
- Các đối tượng đồ họa cơ sở (line, region, ký tự...) được lưu thành các pixel trong Video RAM.
- CRTC điều khiển quét lặp.
- Pixel là các điểm ảnh rời rạc trên đường quét.

Kiến trúc màn hình Raster

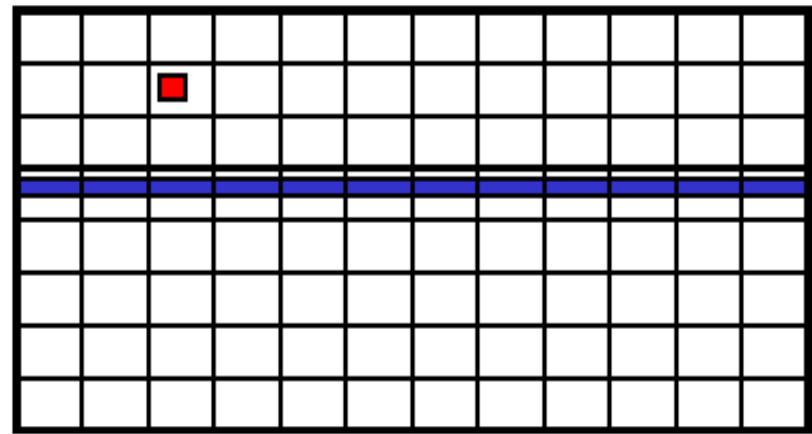


Các định nghĩa cơ sở

- Bitmap là tập hợp các pixels.
- Frame buffer lưu trữ các bitmap
- Raster display lưu trữ các phần tử nguyên thủy (line, characters, and solid shaded or patterned area)
- Frame buffers
 - Hình thành từ Video RAM.
- Video RAM là bộ nhớ dual-ported có khả năng
 - Xâm nhập ngẫu nhiên
 - Đầu ra nối tiếp tốc độ cao: thanh ghi dịch nối tiếp cho khả năng xuất toàn bộ scanline với tốc độ cao, đồng bộ với đồng hồ điểm ảnh.

Các định nghĩa cơ sở

- Raster: mảng các hình vuông của các points hay dots.
- Pixel (picture element): một phần tử dot hay phần tử ảnh của raster.
- Scan line: một hàng các pixels

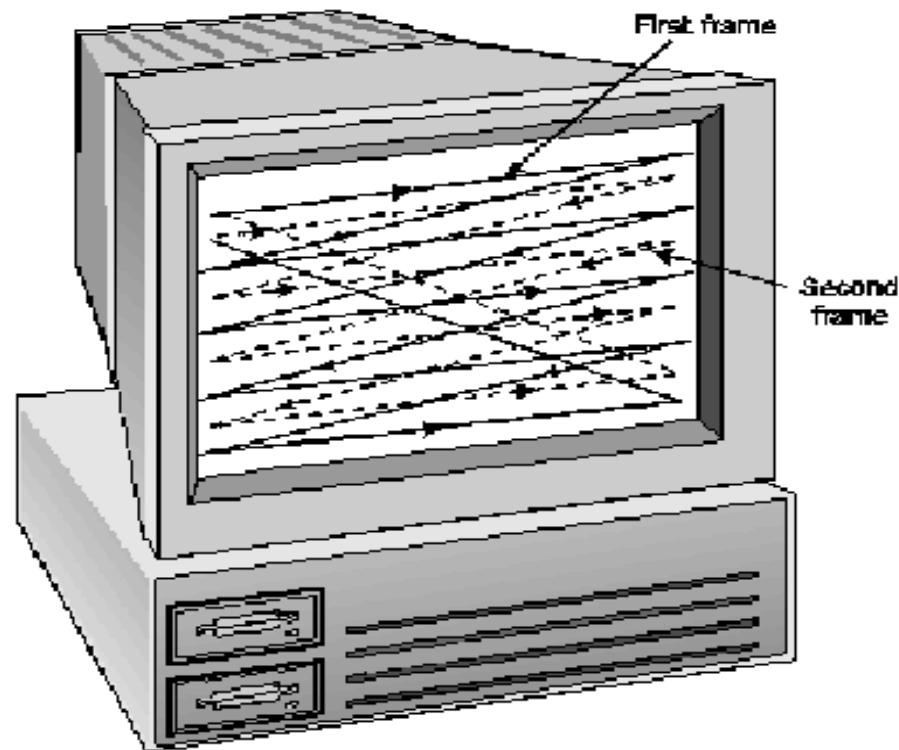


Các thiết bị raster hiển thị ảnh bằng cách vẽ trình tự các điểm ảnh của Scan lines để hình thành raster.

Làm tươi Raster

số lần trên 1 giây ảnh được vẽ lại (thường là 60 lần/s đối với raster)

Do ánh sáng phosphor yếu.



Resolution

- Tổng số điểm cực đại các điểm có thể hiển thị trên màn hình CRT.
- Phụ thuộc vào
 - Loại phosphor
 - Cường độ được hiển thị
 - hệ thống focusing và deflection
- REL SGI 02 monitors: 1280 x 1024

Aspect Ratio

□ Frame aspect ratio

(FAR)=horizontal/vertical size

- TV 4:3
- HDTV 16:9
- Page 8.5:11~3/4

□ Pixel aspect ratio (PAR)=FAR

vres/hres

Refresh rates and bandwidth

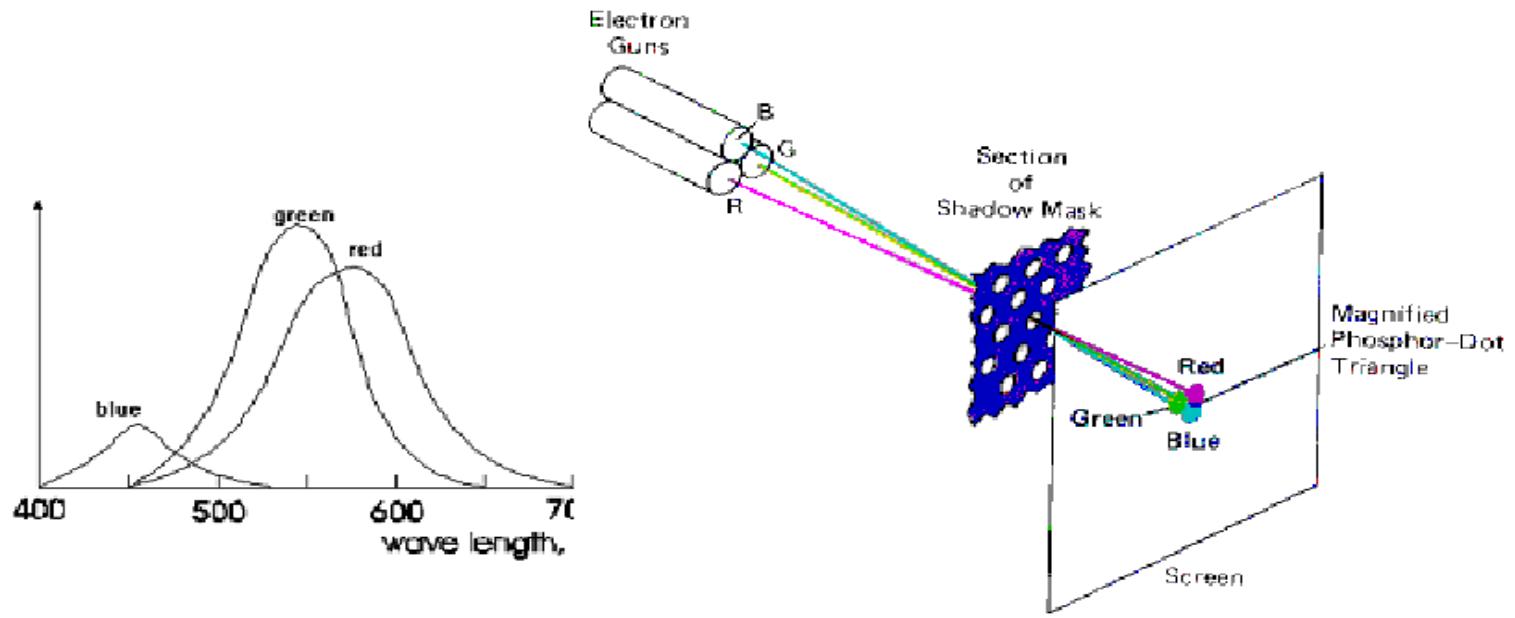
- Frames per second (FPS)
- Film (double framed) 24 FPS
- TV (interlaced) 30 FPS $\times \frac{1}{4} = 8\text{MB/s}$
- Workstation (non-interlaced) 75 FPS
 $\times 5 = 375\text{ MB/s}$

Interlaced scanning

- Scan frame 30 lần trong 1 giây
- để tránh nhấp nháy, chia frame thành hai trường
 - Các dòng quét chẵn
 - Các dòng quét lẻ
- Quét luân phiên các nhóm chẵn lẻ để tạo ra ảnh đan chiều nhau.

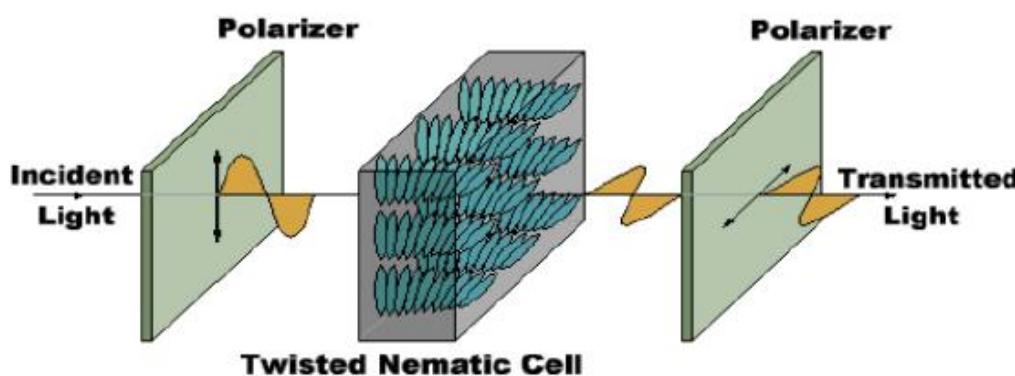
Công nghệ màn hình CRT màu

- Các CRT màu có
 - Mặt trong của tấm kính màn hình không phủ phosphor đồng chất mà là lớp khàm gồm những chấm nhỏ li ti gọi là những "triad", mỗi triad gồm 3 chấm tròn kề sát nhau
 - Ba súng bắn tia điện tử
 - Mặt nạ "shadow mask" để khu biệt các tia điện tử



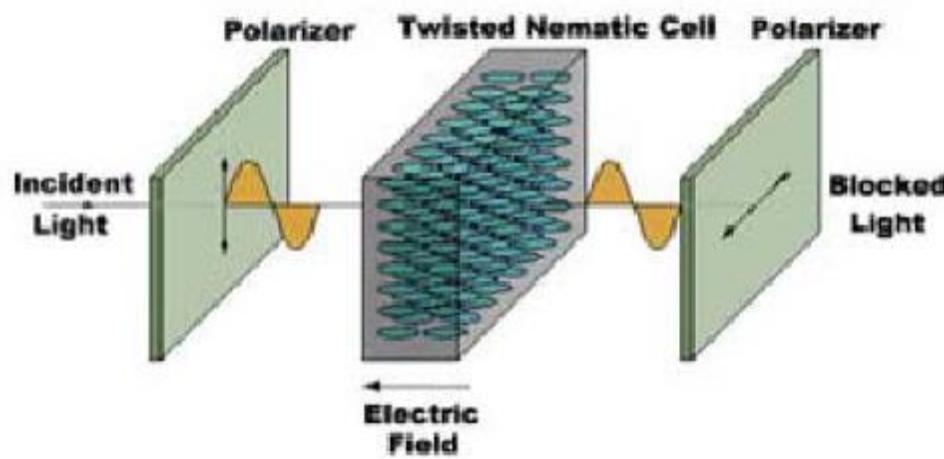
Công nghệ màn hình LCD

- Liquid Crystal Displays (LCDs)
 - LCDs: phân tử hữu cơ, trạng thái tự nhiên: kết tinh, nó bị hóa lỏng khi bị đốt nóng hay có trường điện từ (E field)
 - LCD có các cells cho ánh sáng đi qua
 - Trạng thái tinh thể làm xoắn ánh sáng cực 90° .
 - Tốn ít năng lượng, phẳng, nhẹ
 - Chất lượng ảnh phụ thuộc vào góc độ quan sát



Công nghệ màn hình LCD

- Quá độ giữa trạng thái tinh thể và trạng thái lỏng của LCD là tiến trình từ từ.
- Tương tự phosphors, LCDs ở trạng thái “on” trong khoảng thời gian sau khi có E field. Do vậy, crystals cần phải được làm tươi.



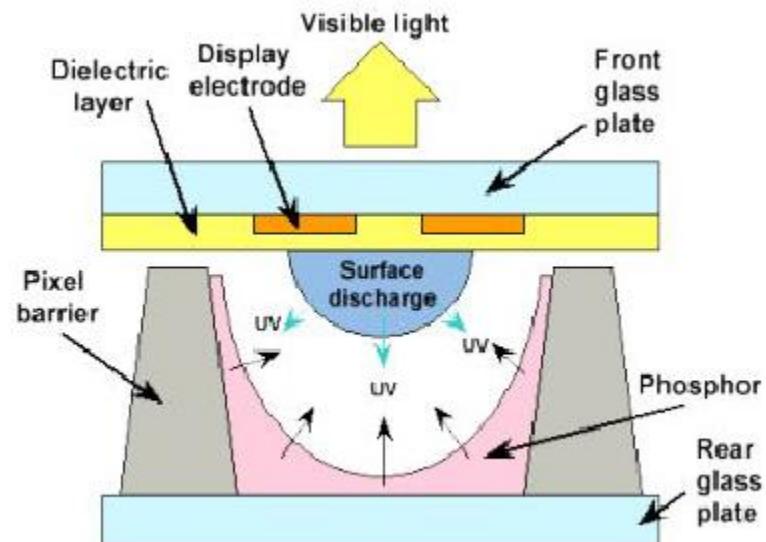
LCD

□ Transmissive & reflective LCDs:

- LCDs hoạt động như “van”, không phát ánh sáng do vậy nó phụ thuộc vào nguồn sáng ngoài.
- Laptop screen: backlit, transmissive display
- Palm Pilot/Game Boy: reflective display

Công nghệ màn hình Plasma

- Nguyên lý tương tự đèn huỳnh quang
- Một ống nhỏ đầy gas: khi bị tác động bởi trường điện từ nó phát ra ánh sáng UV
- UV tác động lên phosphor
- Phosphor phát ra một vài màu khác.



Công nghệ màn hình Plasma

□ Pros

- Góc quan sát rộng
- Phù hợp với màn hình rộng
- Ánh sáng rõ

□ Cons

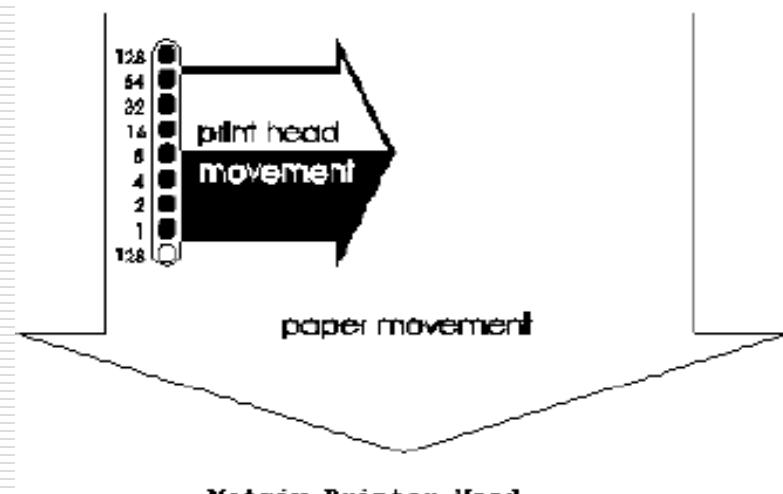
- Đắt tiền
- Kích thước pixel khá lớn ($\sim 1\text{mm}$ so với $\sim 0.2\text{mm}$)
- Phosphor bị yếu dần
- Ánh sáng yếu hơn CRTs, sử dụng nhiều năng lượng

Máy in

- Máy in kim
- Máy in Laser
- Máy in nhiệt

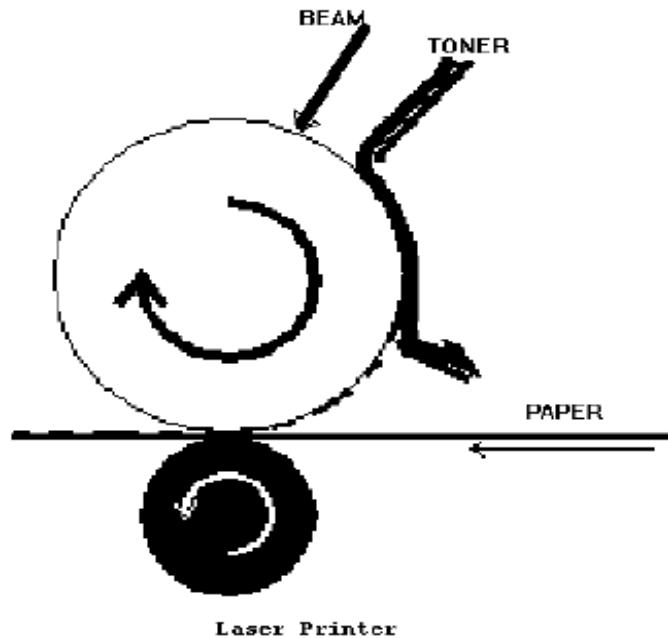
Máy in ma trận điểm-dot matrix printer

- Đầu in có 9 hoặc 24 kim
 - 9 kim cho max 240 dpi chiều ngang và 72 chiều đứng
 - Mỗi lần quét ngang qua giấy, in 9 hàng điểm tương ứng với 9 hàng pixel trên màn hình.
 - 24 kim cho 360 dpi chiều ngang và 360 chiều đứng



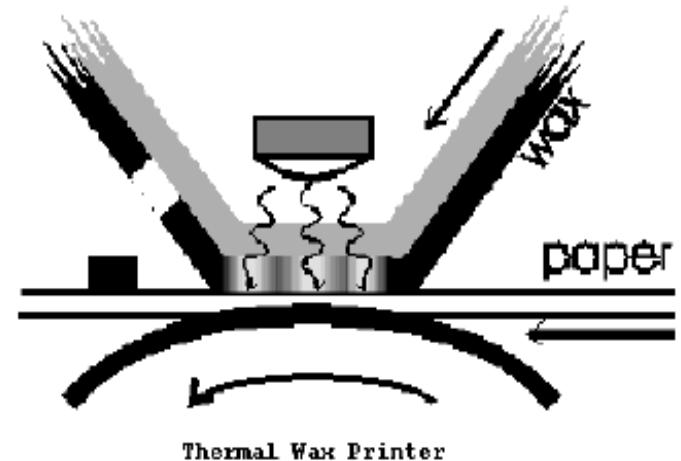
Laser printer

- Tương tự photocopy (Xerox), kỹ sư Bungari
- Trống được phủ selenium
- Tia laser chiếu vào trống làm thay đổi điện tích tĩnh trên bề mặt trống, sau đó một bộ phận tự động rót mực. Mực bám vào nơi có tia laser chiếu.
- Mực tích điện (toner)
 - Mật độ cao 300-1200 dpi
 - In từng trang: 1 trang A4 độ phân giải 300x300 cẩn 1Mb

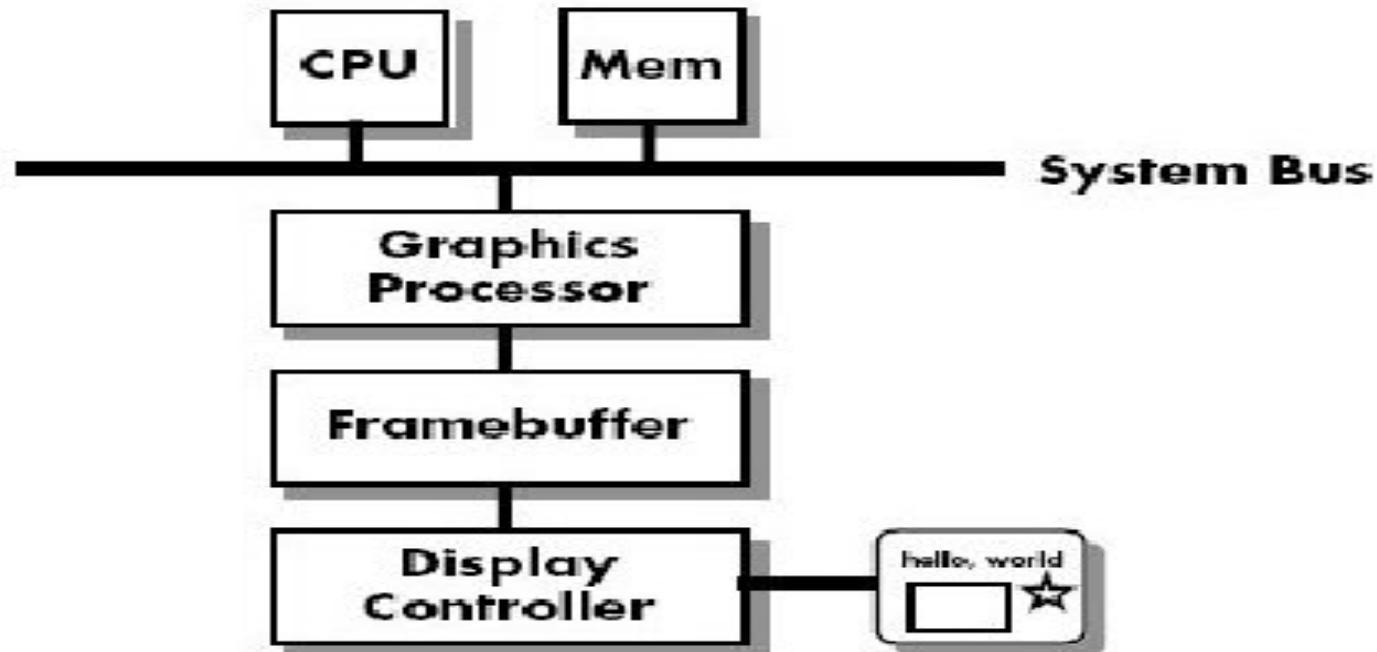


Thermal printer

- Hai loại máy in nhiệt
 - Thermal - wax (sáp) transfer printer: sáp từ băng mực bị đốt nóng truyền sang giấy
 - Dye (nhuộm) sublimation printer: đầu in đốt nóng giấy, giấy đổi màu.



Cấu trúc hệ thống đồ họa trên PC



Raster Graphics System Architecture

Frame buffer

- Khối nhớ lưu giữ nội dung đồ họa sẽ hiển thị
- Pixel là 1 phần tử của frame buffer.
- Vấn đề ở đây frame buffer lớn như thế nào:
 - Với màn hình $640 \times 480 \rightarrow$ framebuffer = $640*480$ bits
- Bit depth: số bits cho mỗi pixel trong buffer

DAC Digital to Analog Converter

- Giá trị trong frame buffer sẽ được chuyển từ số thành tín hiệu tương tự để hiển thị ra màn hình. DAC thực hiện thao tác này, mỗi frame thực hiện một lần.

Bit depths

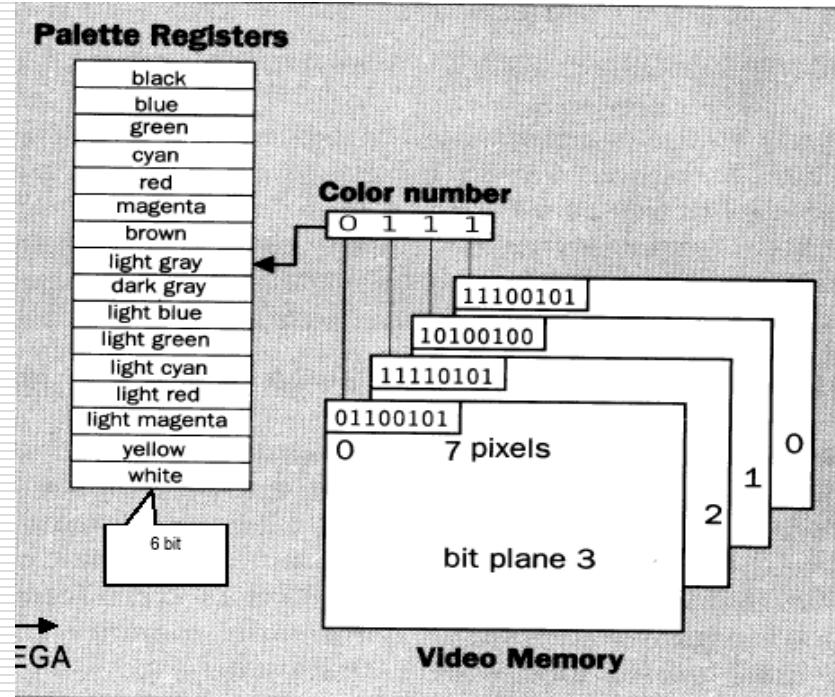
- 16 bits per pixel (high color)
 - 5 bits for red, 6 bits for green, 5 bits for blue
 - potential of 32 reds, 32/64 green, 32 blues
 - total colors: 65536
- 32 bits per pixel (true color)
 - 8 bits for red, green, blue, and alpha
 - potential for 256 reds, greens, and blues
 - total colors: 16777216 (more than the eye can distinguish)

Graphic Card Memory

- How much memory is on our graphic card?
 - $640 * 480 * 32 \text{ bits} = 1,228,800 \text{ bytes}$
 - $1024 * 768 * 32 \text{ bits} = 3,145,728 \text{ bytes}$
 - $1600 * 1200 * 32 \text{ bits} = 7,680,000 \text{ bytes}$
- How much memory is on your graphics card?

Video RAM-Bộ nhớ màn hình

- ☐ Vùng bộ nhớ chứa dữ liệu trực tiếp hiện ra màn hình, dữ liệu trong bộ nhớ màn hình thay đổi sẽ trực tiếp thay đổi trên màn hình



Bộ nhớ màn hình

- Ở chế độ đồ họa bộ nhớ của máy dành cho màn hình được bắt đầu từ địa chỉ A000:0000
- Có 2 cách tổ chức là
 - Dạng gói (packed format)
 - tổ chức dạng mảng một chiều, mỗi phần tử mảng là 1 byte
 - Dạng mảng (Bit plane)
 - tổ chức logic thành mảng hai chiều
 - Ví dụ: graphics mode 16 màu bộ nhớ màn hình được tổ chức thành 4 mảng (4 bit plane) đánh số từ 0->3. Tất cả đều được truy cập cùng đ/c A000:0000

Bộ nhớ màn hình

- Truy nhập các bitplane bằng
 - lệnh thông qua cổng
 - Ngắt của Rom Bios
 - Nguyên tắc: chỉ đọc/ghi 1 byte trên cùng một bit plane. Muốn đọc thông tin của một pixel phải đọc 4 byte, rồi tách 4 bit plane từ 4 byte này. Ghi 1 pixel ta cũng phải ghi 4 byte
-

Trang màn hình

- Vùng bộ nhớ màn hình ứng với lượng thông tin hiển thị trên một màn hình.
- Vùng bộ nhớ màn hình chuẩn đạt tới 256Kb, lượng thông tin hiển thị trên một màn hình lại nhỏ hơn -> tốc độ hiển thị nhanh, bộ nhớ cho phép hiển thị thành từng trang màn hình đánh số từ 0.
- Trang đang hiển thị là trang làm việc (active page)
- tuỳ thuộc vào chế độ làm việc của màn hình mà bộ nhớ màn hình có số trang khác nhau.
- Ta có thể tự hạn chế số trang cho mỗi chế độ màn hình

Truy cập vào bộ nhớ màn hình

- Có 2 cách
 - sử dụng các dịch vụ ngắt của ROM BIOS, ROM BIOS cung cấp tương đối đầy đủ các chương trình con ứng với các ngắt để phục vụ cho công việc này
 - Truy cập trực tiếp đưa vào các đ/c của các cổng (port) thông qua các ngôn ngữ lập trình.

Sử dụng dịch vụ ngắt 10H của ROM BIOS

□ Disadvantage

- Chậm không đáp ứng được các yêu cầu của ứng dụng đồ họa

Truy cập trực tiếp qua cổng và bộ nhớ màn hình

□ 320x200 256 màu

- Mode \$13, bộ nhớ màn hình tổ chức dạng gói gồm $320 \times 200 = 64000$ phần tử (64K): bộ nhớ dành cho A000:0000 đến A000:FFFF
- Để vẽ điểm (x, y) với màu color lên màn hình ta chỉ cần gán `Mem[$A000:320*y+x]:=color`
- Để đọc màu của điểm (x, y) chỉ cần `color:=Mem[$A000:320*y+x]`

Truy cập trực tiếp qua cổng và bộ nhớ màn hình

□ 640x480 16 màu

- giả sử mode 16 màu, độ phân giải $n \times m$, xác định byte chứa pixel có tọa độ (x, y) :
- $k = y * n \text{ div } 8 + x \text{ div } 8$
- vị trí bit = $7 - (x \text{ mod } 8)$

Truy cập trực tiếp qua cổng và bộ nhớ màn hình

□ 640 x 480 256 màu

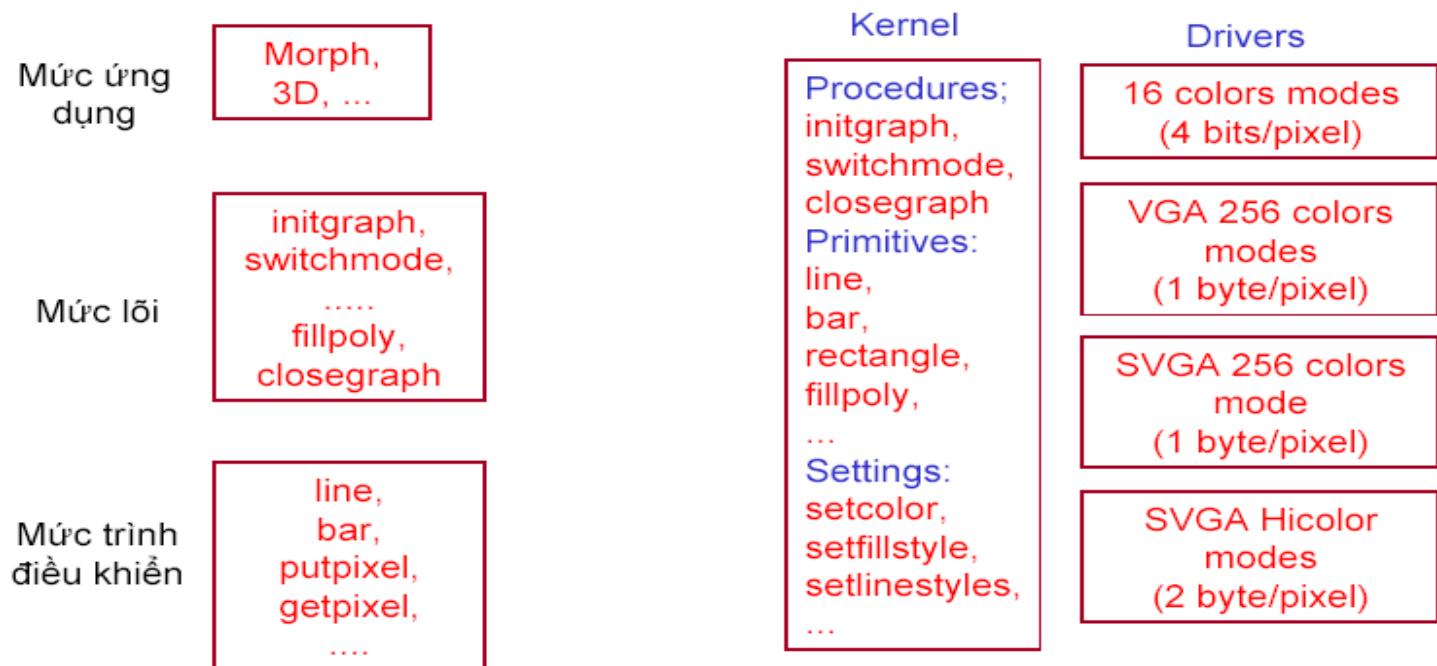
- Màn hình được tổ chức thành 5 mảng 1 chiều, mỗi mảng gồm 65536 phần tử-> cần có 5 mảng, mỗi mảng gọi là Bank (dải)
- Điểm (x, y) trên màn hình sẽ là điểm thứ linear_address:=y*640+x, nằm ở bank thứ bank_num:=linear_address Div 65536; đ/c điểm đó nằm ở bank này là pixel_offset:=linear_address Mod 65536

Lập trình đồ họa

☐ Nhiệm vụ

- Nhận biết loại card màn hình bằng BIOS
- Khởi động chế độ đồ họa
- vẽ đồ họa
- Kết thúc chế độ đồ họa

Thiết kế hệ thống đồ họa



Tóm tắt

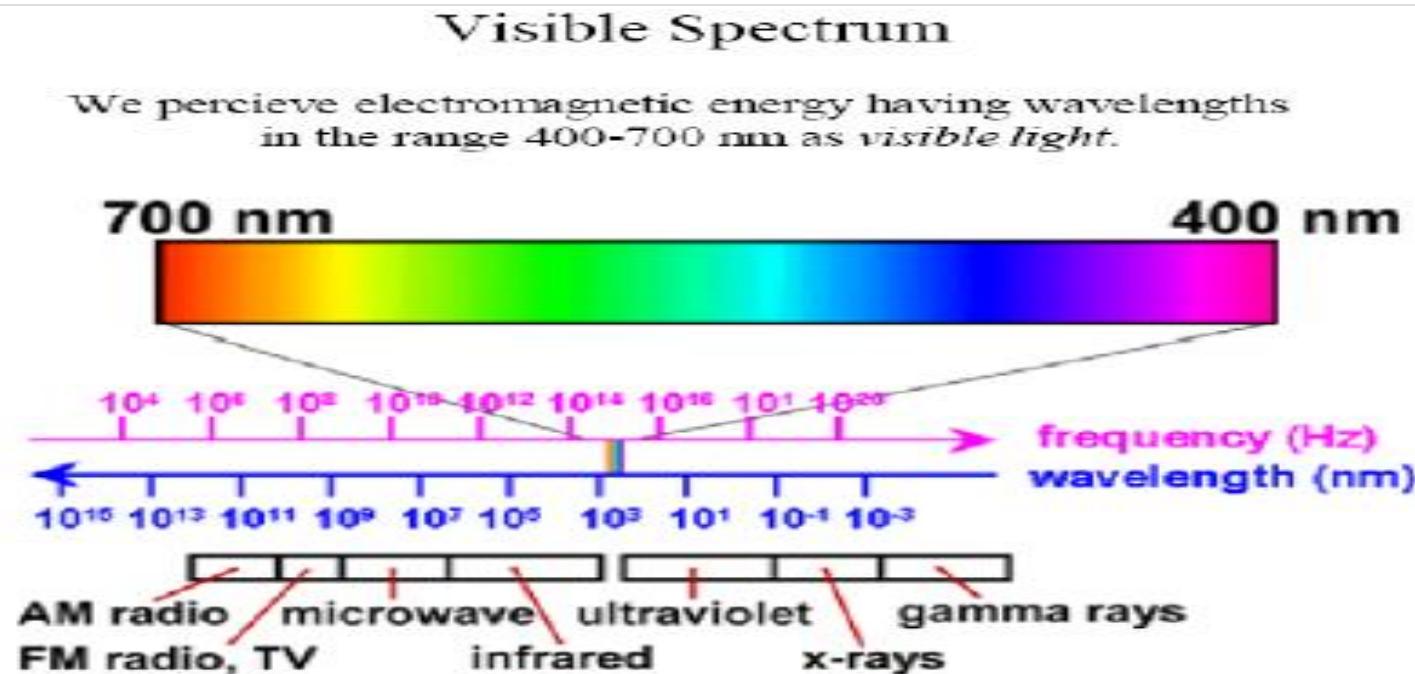
- Ứng dụng đồ họa
- Các loại màn hình
- Các loại máy in
- Kiến trúc Video RAM
- Xây dựng hệ thống đồ họa

MÀU VÀ KHÔNG GIAN MÀU TRÊN MÁY TÍNH

Computer Graphics

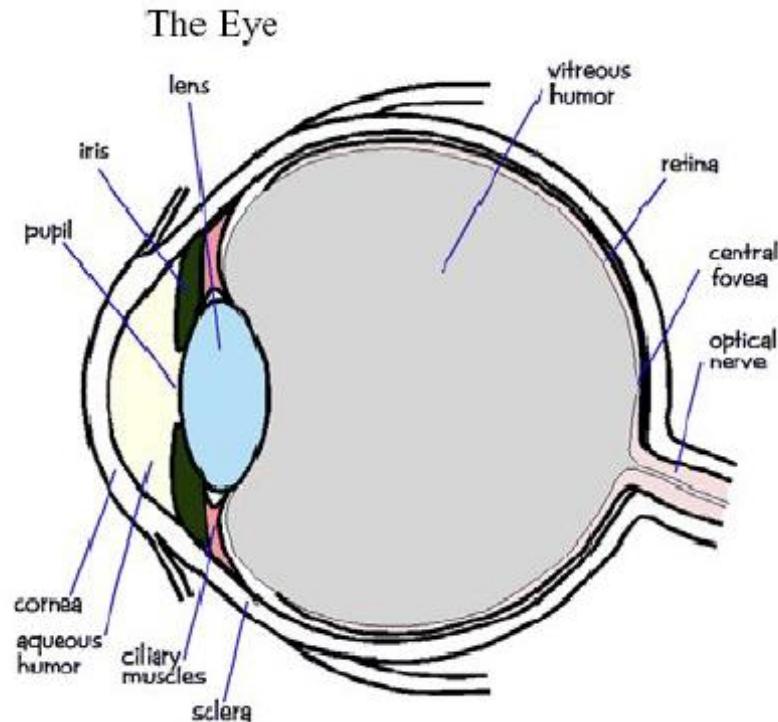
Ánh sáng nhìn thấy là gì?

Ánh sáng mà con người nhận biết (hay màu khác nhau) là dải tần hẹp trong quang phổ điện tử



Con người cảm nhận ánh sáng thế nào?

- Phản nhạy cảm với ánh:
võng mạc (retina)
 - Retina bao gồm hai loại tế bào: rod (que) và cone(nón)
 - Cone có trách nhiệm nhận biết màu.
- Cones có ba loại: S, M, L
tương ứng với cảm biến B
(430 nm), G (560nm),
R(610nm)

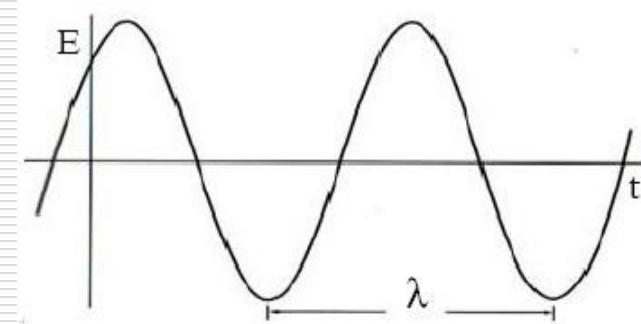


Đặc tính của ánh sáng

- Vì ánh sáng là sóng điện từ cho nên có thể mô tả nó bằng tần số hay bước sóng
 - Ánh sáng mặt trời truyền đi mọi tần số trong dải nhìn thấy để tạo ra ánh sáng trắng
 - Khi ánh sáng trắng chiếu lên đối tượng:
 - Một vài tần số phản xạ, một số khác bị hấp thụ
 - tổ hợp của các tần số phản xạ hình thành cái gọi là màu đối tượng
Ví dụ: nếu tần số thấp chiếm ưu thế -> màu đỏ
 - Tần số (bước sóng) chiếm ưu thế được gọi là Color/Hue hay Light
- Khi ta quan sát nguồn sáng, mắt ta đáp ứng màu và hai cảm giác khác
 - Luminance (Brightness): Liên quan đến cường độ (năng lượng) ánh sáng: năng lượng càng cao -> nguồn sáng càng chói.
 - Purity (saturation): độ tinh khiết của màu sáng
- **Vậy ánh sáng có 3 đặc tính cơ bản liên quan trực tiếp tới cảm nhận của mắt người là: tần số, độ chói và độ tinh khiết**

Màu là gì?

- Có nhiều định nghĩa về màu (không có định nghĩa chính thức)
 - từ góc nhìn khoa học:
 - Màu là phân bổ các bước sóng λ (red: 700nm, violet: 400nm)
 - Và tần số f
(Tốc độ ánh sáng: $c=\lambda f$)
 - từ góc nhìn nghệ thuật và cuộc sống
 - Màu là tổ hợp của Hue (sắc), Brightness (độ sáng), saturation (sự bão hòa) của ánh sáng



Mô hình màu & Không gian màu

- Mô hình màu là phương pháp diễn giải các đặc tính và tác động của màu trong ngữ cảnh nhất định
- Không có mô hình màu nào là đầy đủ cho mọi khía cạnh của màu
 - sử dụng các mô hình màu khác nhau để mô tả các tính chất được nhận biết khác nhau của màu
- Mỗi mô hình màu được biểu diễn trong một không gian màu sắc
 - Không gian RGB: ánh sáng Red, Green, và Blue ứng dụng cho màn hình, TV.
 - Không gian CMYK: máy in
 - Không gian HSV: nhận thức con người

Mô hình màu & Không gian màu

- Ánh sáng có thể hình thành từ hai hay nhiều nguồn
 - Lựa chọn cường độ phù hợp cho hai nguồn màu khác nhau sẽ hình thành được các màu khác nhau.
- Nếu tổ hợp 2 nguồn để có màu trắng -> gọi chúng là complementary colors.
 - Ví dụ: Red+Cyan, Green+Magenta, Blue+Yellow
- Mô hình màu được sử dụng để mô tả tổ hợp ba colors để có dải màu (gamut - gam màu)
- Hai hay ba màu được sử dụng để mô tả các màu khác được gọi là primary colors (đây chính là cơ sở của không gian màu).
- Thực tế là số primary colors là không có giới hạn. Tuy nhiên chỉ 3 màu cơ sở đã đủ cho phần lớn các ứng dụng.
- Mô hình màu được sử dụng để biểu diễn màu duy nhất trong hệ thống màu ba hay nhiều chiều.

Định lý cơ bản về biểu diễn màu

Định lý Gassman(1853)

1. Để biểu diễn một màu bất kỳ cần ít nhất 3 thành phần
2. Trong tổ hợp 3 thành phần của màu, nếu thay đổi liên tục một thành phần và giữ nguyên hai thành phần còn lại thì màu tổ hợp cũng biến đổi liên tục theo
3. Màu tổ hợp chỉ phụ thuộc vào các thành phần tạo nên mà không phụ thuộc vào cách thức tạo nên các thành phần đó

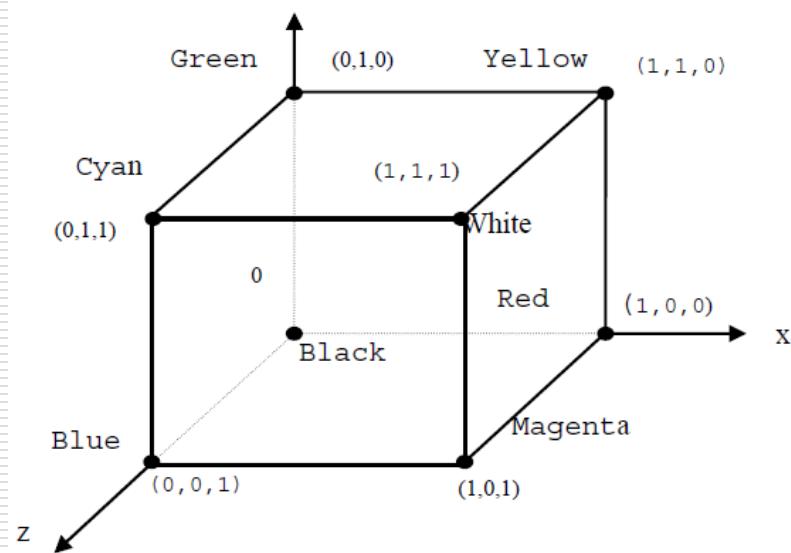
Không gian màu RGB

- Biểu diễn RGB thuộc mô hình cộng:
 - Phát sinh màu mới bằng cách cộng cường độ màu cơ sở
- Gán giá trị từ 0 đến 1 cho R, G, B
 - Red+Blue -> Magenta (1, 0, 1)
 - Đường chéo từ (0, 0, 0) đến (1, 1, 1) biểu diễn màu xám
- Nhận xét
 - Mô hình này không thể biểu diễn mọi màu trong phổ nhìn thấy
 - đủ cho các ứng dụng máy tính
 - Màn hình máy tính và TV sử dụng mô hình này
 - Được sử dụng rộng rãi nhất
 - Đơn giản

Màu và các giá trị tương ứng trong không gian RGB

RGB Value	Short Name	Long Name
[1 1 0]	y	yellow
[1 0 1]	m	magenta
[0 1 1]	c	cyan
[1 0 0]	r	red
[0 1 0]	g	green
[0 0 1]	b	blue
[1 1 1]	w	white
[0 0 0]	k	black

$$Color = [x, y, z]$$
$$x, y, z \in [0,1]$$

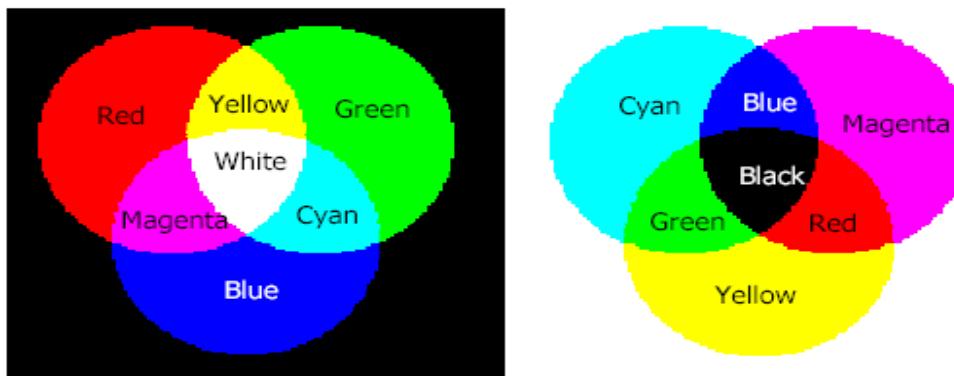


Không gian màu CMYK

- Với màn hình: màu là tổ hợp các ánh sáng phản xạ từ phosphor.
- Với giấy: phủ chất màu lên giấy, mắt ta nhận biết ánh sáng phản xạ sau khi chất màu đã hấp thụ
 - Mực viết chữ màu đen có nghĩa rằng mực đã hấp thụ toàn bộ ánh sáng nhìn thấy trên nó
 - **Những dòng chữ này có màu green vì mực hấp thụ toàn bộ bước sóng tương ứng với màu R và B. Ánh sáng còn lại phản xạ vào mắt ta.**

Mô hình màu CMYK

- Mô hình màu xác định bởi màu cơ sở cyan, magenta và yellow dành cho máy in màu. Mô hình CMY (phải) là bù của mô hình RGB (trái)
- Biểu đồ CMY thuộc loại mô hình trừ, trong khi RBG là cộng
- CMY-CMYK (Cyan-Magenta-Yellow-Black)
- Tổ hợp ánh sáng phosphor RGB trên màn hình và mực CMYK trên giấy được gọi là màu của đồ họa máy tính. Nhưng là hai vấn đề khác nhau hoàn toàn



Chuyển đổi giữa RGB và CMY

■ RGB -> CMY

```
void RGB2CMY(float R, float G, float B, float &C, float &M, float &Y)
```

```
{  
    C = 1 - R;  
    M = 1 - G;  
    Y = 1 - B;  
}
```

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

■ CMYK -> RGB

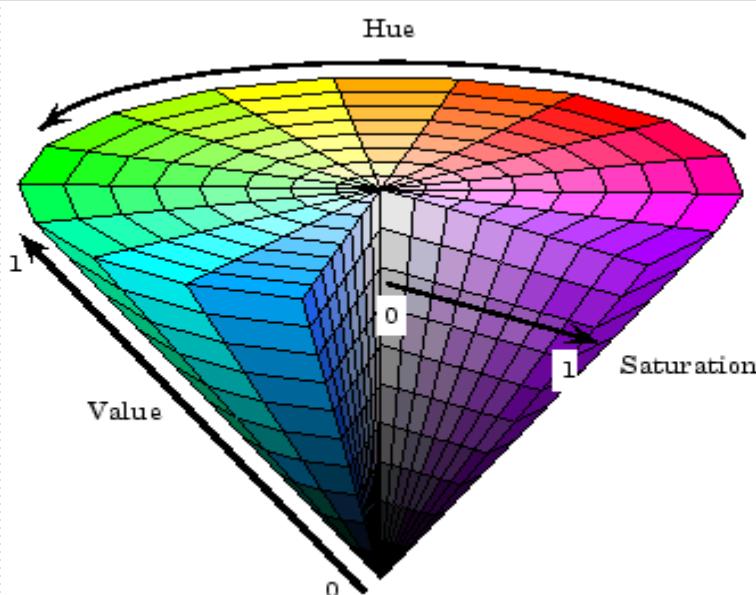
```
void RGB2CMYK(float R, float G, float B, float &C, float &M, float &Y, float &K)
```

```
{  
    RGB2CMY(R, G, B, C, M, Y);  
    K = min3(C, M, Y); // Cho lại giá trị min từ ba đối số  
    C = C - K;  
    M = M - K;  
    Y = Y - K;  
}
```

Mô hình màu HSV

- Mô hình RGB là màu mắt con người cảm nhận ánh sáng chứ ko phải màu mà mắt con người cảm nhận màu sắc.
- Mặc dù mô hình RGB biểu diễn chính xác cái mà màn hình cần hiển thị cho mắt người.
- RGB không phải là mô hình tốt để người sử dụng biểu diễn một màu cụ thể khi họ suy nghĩ (as artist).
 - Nếu người dùng cần biết giá trị RGB để biểu diễn màu tía thì họ rất khó tìm ra giá trị đó
- Do vậy nhiều giao diện chương trình người dùng sử dụng hệ thống màu HSV để xác định màu

Biểu diễn HSV



	RGB	HSV
Red	(1,0,0)	(0°,1,1)
Yellow	(1,1,0)	(60°,1,1)

Không gian HSV được mô tả bằng khối lập phương RGB quay trên đỉnh Black. H (Hue) là góc quay trực V (value) qua 2 đỉnh Black và White. Các giá trị biến thiên của H, S, V như sau :

H (Hue) chỉ sắc thái có giá trị từ 0 - 360 .

S (Saturation) chỉ độ bão hòa.

V (Value) có giá trị từ 0 - 1.

Chuyển đổi giữa RGB và HSV

$$H = \begin{cases} \left(\frac{G - B}{Max - Min} \right) \cdot \frac{360}{6} & if \ R = Max \\ \left(2 + \frac{B - R}{Max - Min} \right) \cdot \frac{360}{6} & if \ G = Max \\ \left(4 + \frac{R - G}{Max - Min} \right) \cdot \frac{360}{6} & if \ B = Max \\ 0 & if \ Max = Min \end{cases}$$
$$S = \frac{Max - Min}{Max}$$

$$V = Max$$

$$Max = \max(R, G, B)$$

$$Min = \min(R, G, B)$$

Không gian màu YIQ

- Mô hình không gian màu này được áp dụng cho National Television Standards Committee (NTSC) để phát sóng TV
- Mô hình này dựa trên thuộc tính của mắt người: nhạy cảm với sự thay đổi độ sáng (Luminance) hơn là sự thay đổi Hue và Saturation nghĩa là khả năng chúng ta phân biệt màu “không gian” yếu hơn khả năng phân biệt đơn sắc. Gợi ý rằng sử dụng nhiều bit hơn dùng để biểu diễn Y (độ chói) hơn là biểu diễn I(*in-phase*) và Q (*quadrature*).
- YIQ là nền tảng của nén ảnh JPEG

Chuyển đổi giữa RGB và YIQ

$$R, G, B, Y \in [0, 1], \quad I \in [-0.5957, 0.5957], \quad Q \in [-0.5226, 0.5226]$$

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.595716 & -0.274453 & -0.321263 \\ 0.211456 & -0.522591 & 0.311135 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.9563 & 0.6210 \\ 1 & -0.2721 & -0.6474 \\ 1 & -1.1070 & +1.7046 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

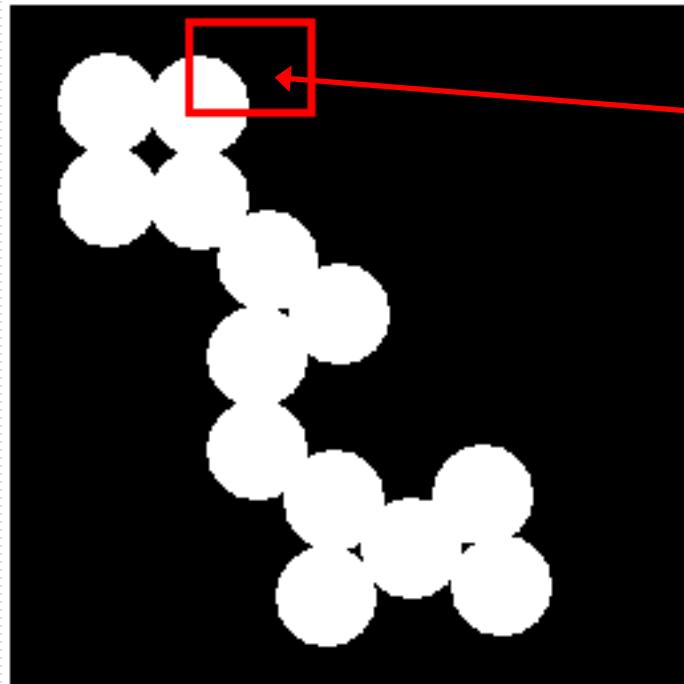
Các dạng biểu diễn ảnh

Một số khái niệm cơ bản

- Pixel: là đơn vị cơ bản mang màu sắc của ảnh hiển thị theo phương thức raster
 - Ảnh số: là ma trận hai chiều mà mỗi phần tử là một pixel
 - Mỗi pixel của ảnh được đặc trưng bởi tọa độ và màu sắc (x, y, V)
-

Số hóa bằng ảnh đen trắng

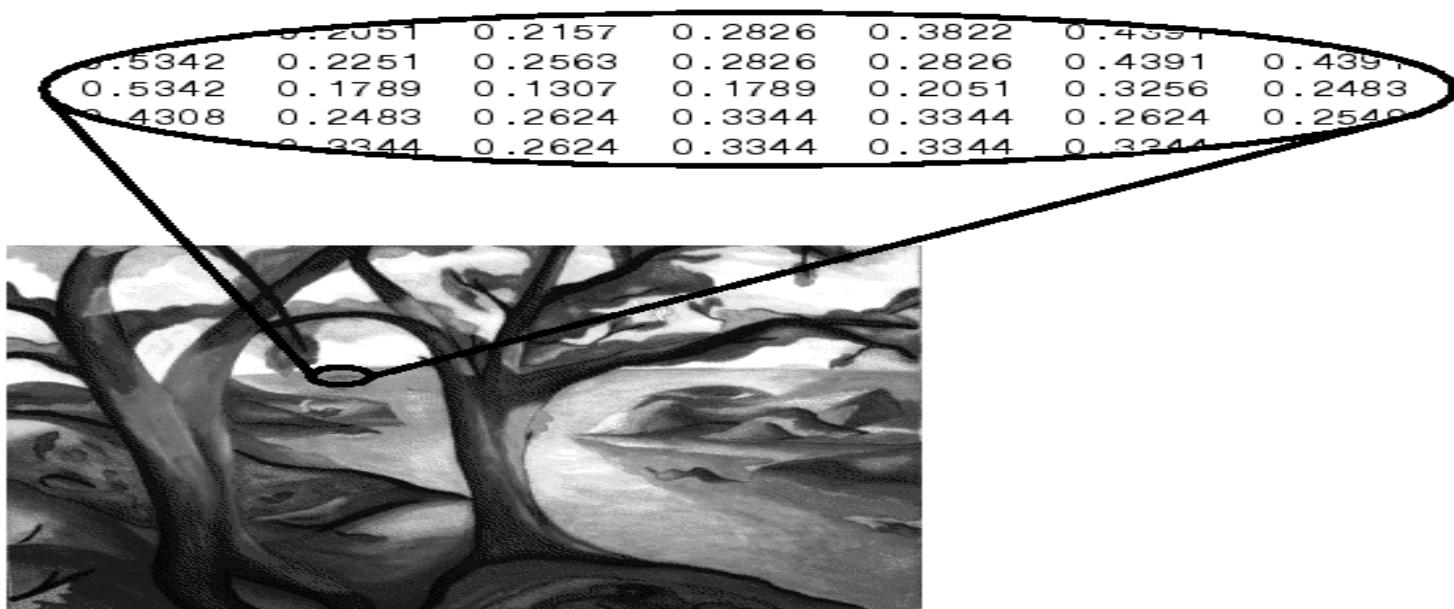
- ☐ Khi V là một bit (0 hoặc 1) thì ảnh thuộc lớp ảnh đen trắng (BW)



0000000000000000
1111000000000000
1111100000000000
1111000000000000
1110000000000000

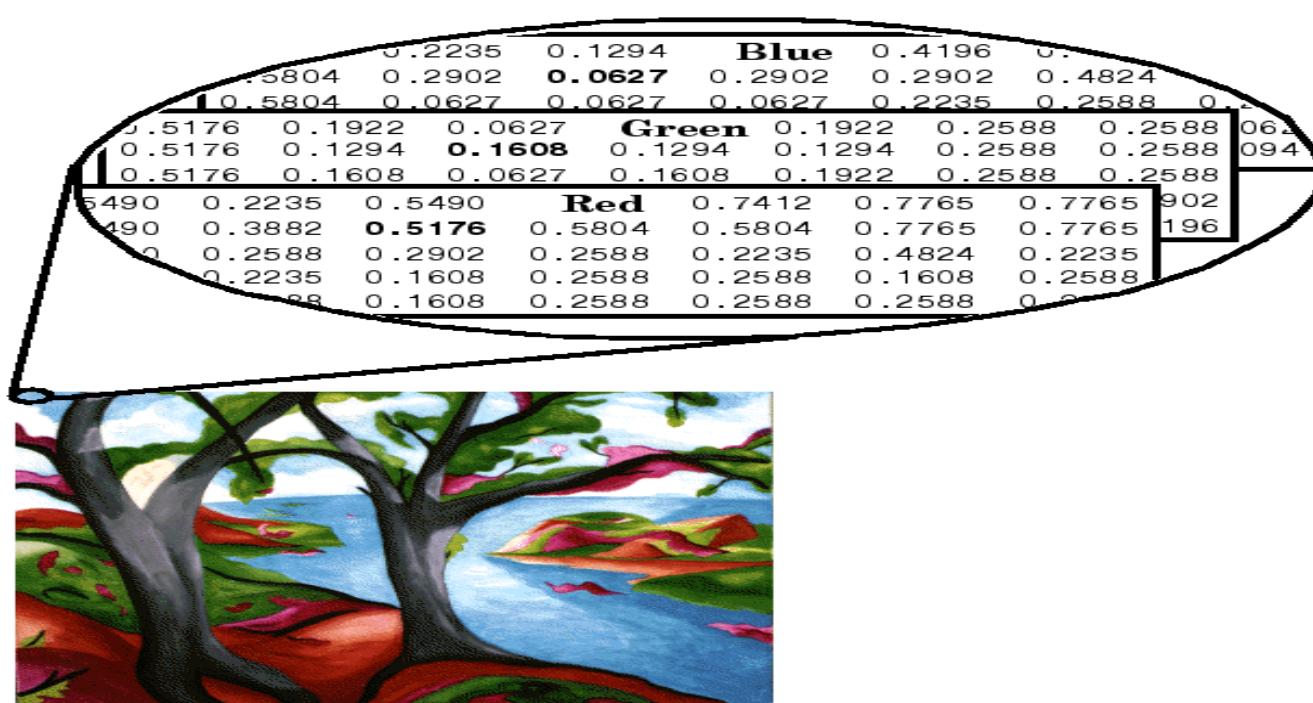
Số hóa bằng ảnh xám

- Khi V là một giá trị thực (float) (trong khoảng 0 đến 1) thì ảnh thuộc lớp ảnh xám (gray image), trong đó V đặc trưng cho độ sáng của pixel (gray intensity)



Số hóa bằng ảnh màu

- Khi V là vector giá trị thực (V_r, V_g, V_b) thì ảnh thuộc lớp ảnh màu (true color image), trong đó V đặc trưng cho màu sắc của pixel



Sinh viên cần nắm bắt

- Khái niệm màu
- Định lý Gassman
- Các không gian màu
- Các hình thức biểu diễn ảnh số

```
byte* B = rgbValues++;
byte* G = rgbValues++;
byte* R = rgbValues++;
byte grey = (byte)((*R) * 0.21 + (*G) * 0.59 +
(*B) * 0.2);
*R = *G = *B = grey;
```

Các phép biến đổi hai chiều (2-dimension)

Computer Graphics

Biến đổi hai chiều

- Các phép toán cơ sở với ma trận
- Các phép biến đổi 2D cơ sở
- Biến đổi 2D gộp

Các phép toán cơ sở với ma trận

□ Cộng, trừ ma trận

- chỉ thực hiện cho hai ma trận cùng cỡ

$$[A(m, n)] + [B(m, n)] = [C(m, n)]$$

$$[c_{ij}] = [a_{ij}] + [b_{ij}]$$

□ Nhân hai ma trận

- Ma trận cỡ $n_1 \times m_1$ và ma trận bậc $n_2 \times m_2$ nhân được với nhau nếu $m_1 = n_2$

$$[A(m, n)][B(n, p)] = [C(m, p)]$$

$$c_{jk} = \sum_{i=1}^n a_{ji} b_{ik}$$

Các phép toán cơ sở với ma trận

- Nghịch đảo ma trận vuông
 - Không có phép chia ma trận
 - Nếu $[A][X]=[Y]$ thì $[X]=[A]^{-1}[Y]$ trong đó $[A]^{-1}$ là ma trận nghịch đảo của ma trận vuông $[A]$
 - $[A][A]^{-1}=[I]$ trong đó $[I]$ là ma trận đơn vị
- $$[A]^{-1} = \frac{1}{\det[A]} \|A\|^T$$
- Tính $\|A\|$: thay các phần tử của $[A]$ bằng các phần phụ đại số của nó
- Phần phụ đại số của phần tử (a_{ij}) là: $(-1)^{i+j}\det[M_{ij}]$
- $[M_{ij}]$ được tạo ra nhờ xóa hàng i , cột j của $[A]$

Ứng dụng biến đổi

□ Mô hình hóa

- Định vị và thay đổi kích thước các phần của đối tượng phức tạp

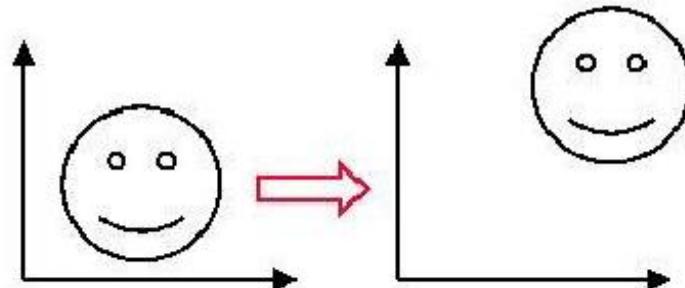
□ Quan sát

- Định vị và quan sát camera ảo

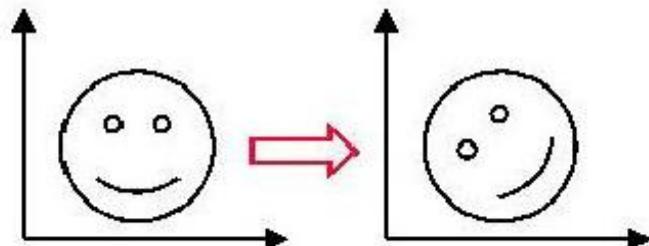
□ Animation

- Xác định đối tượng chuyển động và thay đổi theo thời gian như thế nào

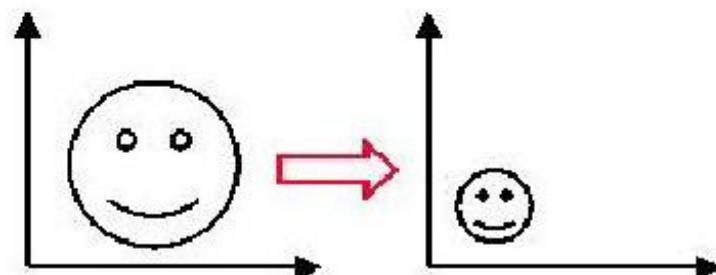
Các ví dụ biến đổi 2D



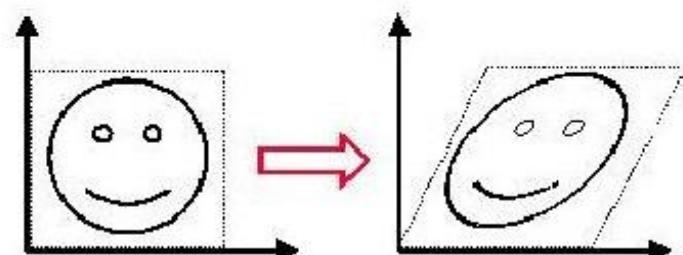
translation



rotation

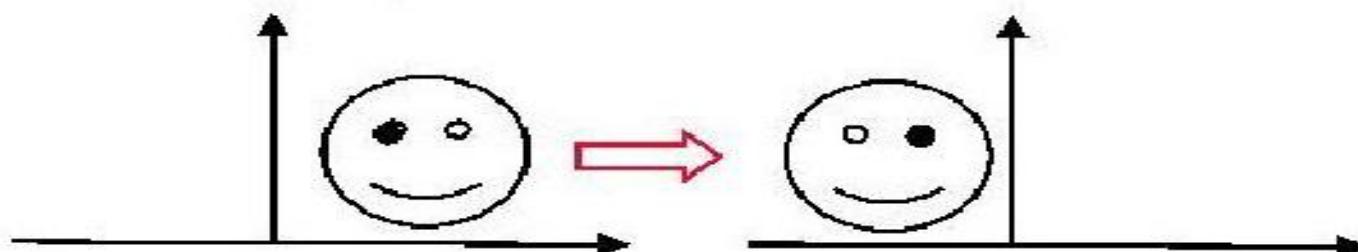


scaling

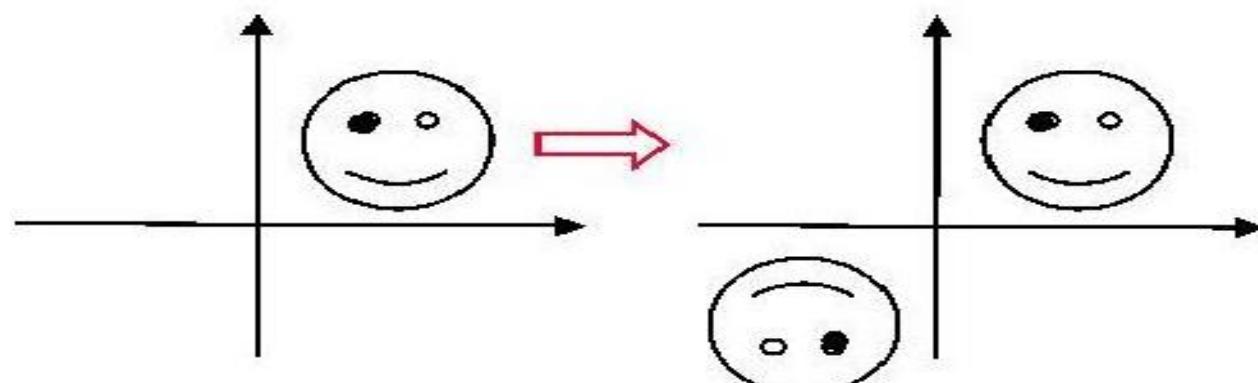


shear

Các ví dụ biến đổi 2D



reflection with respect to the y-axis



reflection with respect to the origin

Các loại biến đổi

- Biến đổi tuyến tính
 - Các đường thẳng giữ nguyên là đường thẳng
 - Các ví dụ trên
- Biến đổi Affine
 - Các đường thẳng song song giữ nguyên song song
 - Các ví dụ trên là Affine. Ví dụ non-affine: chiếu viễn cảnh
- Biến đổi trực giao
 - Bảo toàn khoảng cách, dịch chuyển đối tượng như khôi rắn
- Xoay, dịch chuyển, phản chiếu là affine
 - Bất kỳ biến đổi affine nào cũng có thể viết như sau

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix}$$

$$P' = P \cdot A$$

Các phép biến đổi cơ sở

□ Tịnh tiến (translation)

$$x' = x + Tx$$

$$y' = y + Ty$$

(Tx, Ty) là vecto tịnh tiến

■ Định nghĩa: $P = [x \ y]$, $P' = [x' \ y']$, $T = [Tx \ Ty]$

$$[x' \ y'] = [x \ y] + [Tx \ Ty]$$

□ Co dãn (scaling)

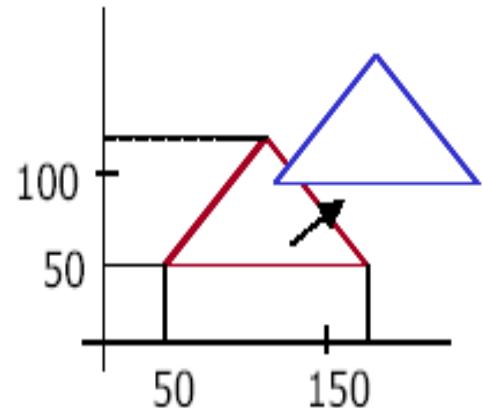
$$x' = x \cdot S_x$$

S_x là thừa số co dãn chiều x

$$y' = y \cdot S_y$$

S_y là thừa số co dãn chiều y.

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$



Các phép biến đổi cơ sở

□ Xoay hình (Rotation)

$$X = R \cos \phi$$

$$Y = R \sin \phi$$

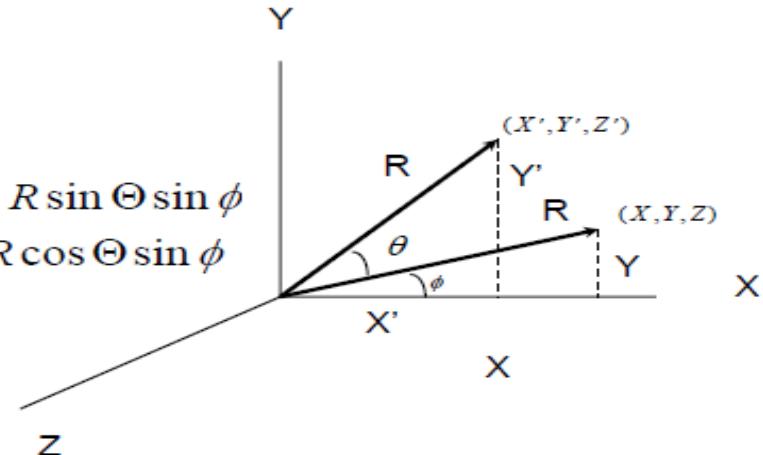
$$X' = R \cos(\Theta + \phi) = R \cos \Theta \cos \phi - R \sin \Theta \sin \phi$$

$$Y' = R \sin(\Theta + \phi) = R \sin \Theta \cos \phi + R \cos \Theta \sin \phi$$

$$X' = X \cos \Theta - Y \sin \Theta$$

$$Y' = X \sin \Theta + Y \cos \Theta$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} \cos \Theta & -\sin \Theta & 0 \\ \sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$



Tọa độ thuần nhất

- Các biến đổi cơ sở có cách xử lý khác nhau
 - $P' = P + T$ (tịnh tiến); $P' = P.S$ (co dãn); $P' = P.R$ (xoay)
- Thực tế: nhu cầu tổ hợp các chuyển đổi cơ sở
 - Cần cách xử lý nhất quán để dễ dàng tổ hợp
 - Sử dụng hệ thống tọa độ thuần nhất (Homogeneous Coordinates)

Tọa độ thuần nhất?

- Mục tiêu ban đầu của hệ tọa độ thuần nhất là để biểu diễn khái niệm vô hạn
 - Không thể biểu diễn giá trị vô hạn trong hệ tọa độ Đẽ các
 - Giả sử với hai số thực w và a
 - Giá trị vô hạn được biểu diễn bởi $v=a/w$
 - Khi $w \rightarrow 0$ thì a/w tiến tới vô hạn: cặp (a, w) biểu diễn khái niệm vô hạn; cặp $(a, 0)$ biểu diễn giá trị vô hạn.
- Áp dụng hệ tọa độ xy trong mặt phẳng
 $f(x, y) = 0$
 $f(x/w, y/w) = 0$
 - Nếu $f(x, y) = 0$ là đa thức bậc n thì nhân nó với w^n để loại bỏ mẫu

Ví dụ về hệ tọa độ thuần nhất

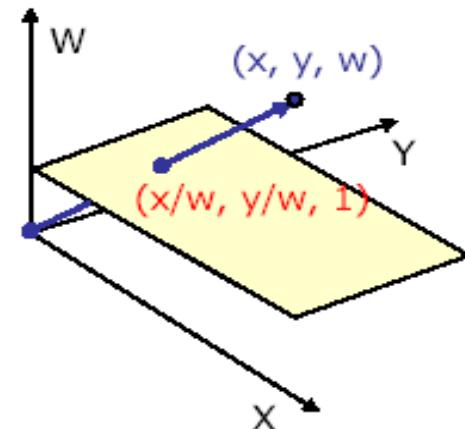
- Phương trình bậc nhất (đường thẳng):
 $Ax + By + C = 0$; thay x, y ta có:
 $A(x/w) + B(y/w) + C = 0$; nhân với w ta có:
 $Ax + By + Cw = 0$
- Đa thức bậc 2:
 $Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Ey + F = 0$
Sau khi thay thế và nhân với w^2 ta có:
 $Ax^2 + 2Bxy + Cy^2 + 2Dxw + 2Eyw + Fw^2 = 0$
- Nhận xét:
 - Các phần tử trong đa thức đều có bậc như nhau
 - Đa thức bậc n thì các thành phần của nó đều có bậc n
 - Cho trước đa thức bậc n , sau khi bổ sung w thì mọi thành phần đều có bậc n -> gọi nó là đa thức thuần nhất và tọa độ (x, y, w) là tọa độ thuần nhất.

Tọa độ thuần nhất

□ Diễn giải hình học

- Cho trước tọa độ thuần nhất (x, y, w) của điểm trong mặt phẳng xy . (x, y, w) là điểm trong không gian xyw .
- Đoạn thẳng nối điểm (x, y, w) với gốc tọa độ trong không gian 3D sẽ cắt mặt phẳng $w=1$ tại $(x/w, y/w, 1)$
- Điểm đồng nhất 2D được xem như điểm trong không gian 3D và chiếu điểm 3D vào mặt phẳng $w=1$

- Bất kỳ biến đổi tuyến tính nào cũng có thể biểu diễn dưới dạng ma trận trong hệ thống tọa độ thuần nhất.



Từ đồng nhất sang 2D:

$$[x, y, w] \rightarrow [x/w, y/w]$$

Kết quả duy nhất

Từ 2D sang đồng nhất:

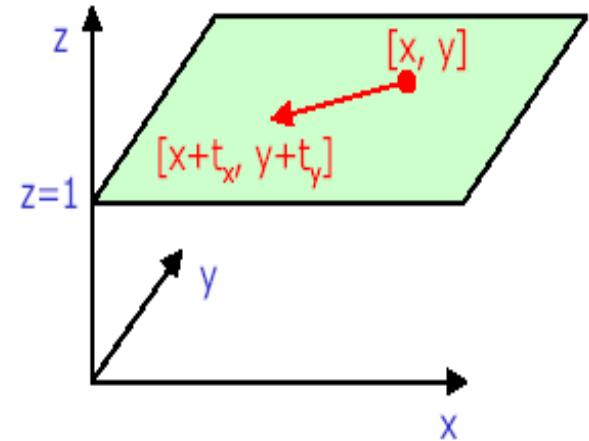
$$[x, y] \rightarrow [kx, ky, k]$$
$$k \neq 0$$

Ma trận biến đổi 2D

- Biểu diễn tọa độ 2D $[x, y]$ trong hệ tọa độ thuần nhất là bộ ba $[x \ y \ 1]$
 - Các điểm là vector hàng 3 phần tử
 - Ma trận biến đổi có kích thước 3×3
- Dịch chuyển

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T(T_x, T_y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$$



Dịch chuyển:

$$x' = x + t_x = x + t_x \cdot 1$$

$$y' = y + t_y = y + t_y \cdot 1$$

$$w' = 1$$

Ma trận biến đổi 2D

■ Co dãn

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$S(Sx, Sy) = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

■ Xoay

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

■ Biến đổi affine tổng quát

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{bmatrix}$$
$$x' = ax + by + c$$
$$y' = dx + ey + f$$

Chuyển đổi gộp

- Giải pháp
 - Tính ma trận kết quả các chuyển đổi thành phần trong chuyển đổi gộp
- Dịch chuyển 2 lần

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x1} & T_{y1} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x2} & T_{y2} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x1} + T_{x2} & T_{y1} + T_{y2} & 1 \end{bmatrix}$$

$T(Tx_1, Ty_1) \cdot T(Tx_2, Ty_2) = T(Tx_1 + Tx_2, Ty_1 + Ty_2)$

- Co dãn hai lần
- $$S(Sx_1, Sy_1) \cdot S(Sx_2, Sy_2) = S(Sx_1 \cdot Sx_2, Sy_1 \cdot Sy_2)$$

- Xoay hai lần
- $$R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$$

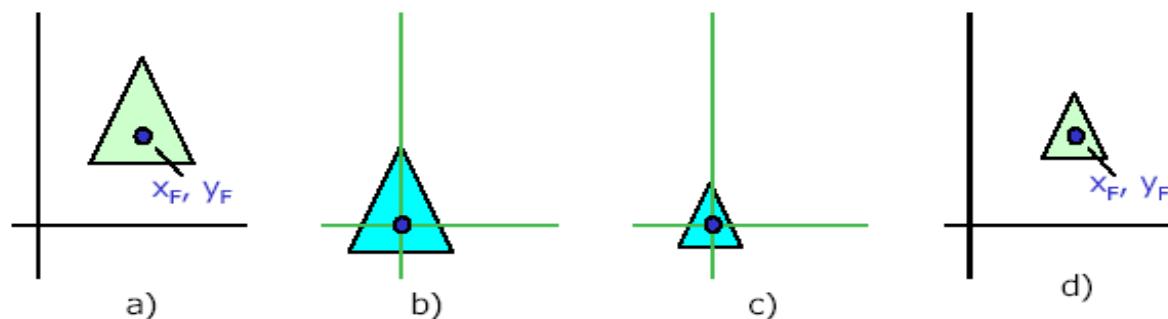
Co dãn đối tượng theo điểm cố định

□ Vấn đề

- Cho trước ΔABC , tọa độ chốt (x_F, y_F) và tỷ lệ co dãn (a)
- Thực hiện biến đổi để có kết quả (d)

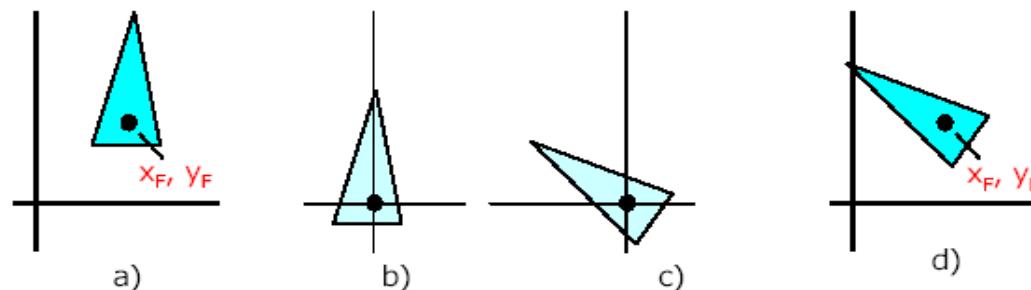
□ Các bước thực hiện

- Dịch đối tượng sao cho điểm chốt trùng với gốc tọa độ
- Thực hiện co dãn theo tỷ lệ cho trước
- Dịch ngược đối tượng sao cho điểm chốt về vị trí ban đầu



Xoay đối tượng quanh điểm cố định

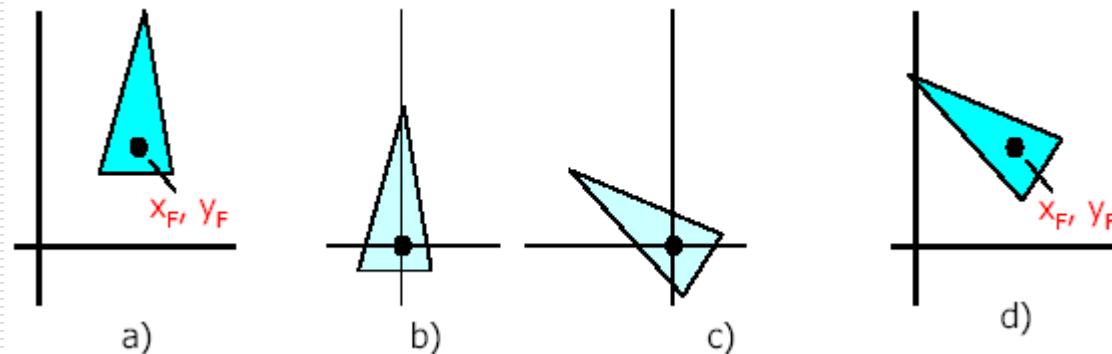
- Vấn đề
 - Cho trước ΔABC , tọa độ chốt (x_F, y_F) và góc xoay (a)
 - Thực hiện biến đổi để có kết quả (d)
- Các bước thực hiện
 - Dịch đối tượng sao cho điểm chốt trùng gốc tọa độ
 - Thực hiện xoay theo góc cho trước
 - Dịch ngược đối tượng sao cho điểm chốt về vị trí ban đầu



Xoay đối tượng quanh điểm cố định

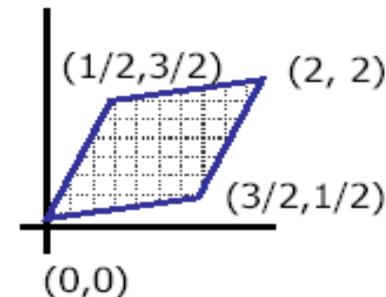
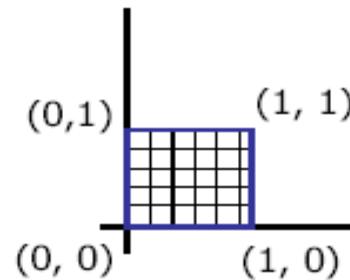
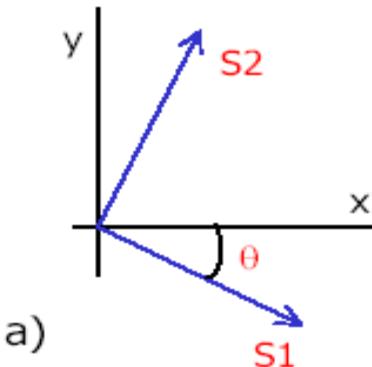
□ Ma trận chuyển đổi được tính

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_R & -y_R & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_R & y_R & 1 \end{bmatrix}$$
$$= \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ (1 - \cos \theta)x_R + y_R \cdot \sin \theta & (1 - \cos \theta)y_R - x_R \cdot \sin \theta & 1 \end{bmatrix}$$



Co dãn theo hướng tùy ý

- Ma trận biến đổi co dãn cơ bản
 - tỷ lệ S_x và S_y áp dụng cho co dãn theo chiều trực x và y
- Co dãn theo hướng tùy ý
 - Thực hiện chuyển đổi gộp: xoay và co dãn
- Vấn đề
 - Cho trước hình vuông ABCD, hãy co dãn nó theo hướng trên hình a) và theo tỷ lệ S_1, S_2



Co dãn theo hướng tùy ý

□ Giải pháp

- Xoay hướng S₁, S₂ sao cho trùng với trục x và trục y (góc xoay θ)
- Áp dụng biến đổi co dãn theo tỷ lệ S₁, S₂
- Xoay trả lại hướng ban đầu

□ Ma trận tổ hợp

$$\begin{bmatrix} S_1 \cdot \cos^2 \theta + S_2 \cdot \sin^2 \theta & (-S_1 + S_2) \sin \theta \cos \theta & 0 \\ (-S_1 + S_2) \sin \theta \cos \theta & S_1 \cdot \sin^2 \theta + S_2 \cdot \cos^2 \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Một số biến đổi cơ sở khác

□ Phản chiếu

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

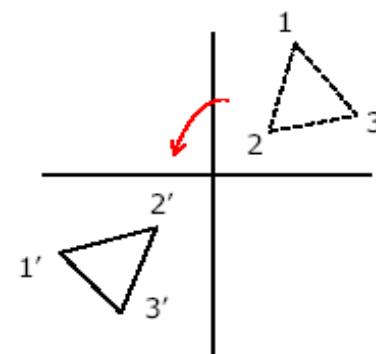
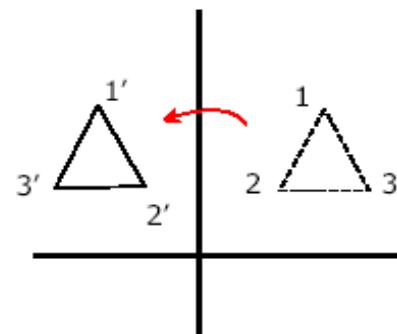
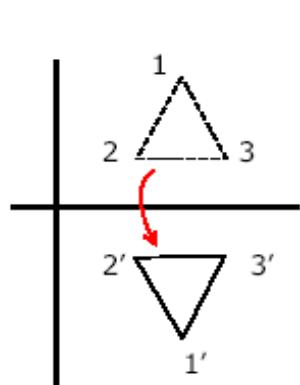
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Phản chiếu qua trục x

Phản chiếu qua trục y

Phản chiếu qua trục z



Tóm tắt

- Tóm tắt các phép toán ma trận ứng dụng trong đồ họa máy tính
- Các biến đổi 2D cơ sở
- hệ thống tọa độ thuần nhất
- Biểu diễn các biến đổi cơ sở 2D bằng ma trận
- Các biến đổi 2D khác

Problems

1. Hãy tìm ma trận biến đổi để có đối tượng phản chiếu qua $y=x$ và $y=-x$.
2. Cho $\Delta A(3, 1), B(1, 3), C(3, 3)$:
 - Hãy xác định tọa độ mới của đỉnh tam giác sau khi xoay một góc 90^0 ngược chiều kim đồng hồ xung quanh điểm $P(2, 2)$
 - Phóng to tam giác lên hai lần, giữ nguyên vị trí của điểm C. Tính tọa độ các đỉnh tam giác sau khi biến hình.
3. Tìm ma trận biến đổi trong phép đối xứng qua đường thẳng nằm nghiêng có độ nghiêng m và đi qua điểm $(0, c)$.
4. Viết các hàm bằng Pascal (hoặc C) thực hiện dịch, co dãn xoay 2D

Kiểm tra

1. Cho $\Delta A(3, 1), B(1, 3), C(3, 3)$:

Sử dụng các phép biến đổi Affine cơ bản tìm tam giác đối xứng với tam giác ABC qua đường thẳng $y=x+2$

CÁC THUẬT TOÁN CƠ SỞ VẼ ĐỒ HỌA

Computer Graphics

Phần 1

Các thuật toán vẽ đường thẳng

Đặt vấn đề

- Đường thẳng trong ĐHMT được biểu diễn như tập hợp các điểm rời rạc (pixel) mà tọa độ của chúng tuân theo phương trình ĐT:

➤ Dạng tổng quát

$$Ax + By + C = 0$$

➤ Dạng tham số

$$y = mx + b$$

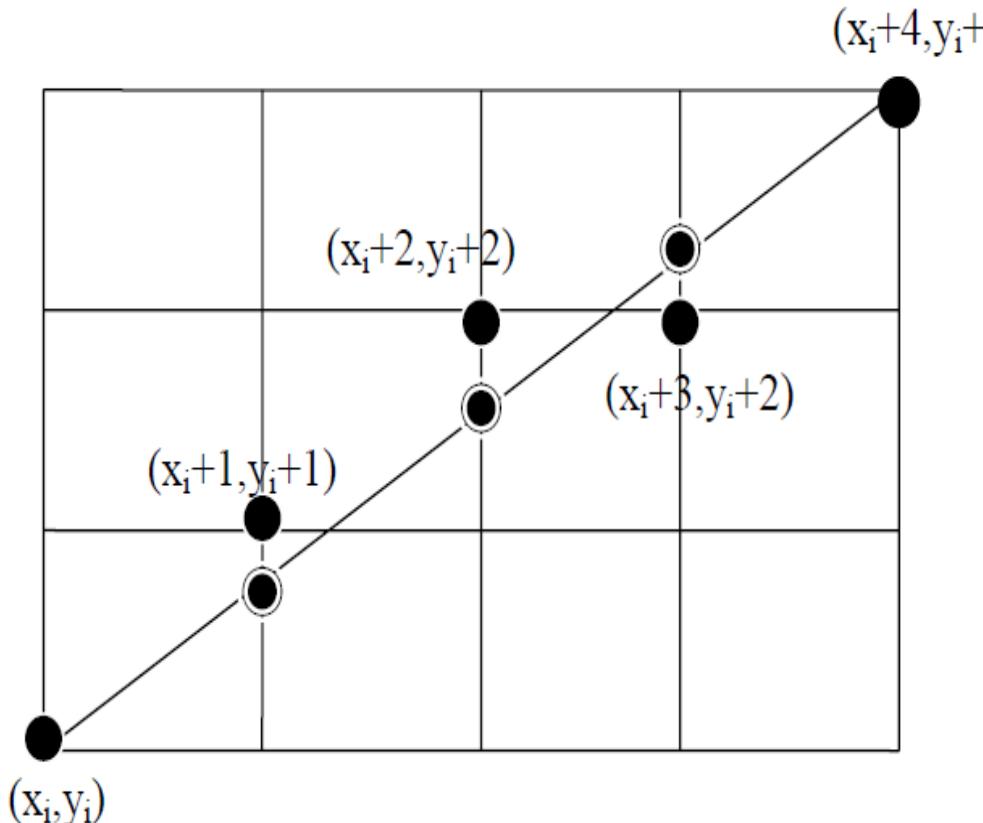
➤ Dạng chính tắc

$$x, y \in Z$$

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

-
- ❑ Thực chất việc vẽ đường thẳng là việc định màu cho các pixel rời rạc.
 - ❑ Do tọa độ pixel chỉ là số nguyên nên khái niệm “thẳng” chỉ là gần đúng.
 - ❑ Chọn tọa độ gần đúng thế nào để đảm bảo chất lượng hình vẽ và tốc độ xử lý?
-

Mô tả việc xác định tọa độ



$$x_{i+1} = x_i + 1$$

$$y_{i+1} = \begin{bmatrix} y_i \\ y_i + 1 \end{bmatrix}$$

Cách biểu diễn thông tin về đoạn thẳng

- Input: điểm đầu (x_1, y_1) , điểm cuối (x_2, y_2) , màu tô color C.
- Phương trình đoạn thẳng đi qua hai điểm

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

$$a = \frac{y_2 - y_1}{x_2 - x_1} \qquad \qquad b = y_1 - ax_1$$

$$y = ax + b$$

Cách biểu diễn thông tin về đoạn thẳng

- Chuyển đổi đường quét (Rasterization)
- Biến đổi đường liên tục thành rời rạc (Sampling)
- Yêu cầu chất lượng đường vẽ
 - Hình dạng liên tục
 - Độ dày và độ sáng đều
 - Các pixel gần đường “lý tưởng” được hiển thị
 - Vẽ nhanh

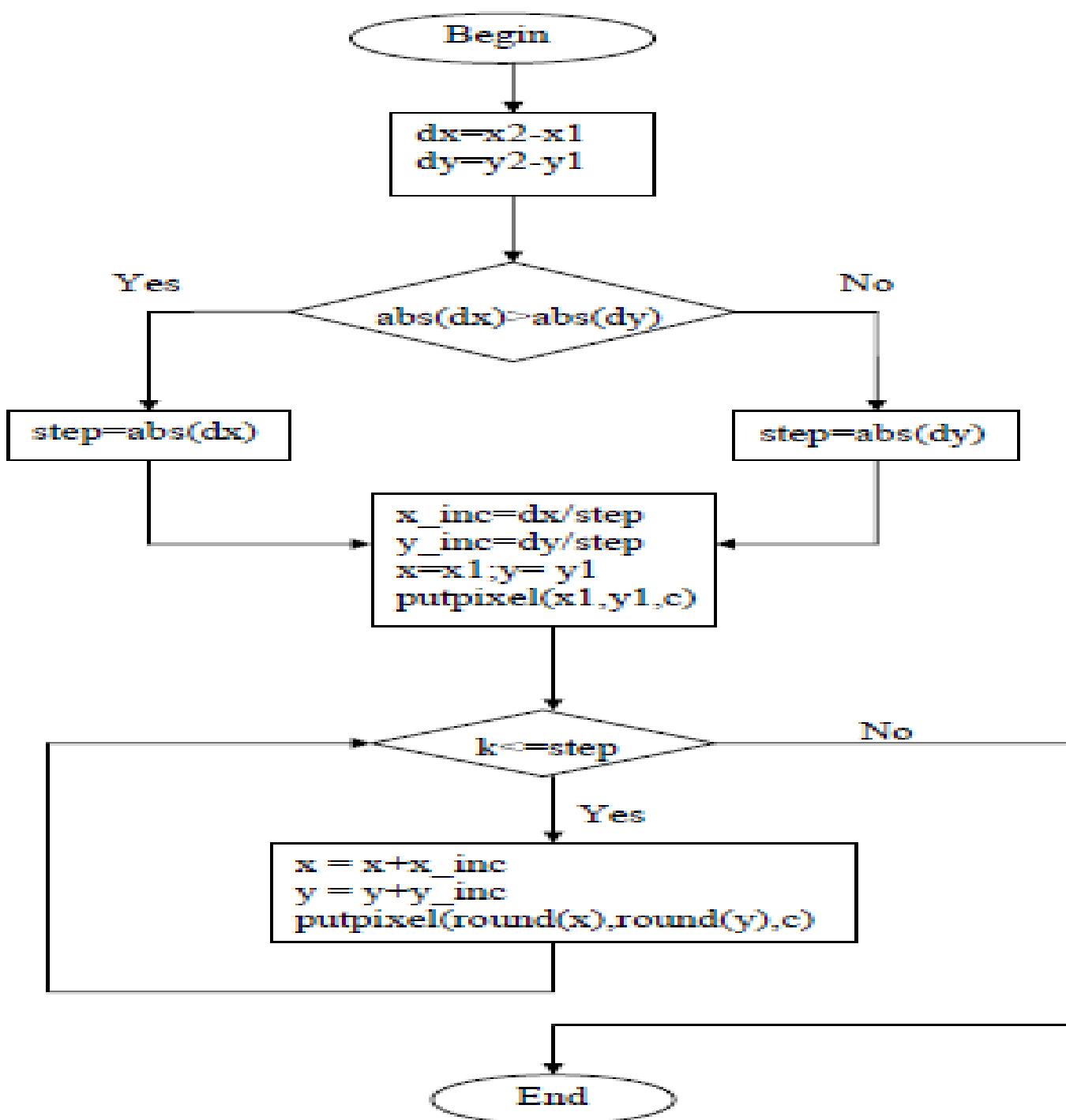
Những thuật toán vẽ đoạn thẳng

- Thuật toán DDA (Digital Defferencial Analyzer) hay thuật toán tăng dần
- Thuật toán Bresenham
- Thuật toán trung điểm

Thuật toán DDA

- DDA tính toán các điểm vẽ dọc theo đường thẳng dựa vào hệ số góc của phương trình đường thẳng dạng $y=mx+b$.
- Cho giá trị bước nhảy trên một trục tính giá trị bước nhảy trên trục kia theo phương trình ĐT
- Với hệ số góc trong khoảng $[0, 1]$:
 - $dy = a \cdot dx$
 - Nếu $dx = 1$ thì $y_{i+1} = y_i + a$
 - Làm tròn số vì a bất kỳ

Ý tưởng thuật toán: Với mỗi bước hãy tính số gia cơ sở bước trước đó.



Thuật toán DDA

```
#define ROUND(a) ((int)(a+0.5))
void lineDDA (int xa, int ya, int xb, int yb)
{
    int dx = xb - xa, dy = yb - ya, steps, k;
    float xIncrement, yIncrement, x = xa, y = ya;
    if (abs (dx) > abs (dy)) steps = abs (dx);
    else steps = abs (dy);
    xIncrement = dx / (float) steps;
    yIncrement = dy / (float) steps;
    PutPixel (ROUND(x), ROUND(y));
    for (k=0; k<steps; k++) {
        x += xIncrement; y += yIncrement;
        PutPixel (ROUND(x), ROUND(y));}
}
```

Thuật toán DDA

□ Nhận xét thuật toán DDA

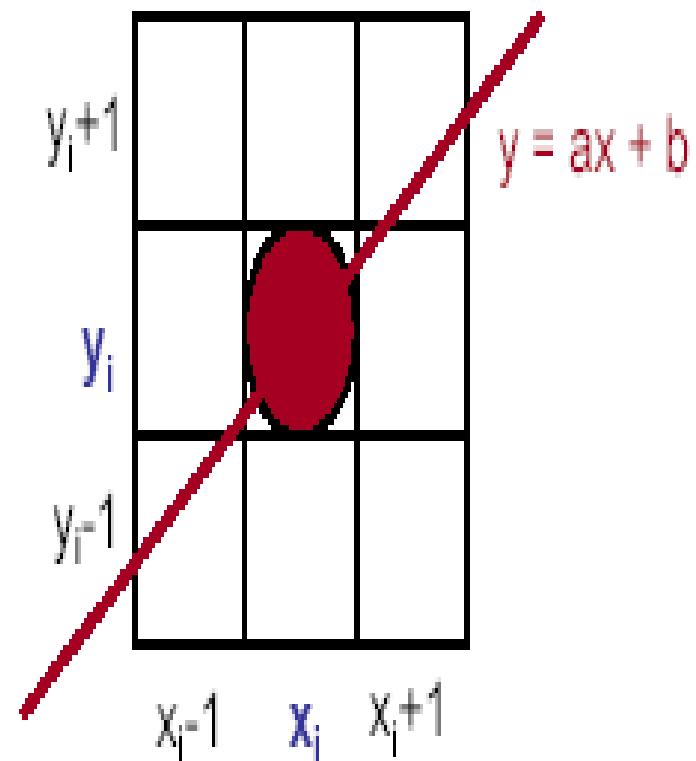
- Không có phép nhân
- Có phép chia và làm tròn số -> chậm

□ Quy tắc tổng quát khi vẽ đồ họa

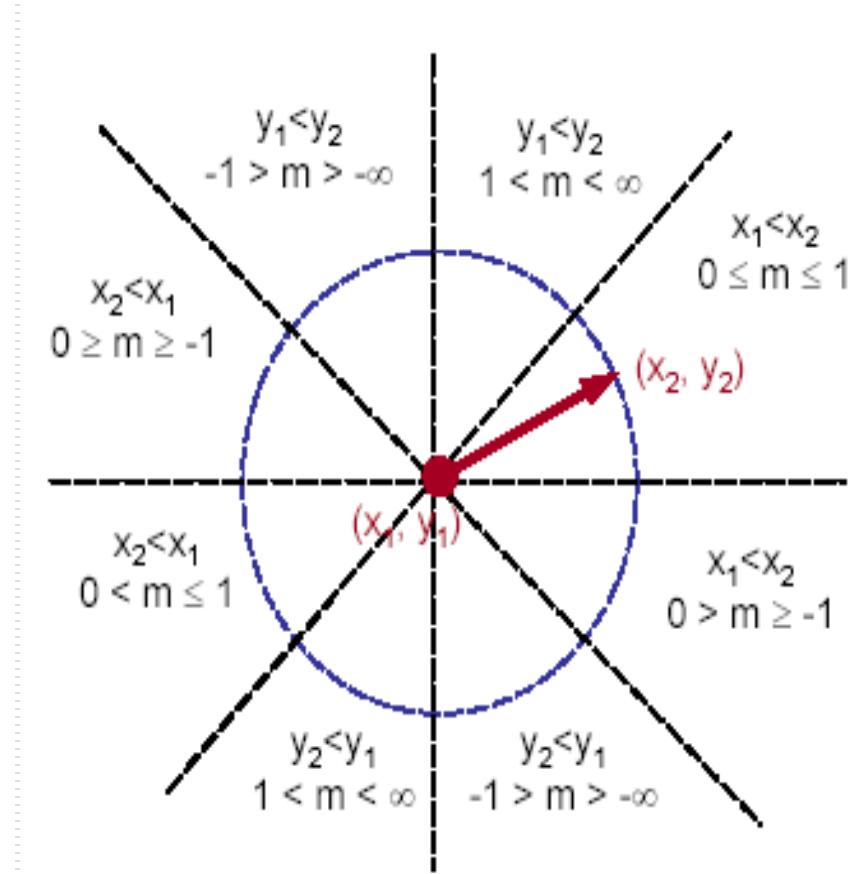
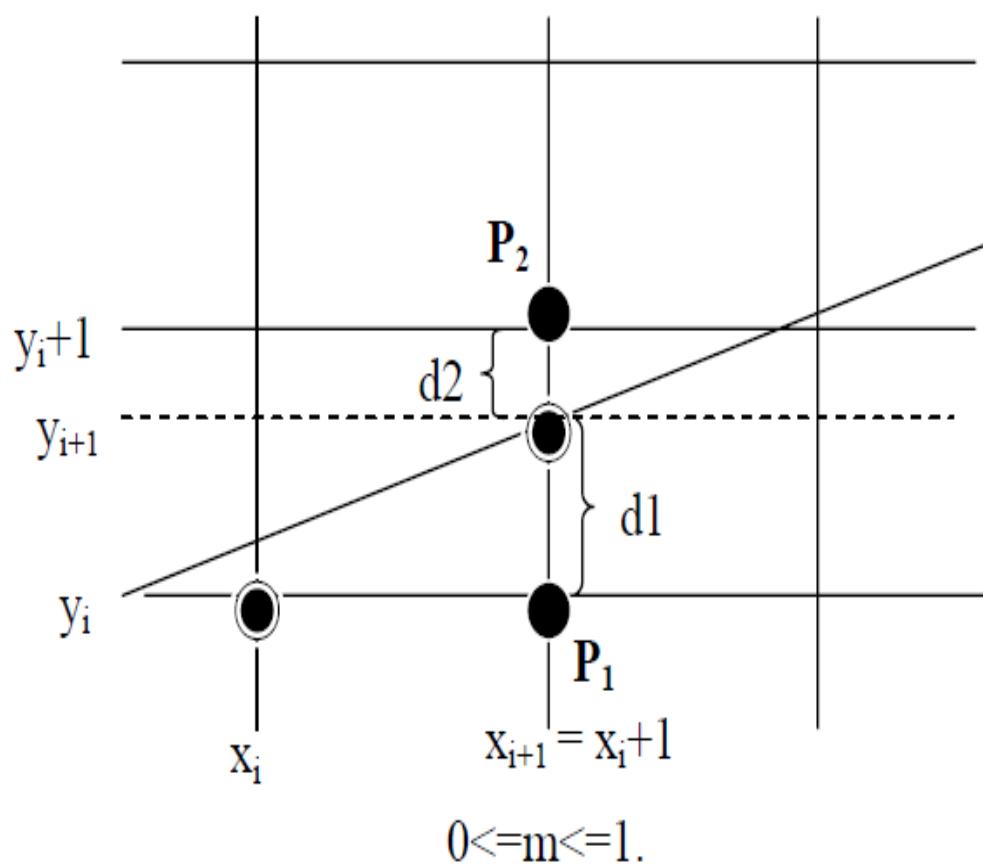
- Cộng và trừ nhanh hơn nhân
- Nhân nhanh hơn chia
- Sử dụng bảng để đánh giá hàm rời rạc nhanh hơn tính toán
- Tính toán số nguyên nhanh hơn số thực
- Tránh các tính toán không cần thiết nhờ nhận ra các trường hợp đặc biệt của đường vẽ.

Thuật toán Bresenham vẽ line

- Giả sử vừa vẽ điểm tại (x_i, y_i) , bây giờ phải xác định điểm sẽ vẽ là một trong 8 pixel liền kề: $(x_i+1, y_i), (x_i-1, y_i), (x_i, y_i-1), (x_i, y_i+1)...$
- Hình dạng đoạn thẳng phụ thuộc vào các giá trị dx và dy
 - $dx=0 \rightarrow$ đ/thẳng song song với trục y
 - $dy=0 \rightarrow$ đ/thẳng song song với trục x
 - $dx>0 \rightarrow$ tọa độ x biến thiên tăng dần
 - $dx<0 \rightarrow$ tọa độ x biến thiên giảm dần
 - Xét tương tự đối với dy
 - Nếu $\text{abs}(dx)>\text{abs}(dy)$: $y=f(x)$
 - Nếu $\text{abs}(dx)<\text{abs}(dy)$: $x=f(y)$

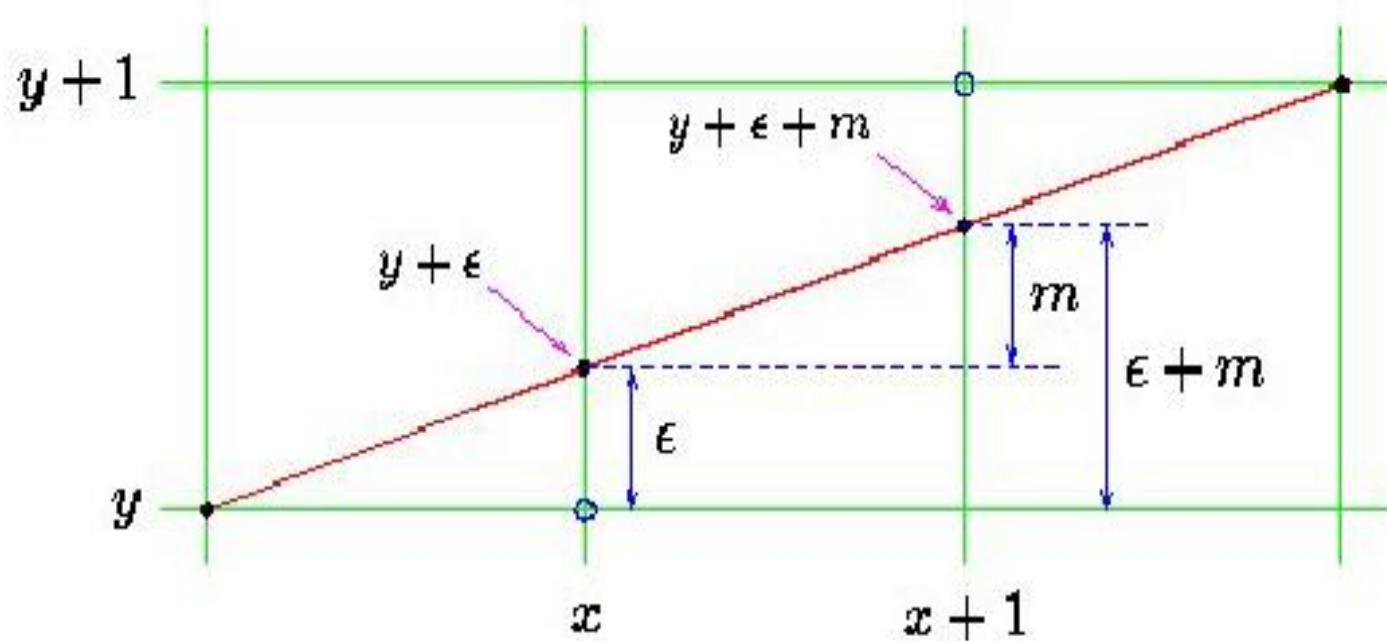


Thuật toán Bresenham



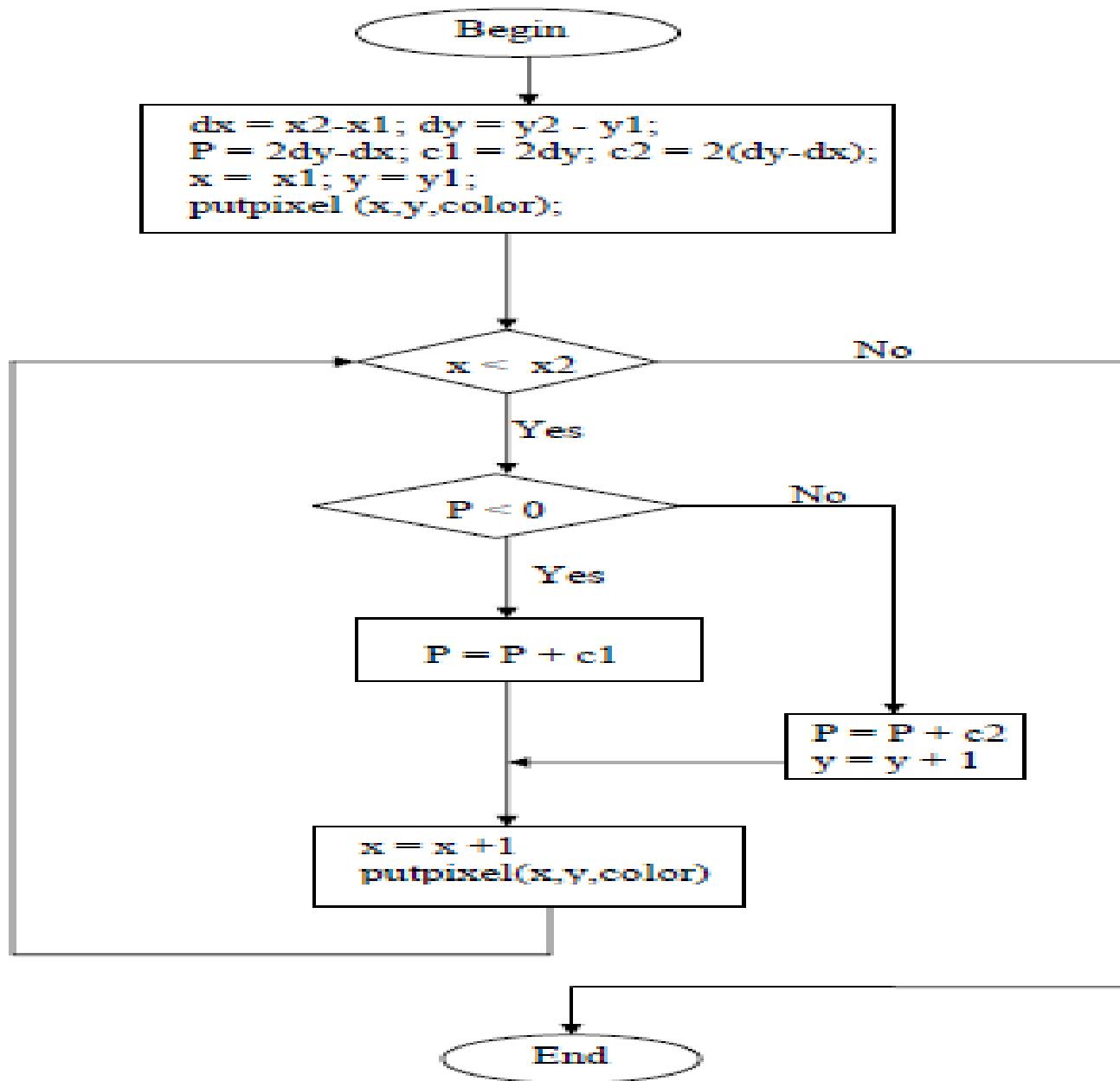
Thuật toán Bresenham

- Xét đoạn thẳng có hệ số góc $0 \leq m \leq 1$.
- Điểm vừa chọn là $(x, y) \rightarrow$ điểm tiếp theo vẽ là $(x+1, y)$ hoặc $(x+1, y+1)$.



Thuật toán Bresenham

- Nếu đã chọn (x, y) , điểm tiếp theo sẽ là:
 - độ lệch so với đường thẳng toán học là ϵ , có giá trị từ -0.5 đến 0.5
 - Nếu $\epsilon + m < 0.5$: chọn $(x+1, y)$, ngược lại chọn $(x+1, y+1)$
- Để thực hiện lặp: ghi kết quả lỗi này vào ϵ , tiếp tục xét cho điểm tại $x+2$.
- Xác định độ lệch tiếp theo
 - Nếu chọn $(x+1, y)$: $\text{new}\epsilon = (y + \epsilon + m) - y = \epsilon + m$
 - Nếu chọn $(x+1, y+1)$: $\text{new}\epsilon = (y + \epsilon + m) - (y+1) = \epsilon + m - 1$



Thuật toán Bresenham

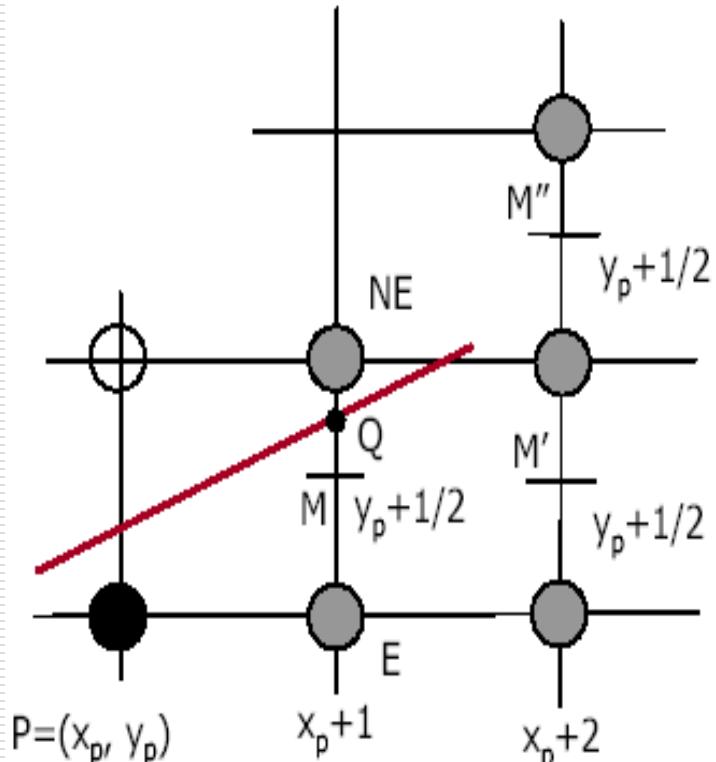
```
void lineBres (int xa, int ya, int xb, int yb)
{
    int dx = abs (xa - xb), dy = abs (ya - yb);
    int p = 2 * dy - dx;
    int twoDy = 2 * dy, twoDyDx = 2 * (dy - dx); int x, y, xEnd;
    if (xa > xb) {
        x = xb;y = yb;xEnd = xa;
    }
    else {x = xa;y = ya;xEnd = xb;}
    PutPixel (x, y);
    while (x < xEnd) {
        x++;
        if (p < 0)
            p += twoDy;
        else { y++; p += twoDyDx;}
        PutPixel (x, y);
    }
}
```

Thuật toán Bresenham vẽ line

- Thuật toán chỉ tính toán với số nguyên
- Nhân 2 -> dịch trái
- Chú ý cài đặt vẽ đoạn thẳng với hệ số góc bất kỳ

Thuật toán trung điểm vẽ line

- Pitteway công bố năm 1967, Van Aken cải tiến 1984.
- Giả sử ta đã chọn P để vẽ, xác định pixel tiếp theo tại E hay NE
 - Giao của đường thẳng với $x = x_p + 1$ tại Q, M là trung điểm của NE và E.
- Ý tưởng:
M nằm phía nào của đường thẳng, nếu M phía trên đường thẳng thì chọn E, ngược lại chọn NE.
- Nhiệm vụ: xác định M ở đâu



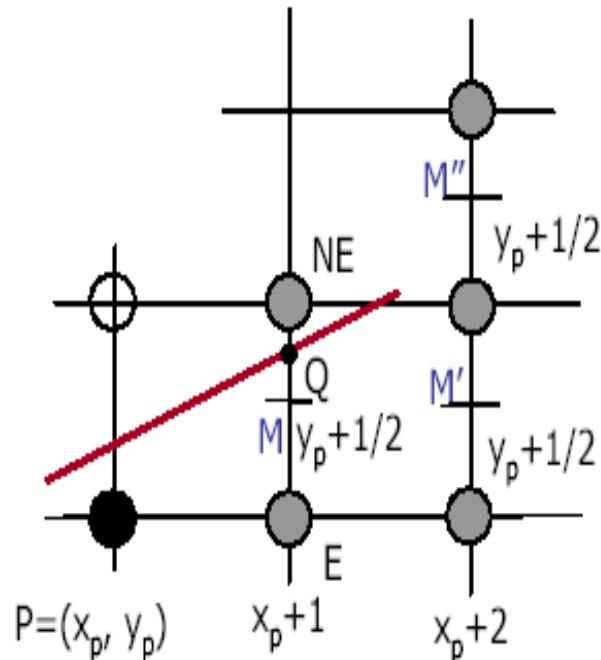
Thuật toán trung điểm

□ Phương trình đường thẳng:

$$F(x, y) = ax + by + c$$

□ Giá trị hàm tại M: $F(M) = F(x_p + 1, y_p + 1/2) = d$

- Nếu $d > 0$, M nằm dưới đường thẳng -> chọn NE
- Nếu $d < 0$, M nằm phía trên -> chọn E
- Nếu $d = 0$, chọn E hay NE tùy ý



Thuật toán trung điểm vẽ line

- Giá trị của hàm tại M của điểm tiếp theo
- Gọi giá trị d vừa tính là

$$d_{old} = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$$

■ giả sử vừa chọn E

$$d_{new} = F(x_p + 2, y_p + \frac{1}{2}) = a(x_p + 2) + b(y_p + \frac{1}{2}) + c$$

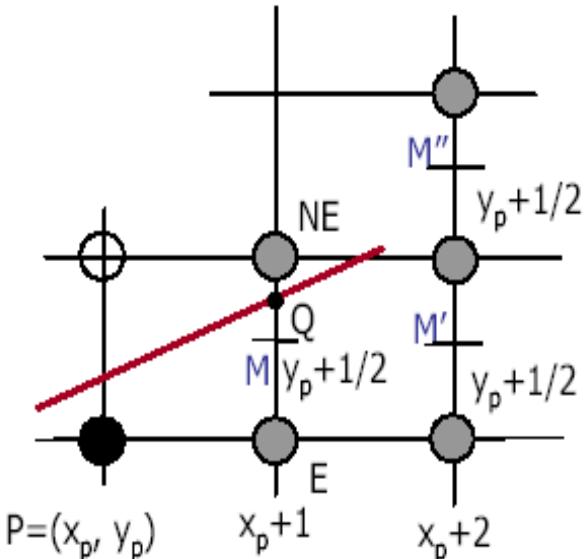
$d_{new} = d_{old} + a = d_{old} + dy \rightarrow dy$ là số gia
của điểm tiếp theo

■ giả sử vừa chọn NE

$$d_{new} = F(x_p + 2, y_p + \frac{3}{2}) = a(x_p + 2) + b(y_p + \frac{3}{2}) + c$$

□ $d_{new} = d_{old} + a + b = d_{old} + dy - dx$

□ $dy - dx$ là số gia của điểm tiếp theo



Thuật toán trung điểm vẽ line

□ Tính giá trị khởi đầu của d

- giả sử vẽ đoạn thẳng từ (x_0, y_0) đến (x_1, y_1) -> trung điểm thứ nhất có tọa độ $(x_0+1, y_0+1/2)$

$$F(x_0 + 1, y_0 + \frac{1}{2}) = a(x_0 + 1) + b(y_0 + \frac{1}{2}) + c =$$

$$ax_0 + by_0 + c + a + \frac{b}{2} = F(x_0, y_0) + a + \frac{b}{2}$$

- $F(x_0, y_0) = 0 \rightarrow d_{\text{start}} = a + b/2 = dy - dx/2$
- Tránh số thập phân của d_{start} định nghĩa lại hàm như sau $F(x, y) = 2(ax + by + c)$
- Do vậy, ta có

$$d_{\text{start}} = 2dy - dx; \quad \Delta_\varepsilon = 2dy; \quad \Delta_{\text{NE}} = 2(dy - dx)$$

Thuật toán trung điểm

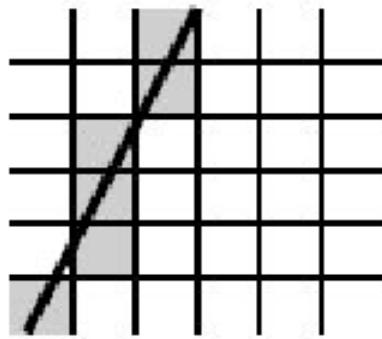
```
void MidpointLine(int x1, y1, x2, y2)
{
    Int dx=x2-x1;
    Int dy=y2-y1;
    Int d=2*dy-dx;
    Int increE=2*dy;
    Int incrNE=2*(dy-dx);
    x=x1;
    y=y1;
    PutPixel(x, y);

    while (x < x2) {
        if (d<= 0) {
            d+=incrE;
            x++;
        } else {
            d+=incrNE;
            x++;
            y++;
        }
        PutPixel(x, y);
    }
}
```

Thuộc tính của đường vẽ

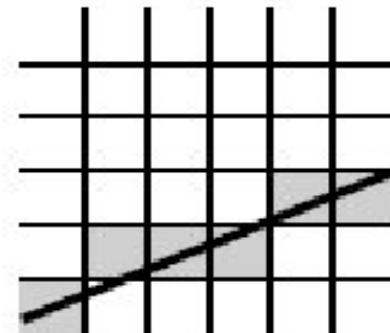
- Thuật toán vẽ đoạn thẳng nói trên đều vẽ đoạn thẳng có độ rộng 1 pixel, nét liên tục
- Hai thuộc tính quan trọng của đường vẽ
 - độ rộng: vẽ đoạn thẳng từ (x_0, y_0) đến (x_1, y_1)
 - Nếu $dy > dx$: các pixel được vẽ thêm tại tọa độ bên trái và bên phải điểm vẽ ($x-1$ và $x+1$)
 - Nếu $dx > dy$: vẽ thêm các pixel phía trên và dưới điểm vừa vẽ
 - Đường nét đứt
 - sử dụng các pattern, mặt nạ với bit cao nhất bằng 1
 - Dựa trên kết quả phép AND mặt nạ với mẫu để quyết định có vẽ điểm ảnh tại vị trí hiện hành hay không.

□ Đường thẳng có hệ số góc khác



**slope > 1 , cannot step
along x**

To handle slope > 1 , swap x and y



**slope < 1 , can step
along x**

CÁC THUẬT TOÁN CƠ SỞ VẼ ĐỒ HỌA

Computer Graphics

Phần 2

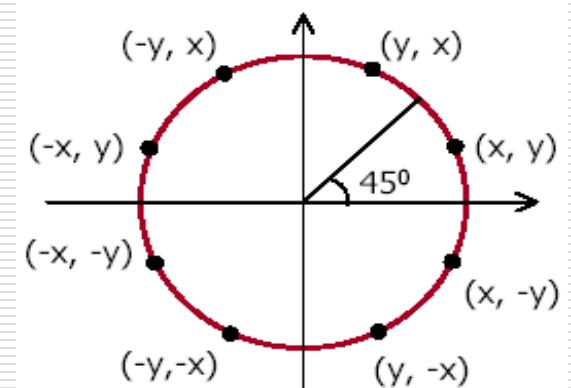
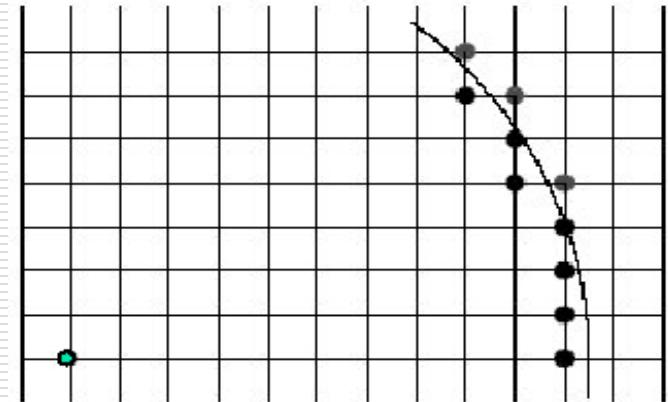
Các thuật toán vẽ đường tròn

Các thuật toán vẽ đường tròn

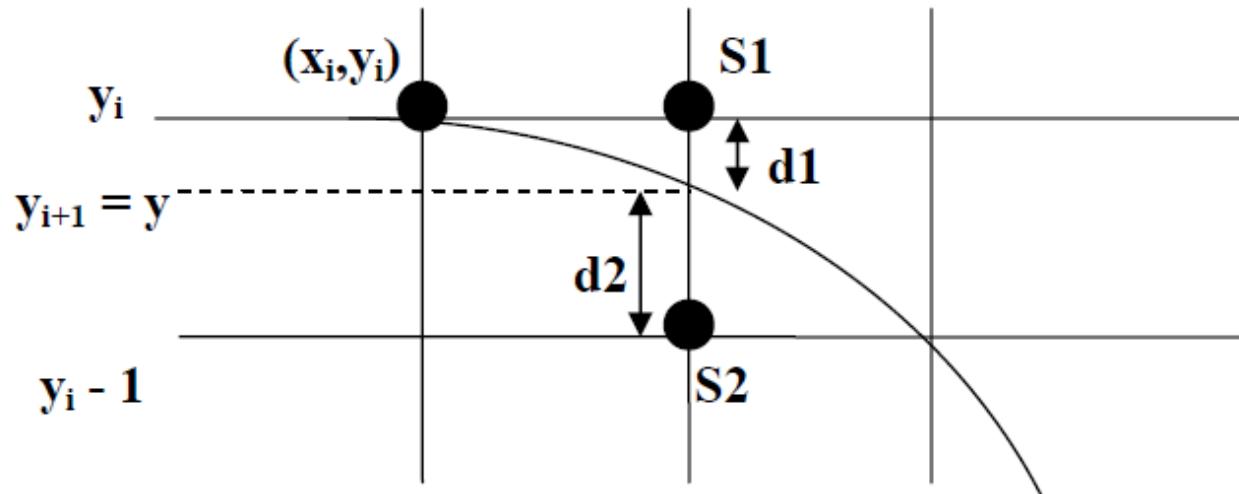
- Một số tính chất đường tròn
- Thuật toán Bresenham
- Thuật toán trung điểm

Các thuật toán vẽ đường tròn

- Tương tự như vẽ đoạn thẳng, đường tròn đồ họa hình thành bởi các pixel gần đường tròn toán học nhất (rasterization).
- Một vài tính chất cơ bản:
 - Vẽ đường tròn tâm tại gốc tọa độ sau đó dịch chuyển đến vị trí mong muốn.
 - Tính đối xứng: khi biết tọa độ 1 điểm dễ dàng suy ra tọa độ của 7 điểm còn lại.
 - Sử dụng phương trình để tính toán tọa độ đường tròn -> dấu phẩy động.



Thuật toán Bresenham vẽ đường tròn



$$d_1 = (y_i)^2 - y^2 = (y_i)^2 - (R^2 - (x_i + 1)^2)$$

$$d_2 = y^2 - (y_i - 1)^2 = (R^2 - (x_i + 1)^2) - (y_i - 1)^2$$

$$P_i = d_1 - d_2$$

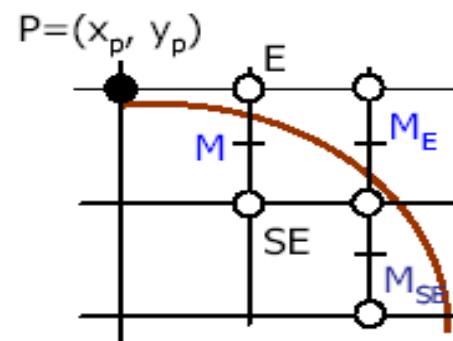
Thuật toán Bresenham vẽ đường tròn

- Chọn vị trí thứ nhất để vẽ có tọa độ $(x_1, y_1) = (0, r)$
 - Tính tham số thứ nhất: $p_1 = 3 - 2r$
 - Nếu $p_1 < 0$: vị trí tiếp theo là $(x_1 + 1, y_1)$. Ngược lại vẽ tại tọa độ $(x_1 - 1, y_1 - 1)$.
 - Tiếp tục tăng x để tính p tiếp theo từ p trước đó
 - Nếu trước đó có $p_i < 0$: $p_{i+1} = p_i + 4x_i + 6$
 - Ngược lại, ta có: $p_{i+1} = p_i + 4(x_i - y_i) + 10$
 - Nếu kết quả $p_{i+1} < 0$: điểm sẽ chọn tiếp theo là (x_{i+1}, y_{i+1})
 - ngược lại, ta chọn $(x_{i+1}, y_{i+1} - 1)$
 - Nếu $p_i < 0$ thì $y_{i+1} = y_i$, ngược lại $y_{i+1} = y_i - 1$
 - Lặp lại bước 3 cho đến khi $x = y$
-

-
- Thuật toán trung điểm vẽ đường tròn
 - Thuật toán trung điểm vẽ ellipse
 - Thuật toán clipping đoạn thẳng, đa giác
 - Thuật toán tô màu đa giác.
-

Thuật toán trung điểm vẽ đường tròn

- Ý tưởng thuật toán: Khi đã vẽ điểm P tại (x_p, y_p) , phải quyết định điểm vẽ tiếp theo là E hay SE
- Phương trình đường tròn
 - $F(x, y) = x^2 + y^2 - R^2$
 - $F(x, y) = 0 \rightarrow (x, y)$ trên đường tròn
 - $F(x, y) < 0 \rightarrow (x, y)$ trong đường tròn
 - $F(x, y) > 0 \rightarrow (x, y)$ ngoài đường tròn
 - Nếu M trong vòng tròn \rightarrow E gần đường tròn
 - Ngược lại \rightarrow SE gần đường tròn



Thuật toán trung điểm vẽ đường tròn

- Biến quyết định d: giá trị hàm tại điểm giữa M

$$d_{old} = F(x_p + 1, y_p - \frac{1}{2}) = (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2$$

- Nếu $d_{old} < 0$ thì chọn E, x_p tăng 1, y_p giữ nguyên.

$$d_{new} = F(x_p + 2, y_p - \frac{1}{2}) = (x_p + 2)^2 + (y_p - \frac{1}{2})^2 - R^2$$

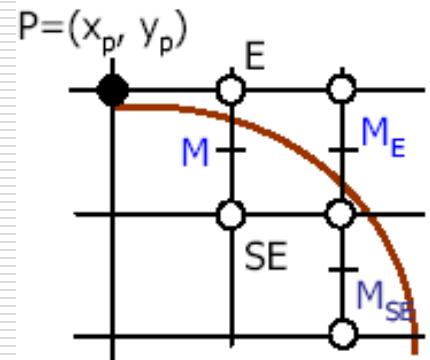
$$d_{new} = d_{old} + (2x_p + 3) = d_{old} + \Delta_E$$

- Nếu $d_{old} > 0$ thì chọn SE, x_p tăng 1, y_p giảm

$$d_{new} = F(x_p + 2, y_p - \frac{3}{2})$$

$$= (x_p + 2)^2 + (y_p - \frac{3}{2})^2 - R^2$$

$$d_{new} = d_{old} + (2x_p - 2y_p + 5) = d_{old} + \Delta_{SE}$$



Thuật toán trung điểm vẽ đường tròn

□ Vòng lặp của thuật toán

- Chọn pixel để vẽ dựa trên dấu biến quyết định d của vòng lặp trước
- Cập nhật biến quyết định d bởi giá trị Δ tương ứng với pixel SE hay E vừa chọn

□ Giá trị khởi đầu

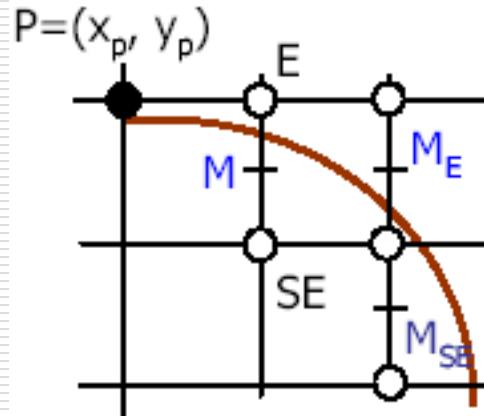
- Điểm vẽ đầu tiên có tọa độ $(0, R)$
- Biến quyết định d có giá trị:

Đặt biến quyết định mới $h = d - 1/4$, ta có:

$$d = F(1, r - \frac{1}{2}) = 1 + (R^2 - R + \frac{1}{4}) - R^2 = \frac{5}{4} - R$$

$$h + \frac{1}{4} = \frac{5}{4} - R$$

$$h = 1 - R$$



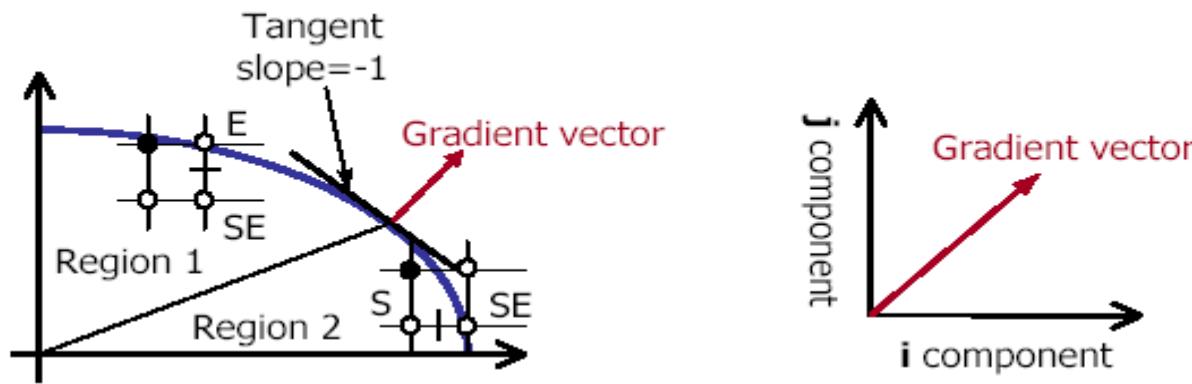
Thuật toán trung điểm vẽ elip

- Phương trình elip có tâm tại gốc tọa độ

$$F(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$$

- Áp dụng thuật toán trung điểm vẽ đường tròn để vẽ elip

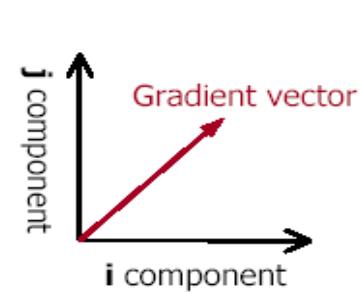
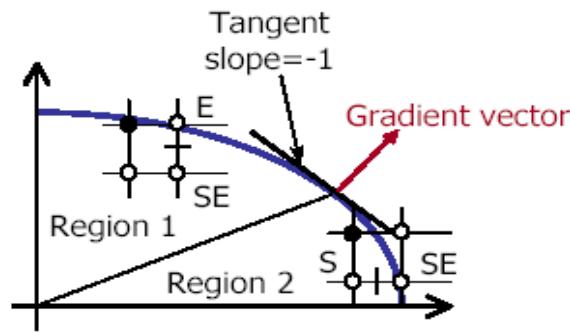
- Tính đối xứng của elip: khi biết tọa độ 1 điểm có thể dễ dàng suy ra tọa độ ba điểm khác.



Thuật toán trung điểm vẽ elip

- Tìm ranh giới giữa hai miền trong ¼ elip
 - Vị trí: điểm P là tiếp điểm của tiếp tuyến có hệ số góc -1
 - Xác định:
 - Vector vuông góc với tiếp tuyến tại tiếp điểm -> gradient
- Tại P các thành phần i và j của vector gradient có cùng độ lớn

Miền 1: Thành phần j lớn hơn thành phần i
 $a^2y > b^2x$



Thuật toán trung điểm vẽ elip

- Ý tưởng: đánh giá hàm tại điểm giữa hai tọa độ pixel để chọn vị trí tiếp theo để vẽ. Dấu của nó cho biết điểm giữa nằm trong hay ngoài elip.
- Với vùng 1:
 - Tính biến quyết định $d = F(x, y) = F(x_p + 1, y_p - 1/2)$
 - Nếu $d < 0$: chọn E, x tăng 1, y không thay đổi

$$d_{old} = F(x_p + 1, y_p - \frac{1}{2}) = b^2(x_p + 1)^2 + a^2(y_p - \frac{1}{2})^2 - a^2b^2$$

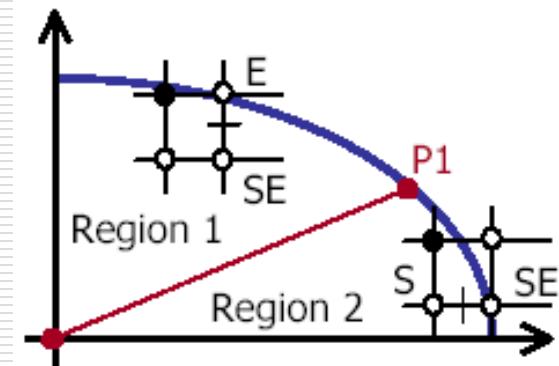
$$d_{new} = F(x_p + 2, y_p - \frac{1}{2}) = b^2(x_p + 2)^2 + a^2(y_p - \frac{1}{2})^2 - a^2b^2$$

$$d_{new} = d_{old} + b^2(2x_p + 3) = d_{old} + \Delta_E$$

- Nếu $d \geq 0$: chọn SE, x tăng 1, y giảm 1

$$d_{new} = F(x_p + 2, y_p - \frac{3}{2}) = b^2(x_p + 2)^2 + a^2(y_p - \frac{3}{2})^2 - a^2b^2$$

$$d_{new} = d_{old} + b^2(2x_p + 3) + a^2(-2y_p + 2) = d_{old} + \Delta_{SE}$$



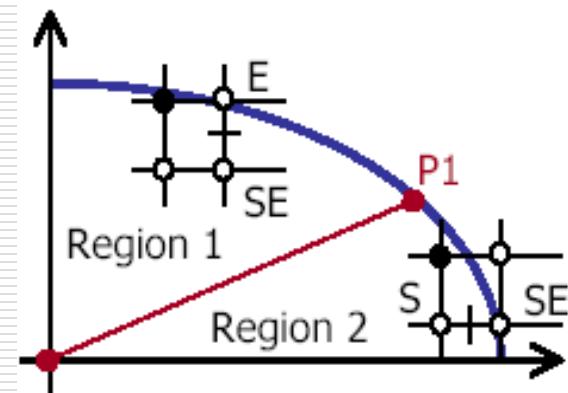
Thuật toán trung điểm vẽ elip

❑ Với vùng 2:

- Tính biến quyết định $d=F(x,y)=F(x_p+1/2, y_p-1)$
 - ❑ Nếu $d < 0$: chọn SE, x tăng 1, y giảm 1
 - ❑ Nếu $d \geq 0$: chọn S, x không tăng, y giảm 1
- Tìm số gia như vùng 1

$$\Delta_S = a^2(-2y_p + 3)$$

$$\Delta_{SE} = b^2(2x_p + 2) + a^2(-2y_p + 3)$$



Thuật toán trung điểm vẽ elip

□ Tìm giá trị khởi đầu của số gia d

Miền 1:

giả sử a, b nguyên: điểm bắt đầu vẽ là

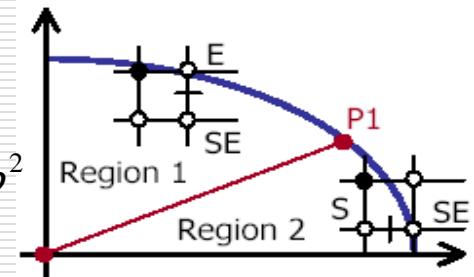
$$(0, b) \quad F(1, b - \frac{1}{2}) = b^2 + a^2(b - \frac{1}{2})^2 - a^2b^2 = b^2 + a^2(-b + \frac{1}{4}) = b^2 - a^2b + \frac{a^2}{4}$$

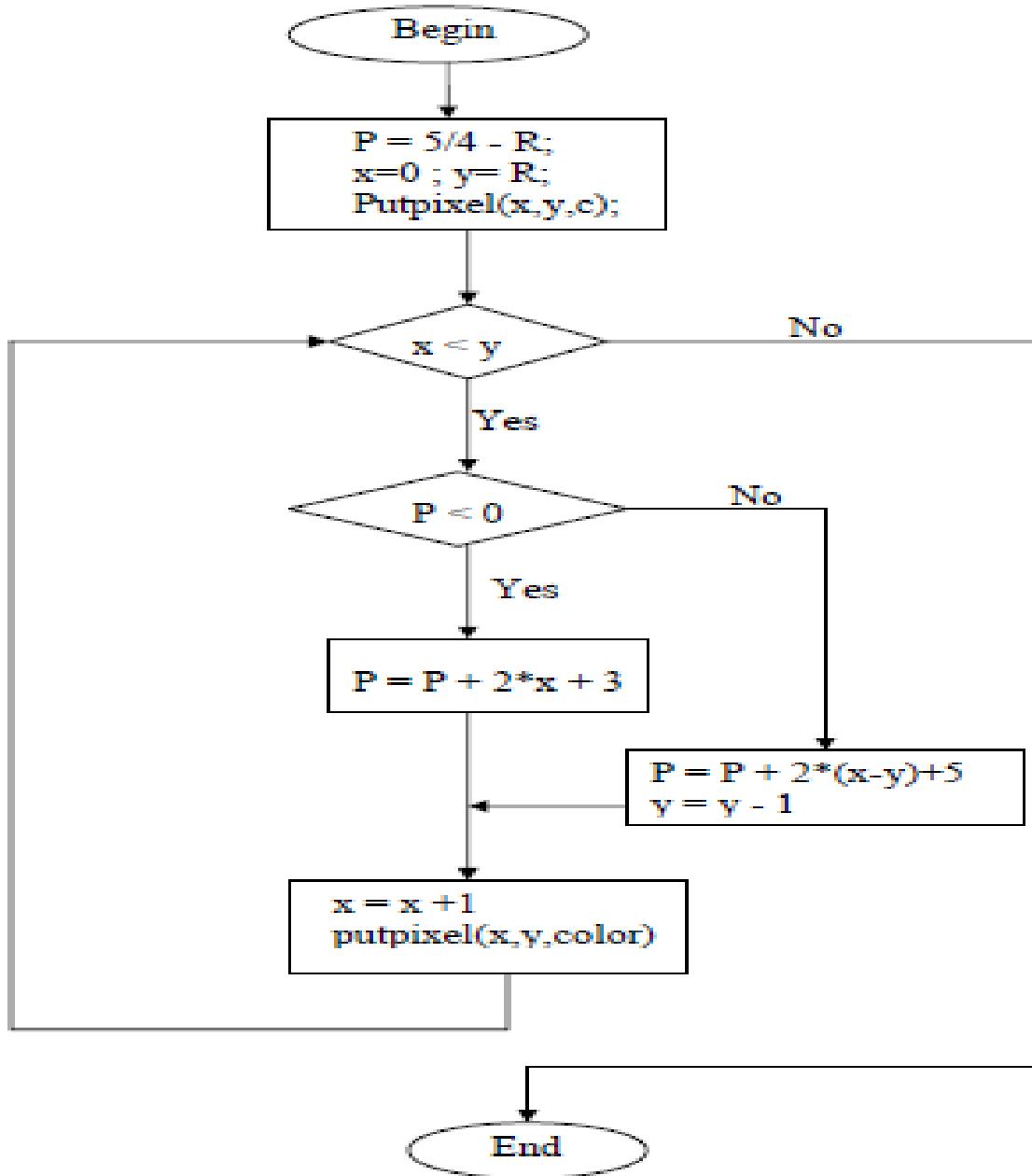
Điểm giữa thứ nhất: $(1, b - 1/2)$

Miền 2:

phụ thuộc vào điểm giữa $(x_p + 1, y_p - 1/2)$ của điểm tiếp theo điểm cuối cùng của miền 1.

$$\begin{aligned} F(x_p + \frac{1}{2}, y_p - \frac{1}{2}) &= b^2(x_p + \frac{1}{2})^2 + a^2(y_p - 1)^2 - a^2b^2 \\ &= b^2x^2 + b^2x + \frac{b^2}{4} + a^2(y_p - 1)^2 - a^2b^2 \end{aligned}$$





CÁC THUẬT TOÁN CƠ SỞ VẼ ĐỒ HỌA

Computer Graphics

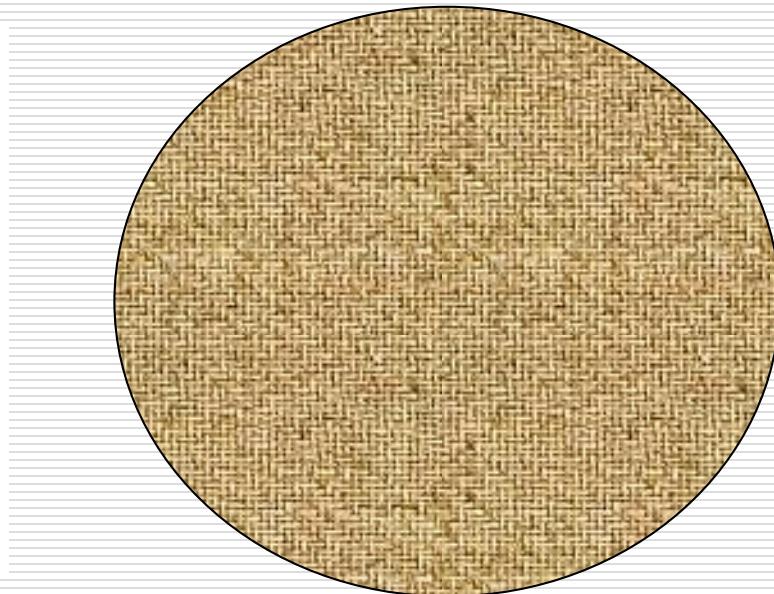
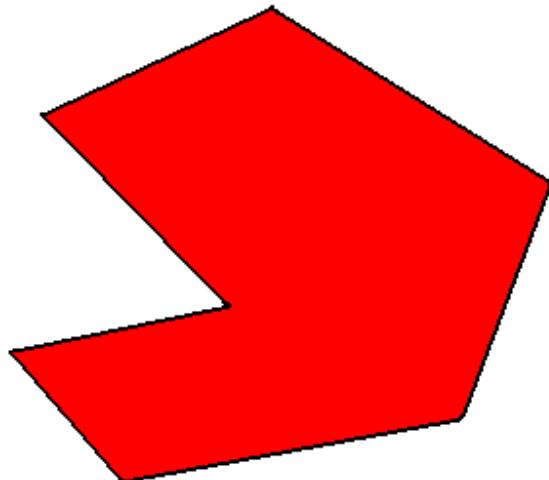
Các thuật toán tô màu

Định nghĩa:

Tô màu một vùng là xác lập màu sắc của tất cả các điểm nằm trong vùng cần tô

Các bước tô màu

- Xác định vị trí các điểm cần tô màu
- Quyết định tô các điểm bằng màu tương ứng

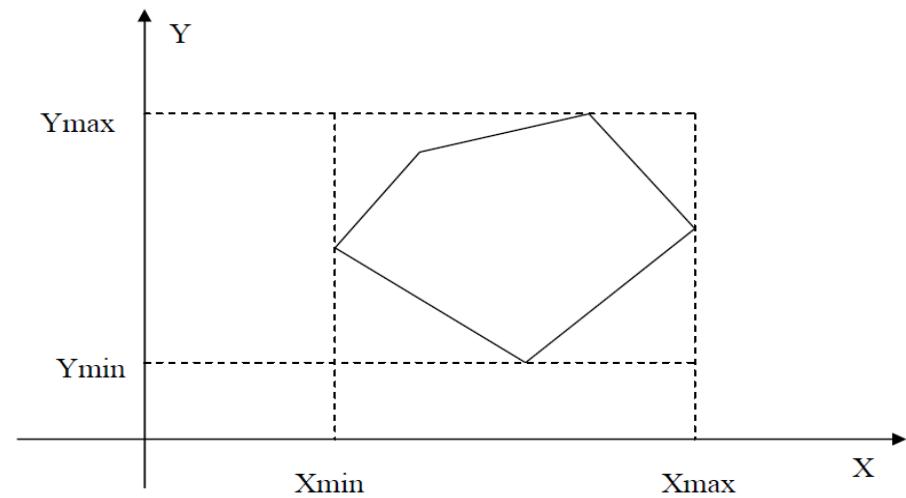
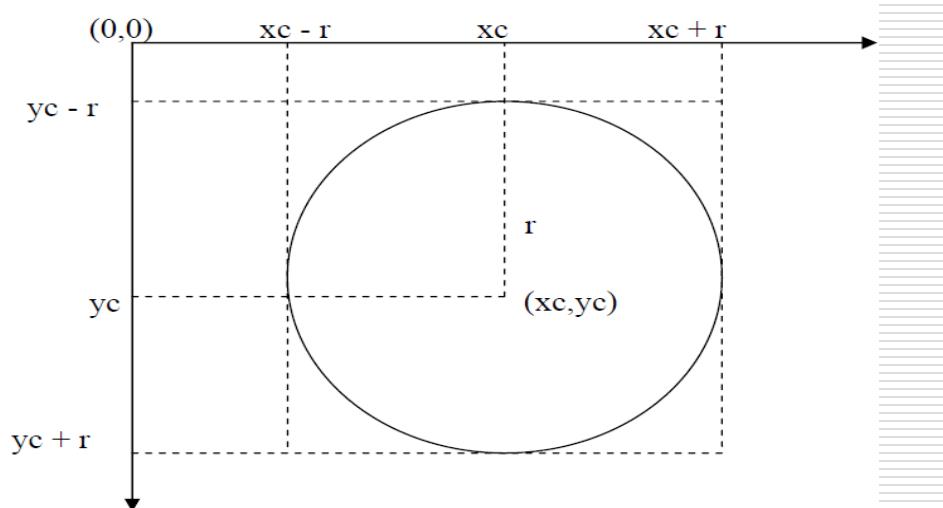


Các cách tô màu

- Tô màu theo từng điểm (tô màu đơn giản)
- Tô màu theo dòng quét
- Tô màu theo đường biên

Tô màu theo từng điểm

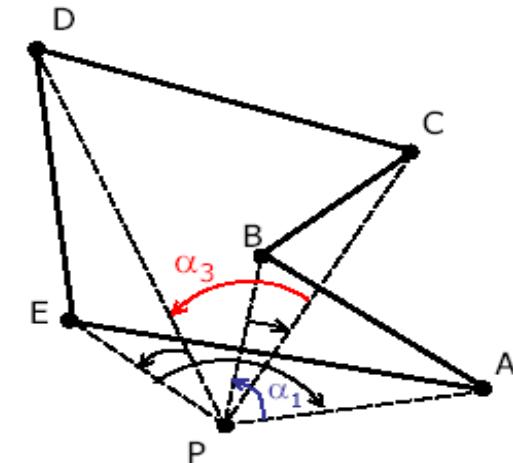
- Xác định điểm có thuộc vùng cần tô màu hay không?
- Xác lập màu sắc nếu điểm thuộc vùng cần tô màu.



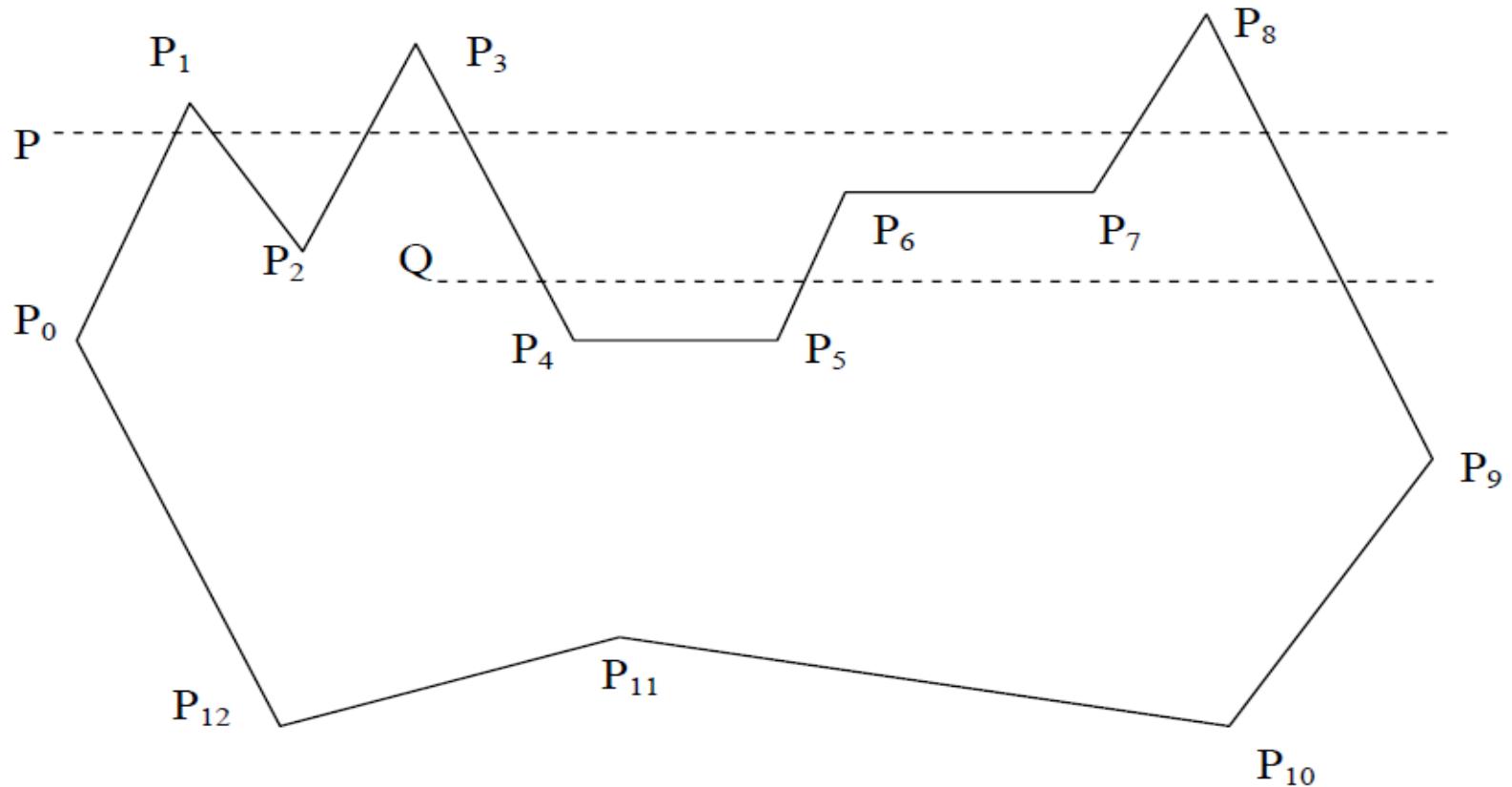
Xác định một điểm có nằm trong đa giác

□ Phương pháp kiểm tra góc

- Thí dụ, xét xem điểm P cho trước có ở trong đa giác ABCDE?
- từ điểm P nối với các đỉnh đa giác để tạo thành các góc theo thứ tự ngược chiều kim đồng hồ
- Các góc này có giá trị dương hoặc âm tùy theo hướng đo
- Tính tổng các góc
 - Nếu tổng các góc bằng 0 thì P nằm ngoài đa giác
 - Nếu tổng các góc bằng 360° thì P nằm trong đa giác



Xác định một điểm có nằm trong đa giác



Xác định điểm trong đa giác

- Thuật toán này ứng dụng để xác định điểm độc lập hay đầu mút đoạn thẳng nằm trong đa giác
- Định lý nửa đường thẳng của Jordan
 - từ điểm cho trước hãy vẽ tia ra cạnh tận cùng của đa giác
 - Tính tổng giao điểm của tia với các cạnh đa giác
 - Nếu tổng số điểm là lẻ thì điểm đó nằm trong đa giác, ngược lại tổng số điểm là chẵn thì điểm đó nằm ngoài đa giác
 - để dễ tính toán hãy dựng tia song song với trục tọa độ

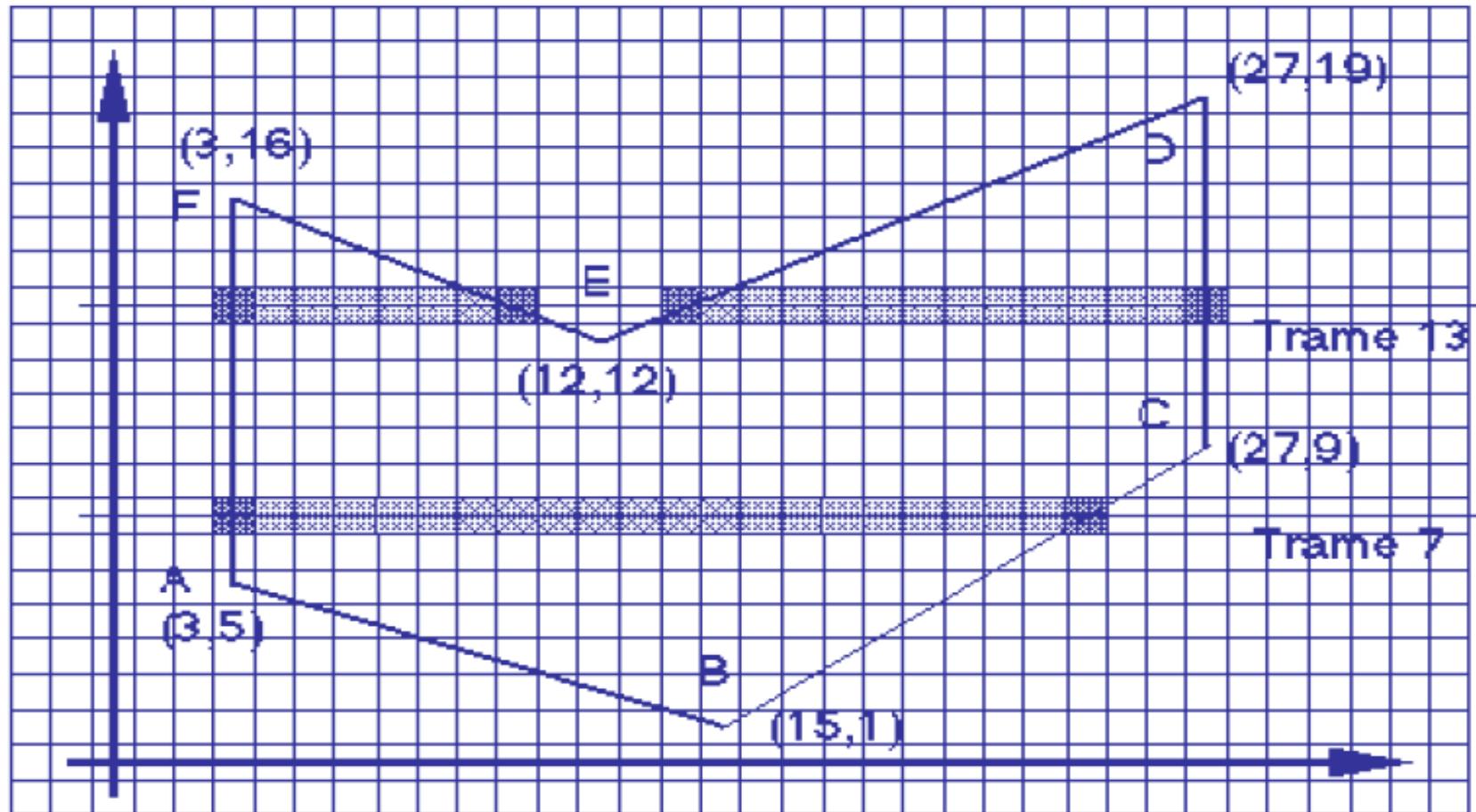
Xác định một điểm có nằm trong đa giác

- Qui ước: tại các đỉnh của đa giác thì số giao điểm được tính hai lần
 - Thuật toán này đúng cả với trường hợp đa giác có lỗ hổng
 - Sinh viên tự tìm hiểu thêm các thuật toán xác định điểm có nằm trong đa giác hay không
-

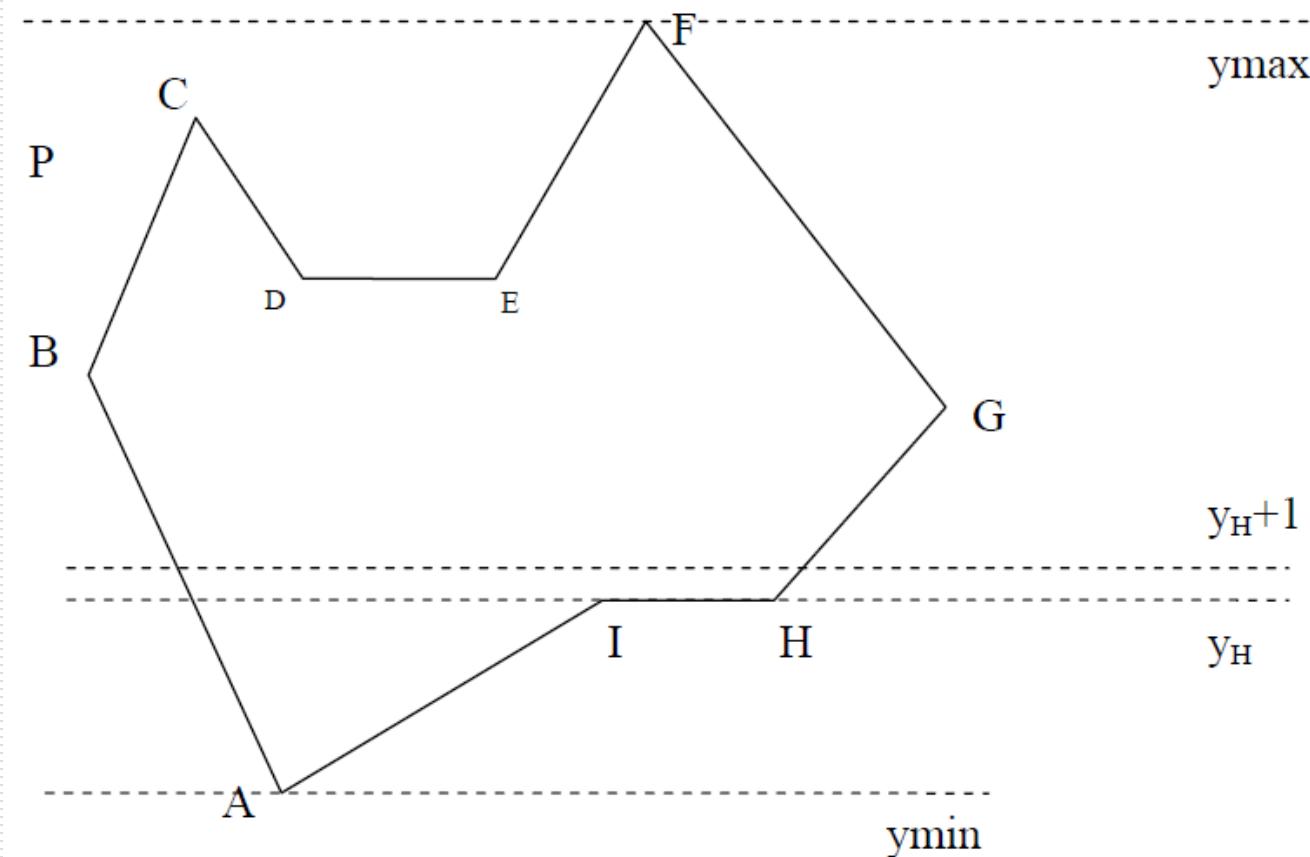
Nhận xét

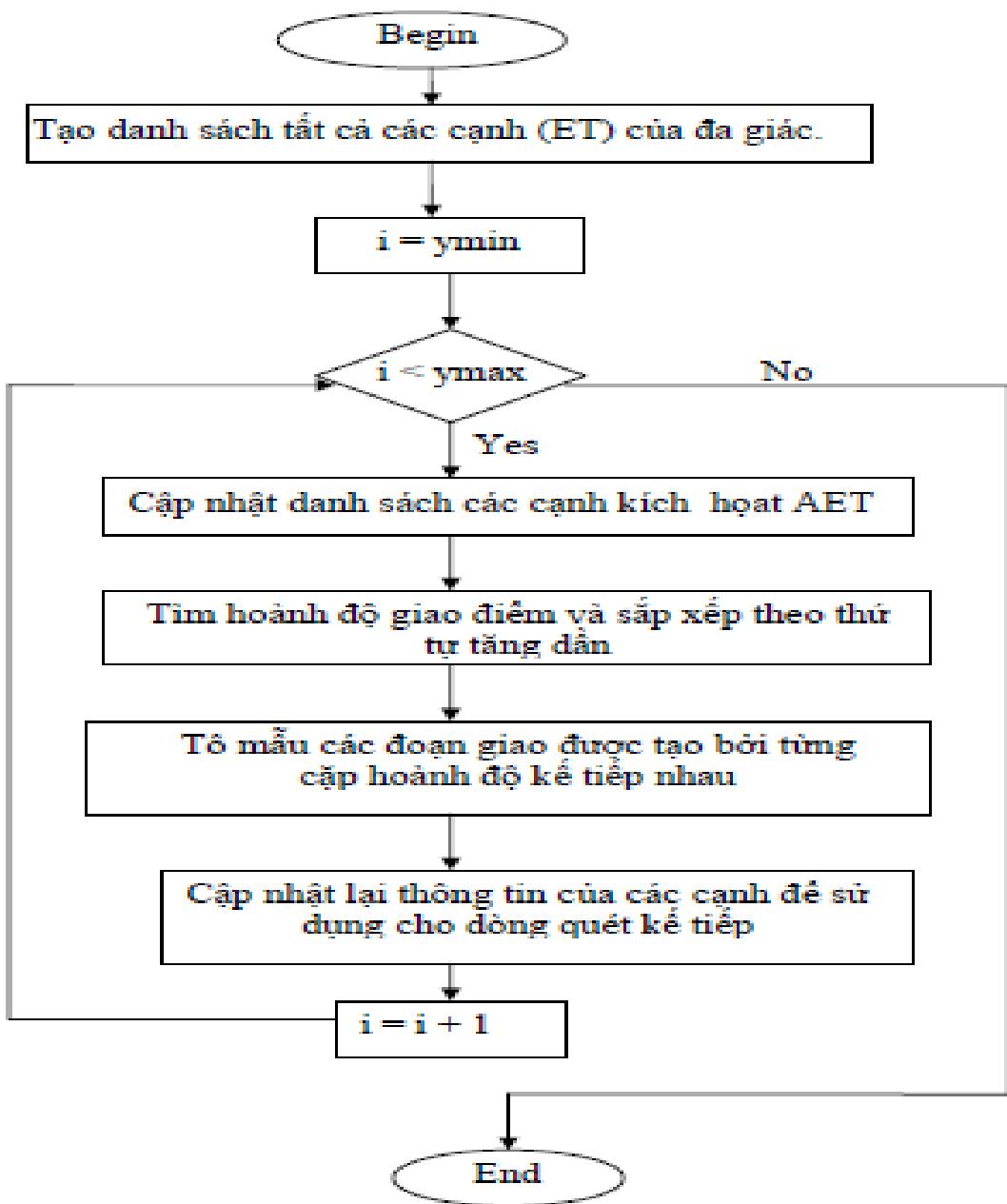
- Tô màu mịn
- Có thể áp dụng cho nhiều loại đối tượng hình học khác nhau
- Tốc độ chậm

Tô màu theo dòng quét



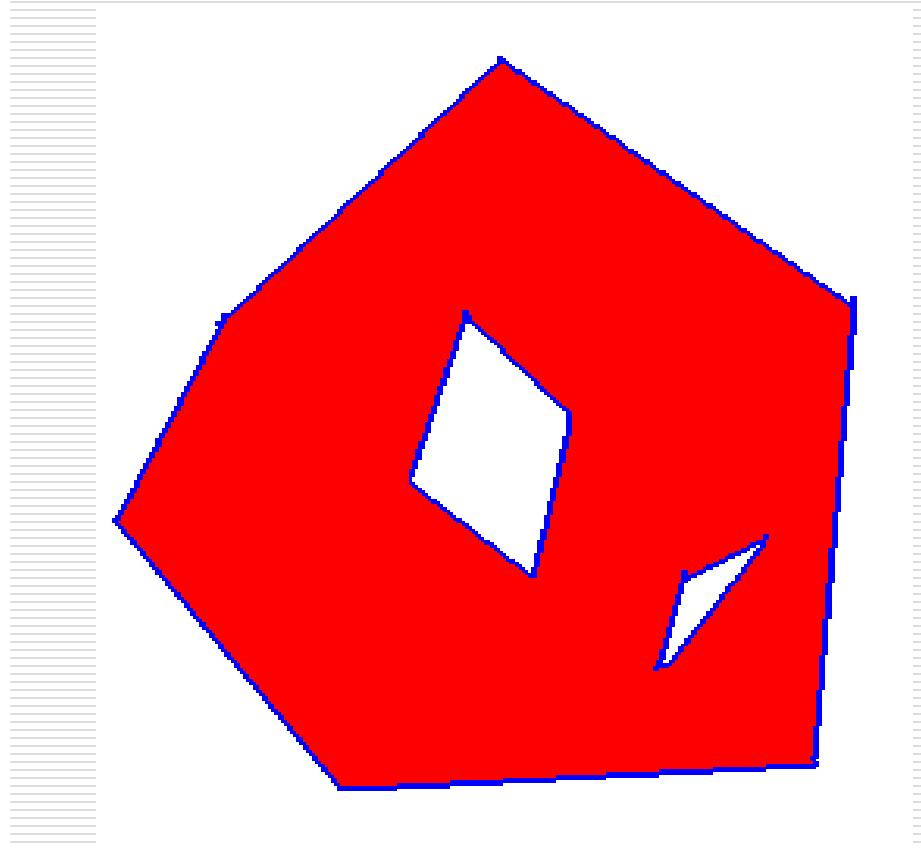
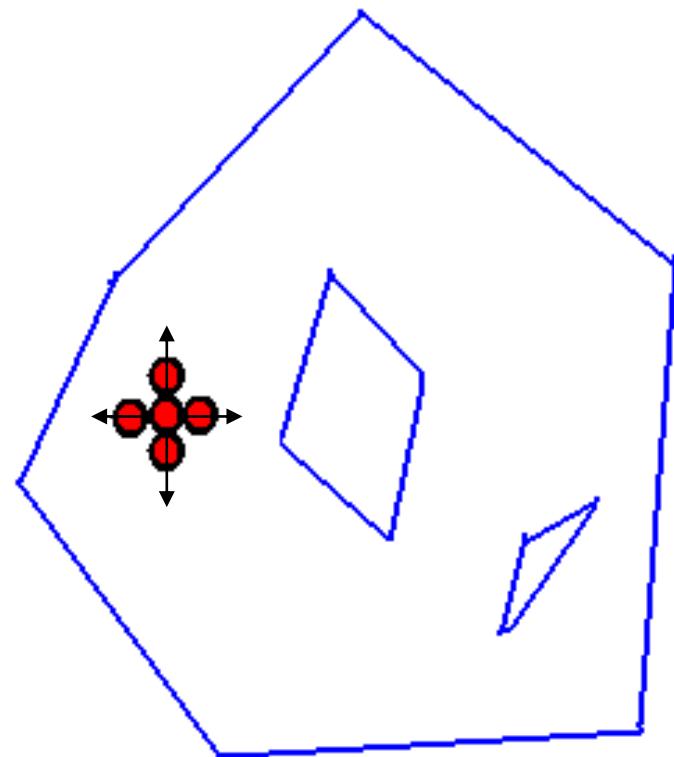
Hạn chế số lượng dòng quét





ET- Edge table
AET – Active Edge Table

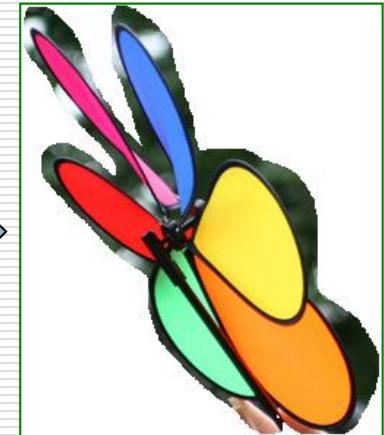
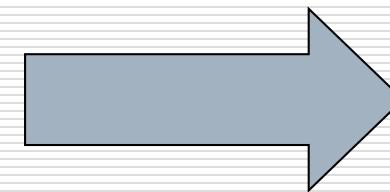
Tô màu theo vùng biên



```
void floodFill4 (int x, int y, int fillColor, int oldColor)
{
    if (getPixel (x, y) == oldColor) {
        setColor (fillColor);
        setPixel (x, y);
        floodFill4 (x+1, y, fillColor, oldColor);
        floodFill4 (x-1, y, fillColor, oldColor);
        floodFill4 (x, y+1, fillColor, oldColor);
        floodFill4 (x, y-1, fillColor, oldColor);
    }
}
```

Các thuật toán clipping

- Cắt xén bởi hình chữ nhật
 - Cắt xén đoạn thẳng
 - Cắt xén đa giác
- Cắt xén vùng bởi đa giác bất kỳ



Xác định giao hai đường thẳng

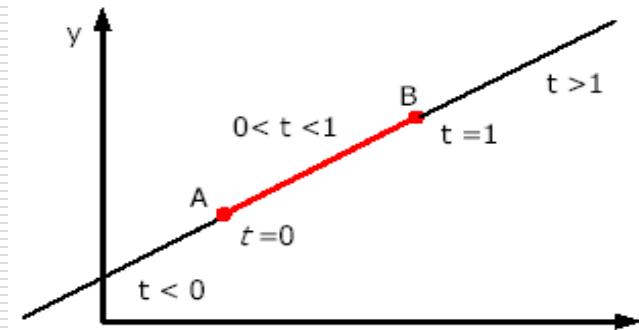
- Giao của hai đường thẳng đi qua hai điểm
 - Ví dụ đơn giản
 - Đường thẳng đi qua tọa độ (4, 2) và (2, 0) có giao với đoạn thẳng đi qua (0, 4) và (4, 0)?
 - Giải pháp
 - Xác định phương trình đường thẳng qua 2 điểm $y=ax+b$, trong đó $a=(y_2-y_1)/(x_2-x_1)$
 - từ ví dụ trên ta có: $y=-2+x$ và $y=4-x$ giao điểm tại (3, 1)
 - Tổng quát: nếu ta có $y=a_1+b_1x$ và $y=a_2+b_2x$ thì giao điểm sẽ ở tại: $x_i = -(a_1-a_2)/(b_1-b_2)$
 $y_i = a_1+b_1x_i$
 - Các trường hợp đặc biệt: song song với trục x hay trục y, song song với nhau.

Xác định giao hai đoạn thẳng

- Nếu sử dụng phương pháp tìm giao đường thẳng: đòi hỏi kiểm tra tọa độ của giao điểm đường thẳng có nằm trong đoạn thẳng hay không?
- Phương pháp khác: biểu diễn đoạn thẳng bằng tham số
 - Đoạn thẳng 1 từ (x_A, y_A) đến (x_B, y_B)
 - Đoạn thẳng 2 từ (x_C, y_C) đến (x_D, y_D)
 - Tính toán giao của hai đoạn thẳng tại tọa độ có t, s :

$$t = \frac{(x_C - x_A)(y_C - y_D) - (x_C - x_A)(y_C - y_A)}{(x_B - x_A)(y_C - y_D) - (x_C - x_D)(y_B - y_A)}$$

$$s = \frac{(x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A)}{(x_B - x_A)(y_C - y_D) - (x_C - x_D)(y_B - y_A)}$$



$$x = x_A + t(x_B - x_A)$$

$$y = y_A + t(y_B - y_A)$$

$$x = x_C + s(x_D - x_C)$$

$$y = y_C + s(y_D - y_C)$$

$$0 \leq t \leq 1 \text{ và } 0 \leq s \leq 1$$

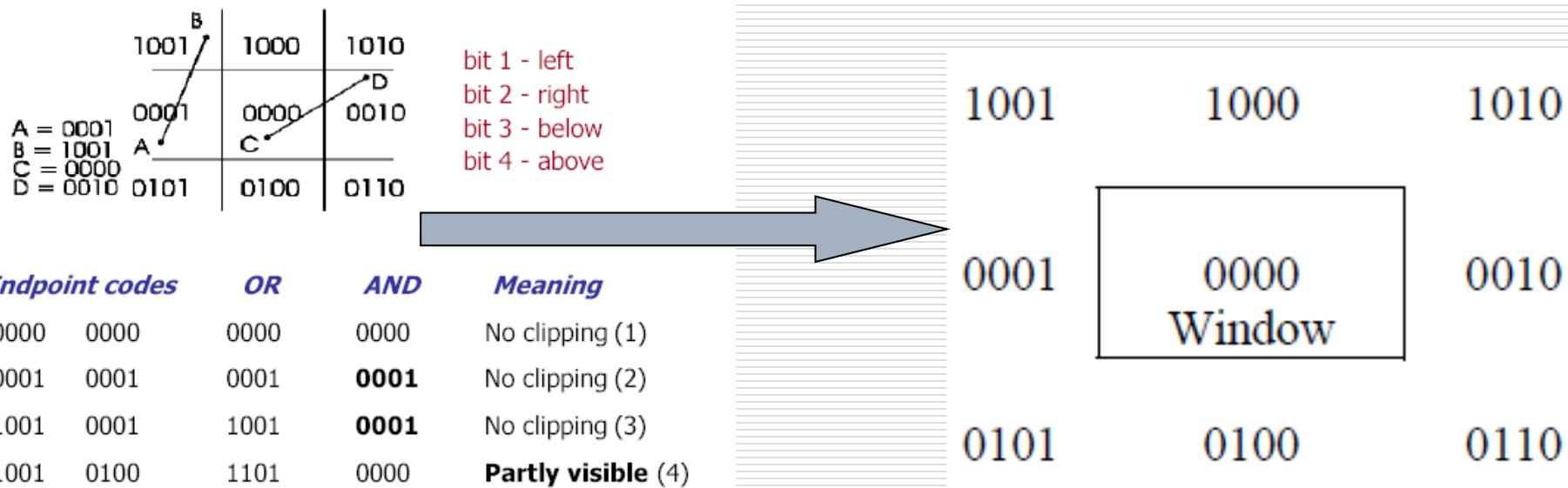
Thuật toán cắt xén đoạn thẳng

❑ Các giải pháp

- Kiểm tra từng pixel của đoạn thẳng có ở trong hcn?
- Tính toán các điểm cắt của từng đoạn thẳng với cạnh chữ nhật cắt xén
- Thuật toán Sutherland-Cohen: loại bỏ các đoạn không cần cắt xén bằng xét tọa độ đầu mút các đoạn thẳng -> đơn giản và hiệu quả.
- Chia nhỏ trung điểm
- Cyrus & Beck algorithm

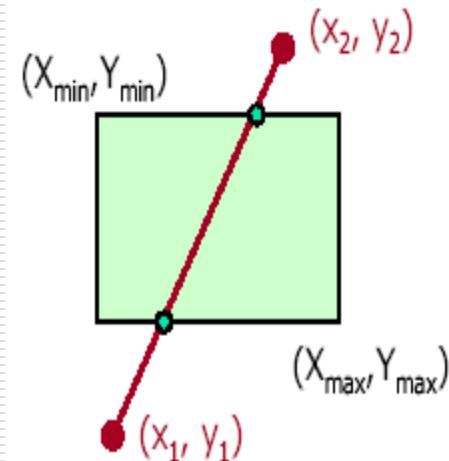
Thuật toán Sutherland-Cohen

- Mã hóa các đầu mút các đoạn thẳng
- Xác định nhanh đoạn thẳng có cần cắt xén hay không nhờ các phép toán logic AND và OR



Thuật toán Sutherland-Cohen

- Kết quả phép OR hai mã đầu mút đoạn thẳng cho kết quả 0: cả hai điểm nằm trong chữ nhật.
- Kết quả phép AND hai mã đầu mút đoạn thẳng cho kết quả khác 0: cả hai điểm nằm ngoài chữ nhật
- Cắt xén
 - Giao của đoạn thẳng với các cạnh chữ nhật song song trực tung
 - x có giá trị X_{\min}, X_{\max} và hệ số góc $a = (y_2 - y_1) / (x_2 - x_1)$
$$y = y_1 + a(x - x_1)$$
 - Giao đoạn thẳng với các cạnh song song trực hoành
 - y có giá trị Y_{\min}, Y_{\max} và hệ số góc
$$x = x_1 + (y - y_1) / a$$



Thuật toán chia nhỏ trung điểm

- Dựa trên phương pháp chia đôi
- Đoạn thẳng được chia tại trung điểm của nó thành 2 đoạn nhỏ hơn → xác định xem thuộc loại cắt nào? → tiếp tục chia nhỏ → chỉ còn đoạn thuộc loại nhìn thấy hoặc không nhìn thấy
- Với khung nhìn trong thiết bị vật lý → đoạn thẳng thu được nhỏ hơn độ chính xác của màn hình.
- Nếu đoạn thẳng có số điểm ảnh là M → phải chia N phép chia trong đó $N = \log_2 M$.

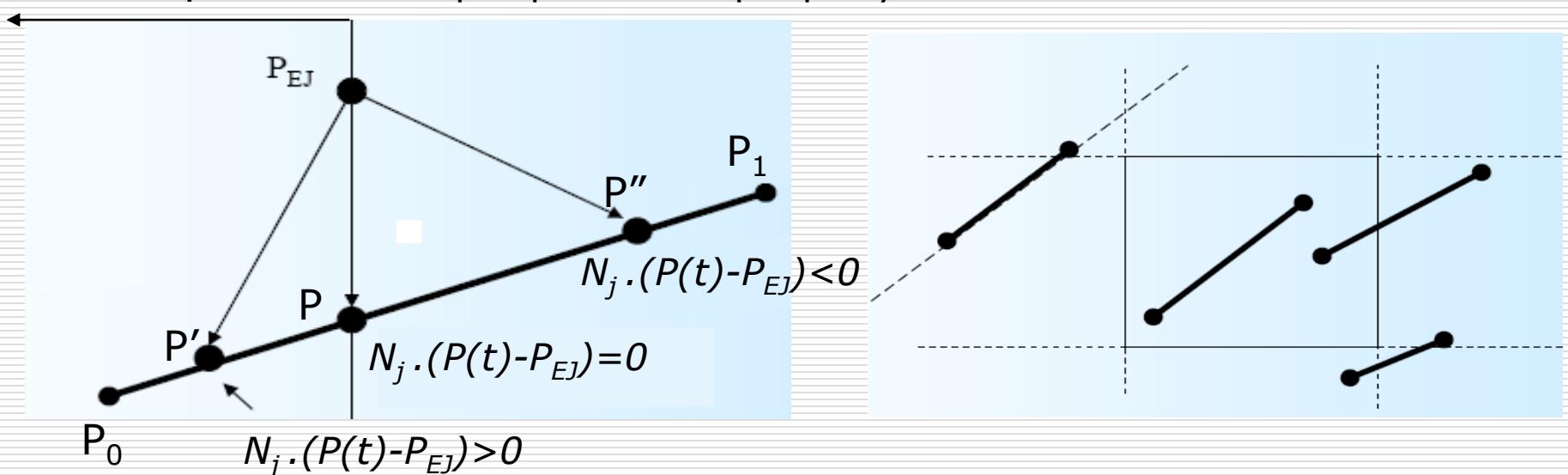
Thuật toán Cyrus & Beck

- Sử dụng phương trình tham số:

$$\bullet P(t) = P_0 + t(P_1 - P_0)$$

- Xác định giao điểm của đoạn thẳng với biên hcn, sau đó xem chúng có thuộc vùng clipping?

- Tìm một điểm trên mp clip và vecto pháp tuyến của nó



$$P(t) = P_0 + (P_1 - P_0)t$$

$$\mathbf{N}_j \cdot (\mathbf{P}(t) - \mathbf{P}_{EJ}) = \mathbf{0} \Leftrightarrow \mathbf{N}_j \cdot (P_0 + (P_1 - P_0)t - \mathbf{P}_{EJ}) = \mathbf{0}$$

$$\Leftrightarrow \mathbf{N}_j \cdot (P_0 - \mathbf{P}_{EJ}) + \mathbf{N}_j \cdot (P_1 - P_0)t = \mathbf{0}$$

$$D = P_1 - P_0$$

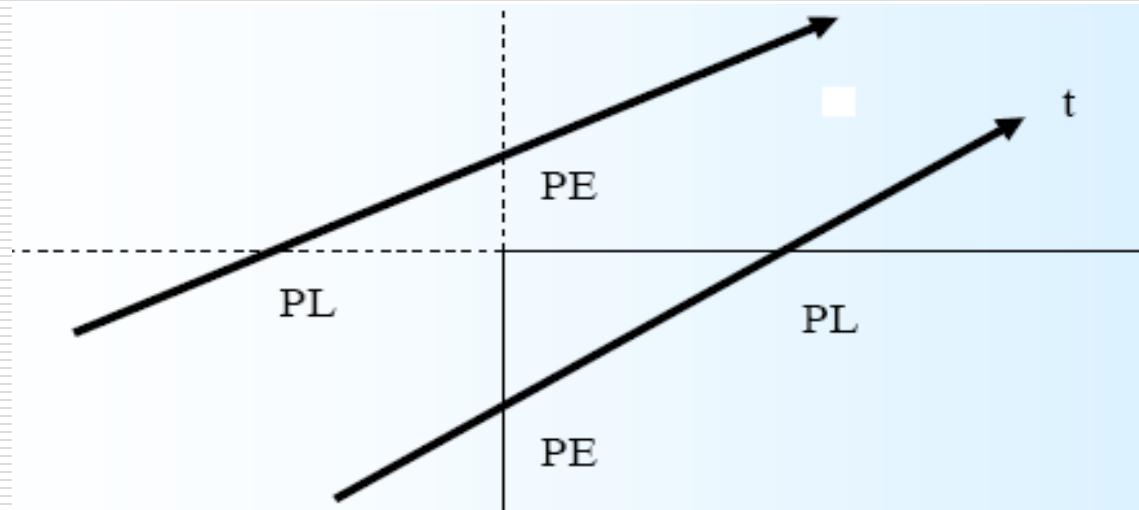
$$\Rightarrow t = -\mathbf{N}_j \cdot (P_0 - \mathbf{P}_{EJ}) / \mathbf{N}_j \cdot D$$

Xác định hướng

$t < 0$: điểm **đi vào** vùng clip (PE)

$t > 0$: **đi ra** vùng clip (PL)

- ☐ Sắp xếp các PE và PL lần lượt theo giá trị của t
- ☐ $PL < PE$: không có giao điểm
- ☐ Vẽ từ PE \rightarrow PL

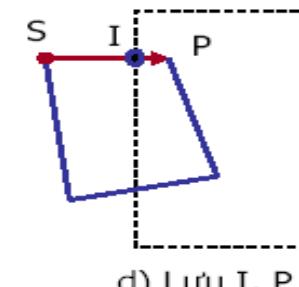
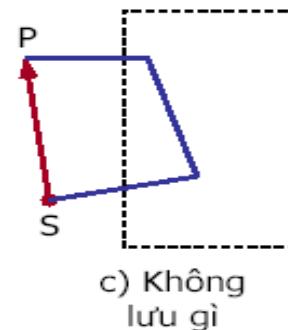
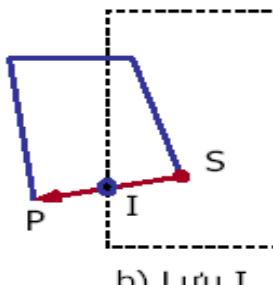
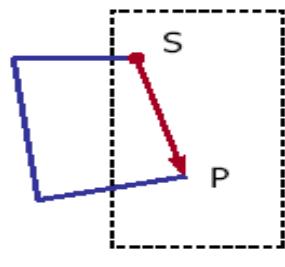


Thuật toán cắt xén vùng

- Tác giả: Sutherland-Cohen
- Input:
 - chữ nhật cắt xén
 - Vùng là đa giác được xác định bởi trật tự các cặp tọa độ.
- OutPut:
 - Các đa giác nằm trong cửa sổ cắt xén
- Ý tưởng thuật toán
 - So sánh lần lượt các đỉnh đa giác với biên cửa sổ
 - Đỉnh nằm ngoài, loại bỏ ngay
 - Đỉnh nằm trong, lưu trữ lại làm kết quả
 - Tính giao điểm của các cạnh đa giác vùng với cạnh chữ nhật

Thuật toán cắt xén vùng

- Duyệt lần lượt (theo chiều kim đồng hồ) các cạnh đa giác
- Nếu đỉnh duyệt xuất phát từ trong cửa sổ theo cạnh đa giác đi ra ngoài cửa sổ: lưu trữ giao của cạnh đa giác với biên cửa sổ
- Nếu đường đi từ ngoài vào trong cửa sổ: lưu trữ đỉnh đa giác và giao điểm
- Thí dụ xét hai đỉnh đa giác S và P:



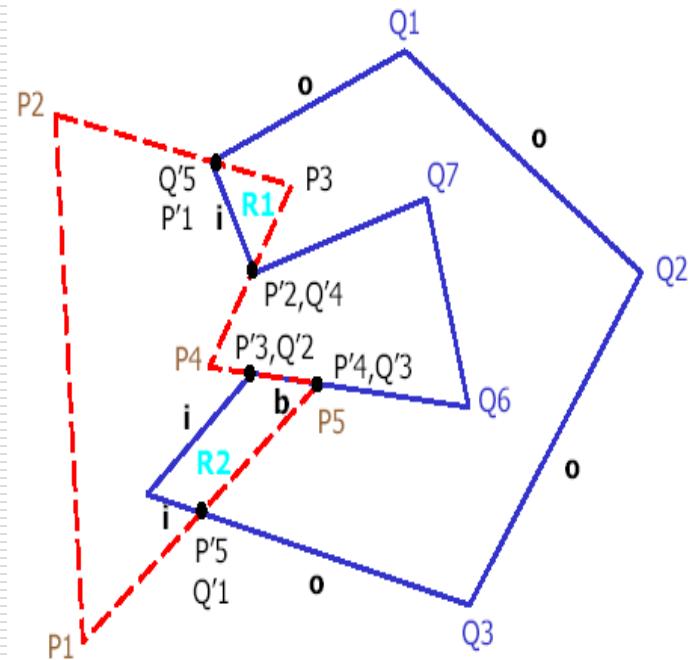
Cắt xén vùng bằng đa giác

- Thuật toán do Clamer Schutte (Đại học Delft, Hà Lan) đề xuất

- Cho trước hai đa giác P và Q không có lỗ hổng, không tự cắt và đỉnh của chúng được sắp xếp theo chiều kim đồng hồ
- Hãy tìm đa giác thuộc tập $P \cup Q, P \setminus Q, Q \setminus P$

- Các bước của thuật toán

- Phân lớp các đỉnh của hai đa giác vào 2 danh sách: gán vào mỗi đỉnh giá trị i(inside), o(outside) hay b(boundary) phụ thuộc vào vị trí của nó so với đa giác kia
 - $Pv = <(P1, o), (P2, o), (P3, i), (P4, o), (P5, b)>$
 - $Qv = <(Q1, o), (Q2, o), (Q3, o), (Q4, i), (Q5, b), (Q6, o), (Q7, o), (Q8, b), (Q9, b)>$

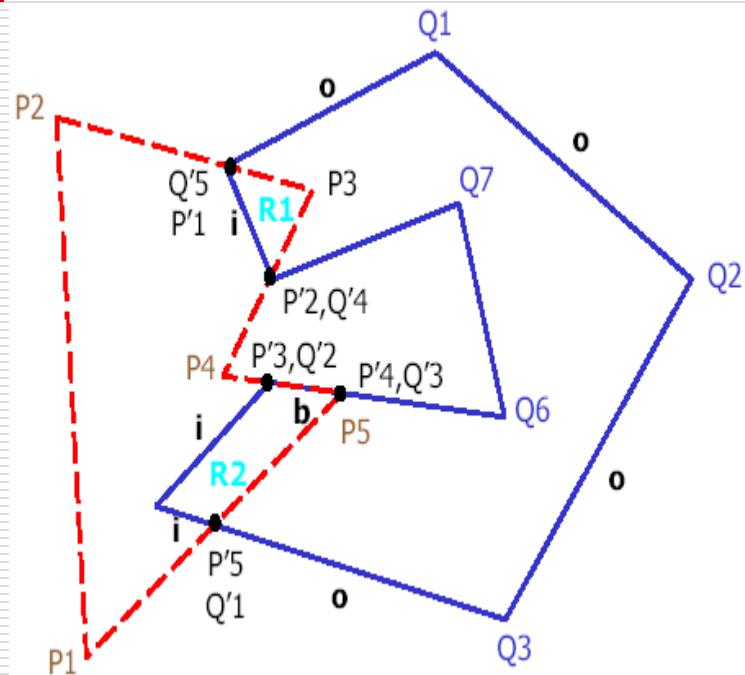


Cắt xén vùng bằng đa giác

- Tìm giao các cạnh của hai đa giác P, Q.
- Mỗi giao điểm được xen vào Pv hay Qv và đánh dấu b(boundary) để có danh sách mới

$Pef = \langle (P1, o), (P2, o), (P'1, b), (P3, i), (P'2, b), (P4, o), (P'3, b), (P5, b), (P'4, b), (P'5, b) \rangle$

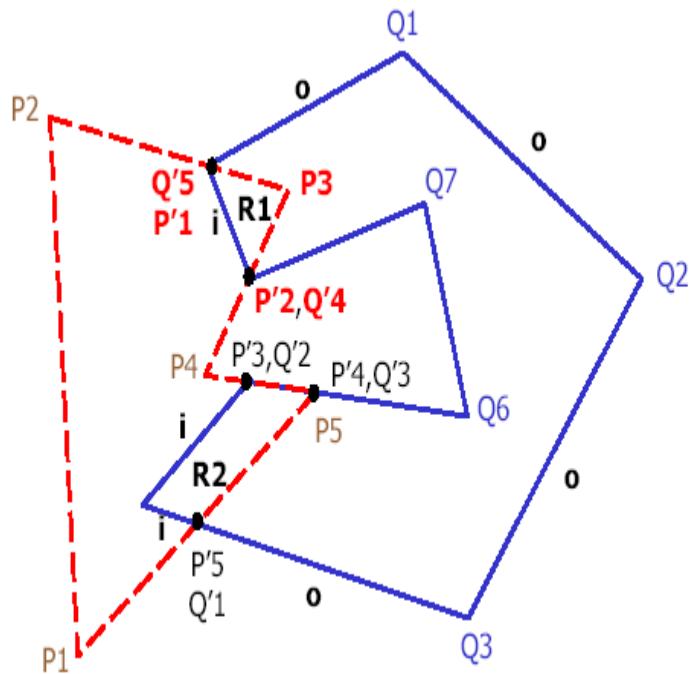
$Qef = \langle (Q1, o), (Q2, o), (Q3, o), (Q'1, b), (Q4, i), (Q5, b), (Q'2, b), (Q'3, b), (Q6, o), (Q7, o), (Q8, b), (Q'4, b), (Q'5, b), (Q9, b) \rangle$



Cắt xén vùng bằng đa giác

□ Danh sách Pef và Qef:

- $Pef = \langle (P1, o), (P2, o), (P'1, b), (P3, i), (P'2, b), (P4, o), (P'3, b), (P5, b), (P'4, b), (P'5, b) \rangle$
- $Qef = \langle (Q1, o), (Q2, o), (Q3, o), (Q'1, b), (Q4, i), (Q5, b), (Q'2, b), (Q'3, b), (Q6, o), (Q7, o), (Q8, b), (Q'4, b), (Q'5, b), (Q9, b) \rangle$





QUAN SÁT 3D

Quan sát ba chiều

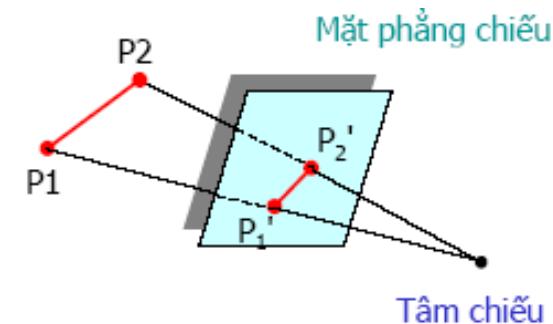
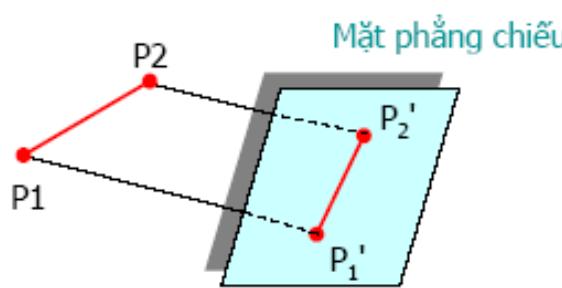
- Quan sát 2D:
 - Chuyển đổi các tọa độ 2D trong mặt phẳng thế giới thực thành các tọa độ 2D trong mặt phẳng thiết bị
- Quan sát 3D:
 - Có nhiều lựa chọn hiển thị: quan sát từ đáy, sườn, trên hay từ bên trong đối tượng...
 - Phức tạp hơn
 - Đối tượng 3D phải được mô tả trên mặt phẳng dẹt của thiết bị

Giới thiệu

- Các họa sỹ, kỹ sư, kiến trúc sư giải quyết sự phức tạp trong biểu diễn một đối tượng trong không gian 3D lên một môi trường trung gian 2D - đó là bài toán chiếu.
 - Phép chiếu là phép chuyển đổi những điểm trong hệ tọa độ n chiều thành thành những điểm trong hệ tọa độ có số chiều nhỏ hơn n.
 - Thực tế, đi sâu vào nghiên cứu các đối tượng n chiều bằng cách chiếu chúng lên mặt phẳng hai chiều để biểu diễn.
 - chỉ giới hạn với các phép chiếu từ ba chiều (3D) vào hai chiều (2D).

Các phép chiếu cơ bản

- Hai phép chiếu đối tượng 3D sang 2D cơ bản
 - Chiếu song song (parallel projection)
 - Chiếu các điểm trêm đối tượng theo đường song song
 - Sử dụng nhiều trong đồ họa máy tính
 - Chiếu phối cảnh (perspective projection)
 - Chiếu các điểm trêm đối tượng theo đường hội tụ đến tâm chiếu
 - Sử dụng nhiều trong các trò chơi (cảm giác thực hơn)
- Các biến thể của hai loại trên



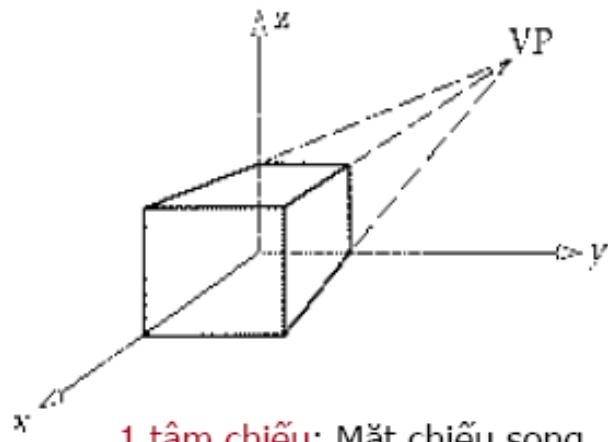
Phép chiếu

- Tác động trực quan của phép chiếu phối cảnh tương tự như trong chụp ảnh và hệ thống thị giác của con người (định luật phối cảnh xa gần - perspective foreshortening):
 - Kích thước của đối tượng qua phép chiếu nghịch đảo với khoảng cách từ đối tượng tới tâm chiếu (nếu mặt phẳng chiếu giữ nguyên).
 - Có khuynh hướng cho ảnh trông thật.
 - không dùng để biểu diễn chính xác một bề mặt hoặc kích thước của đối tượng
 - các góc được bảo toàn chỉ khi các mặt của đối tượng song song với mặt phẳng chiếu
 - các đường thẳng song song khi qua phép chiếu phối cảnh thường là không song song.

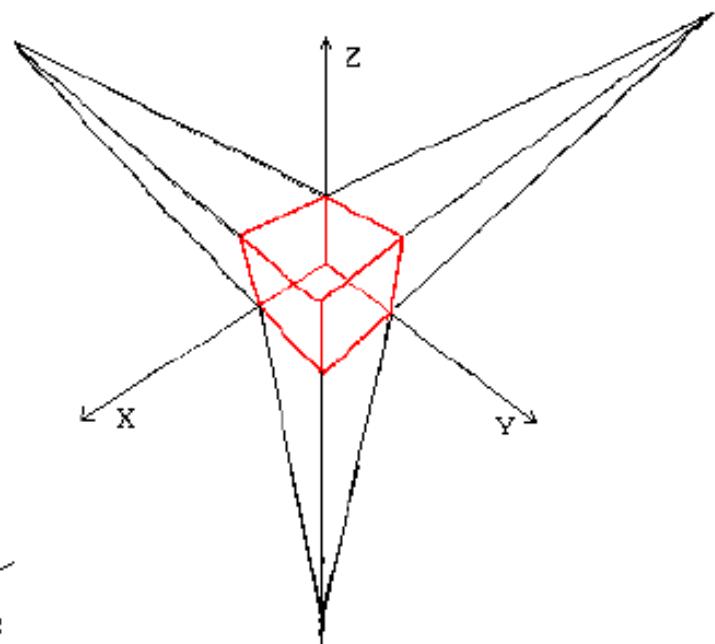
- Phép chiếu song song không cho hình ảnh thật của đối tượng bởi vì nó không có định luật phối cảnh xa gần.
 - Sử dụng để xác định kích thước một cách chính xác
 - Các đường thẳng song song khi qua phép chiếu vẫn song song với nhau.
 - các góc được bảo toàn chỉ khi các mặt của đối tượng song song với mặt phẳng chiếu

Phép chiếu phối cảnh

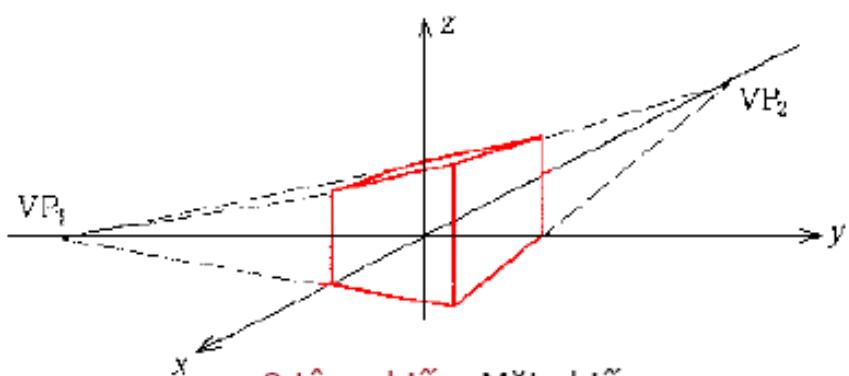
- Nguyên tắc người họa sĩ dùng để vẽ phối cảnh một đối tượng ba chiều.
 - Mắt là tâm chiếu
 - bức vẽ hay mặt phẳng chứa bức vẽ được gọi là mặt phẳng quan sát.
 - Điểm ảnh được xác định tại giao của tia chiếu (projector - tia vẽ từ một điểm trên đối tượng tới tâm chiếu) với mặt phẳng quan sát.
- Tập đường thẳng // qua phép chiếu sẽ giao nhau tại 1 điểm → điểm ảo (vanishing points) → đặc trưng của phép chiếu



1 tâm chiếu: Mặt chiếu song
song với hai trục tọa độ



3 tâm chiếu: Mặt chiếu
không song song với bất kỳ
trục tọa độ nào



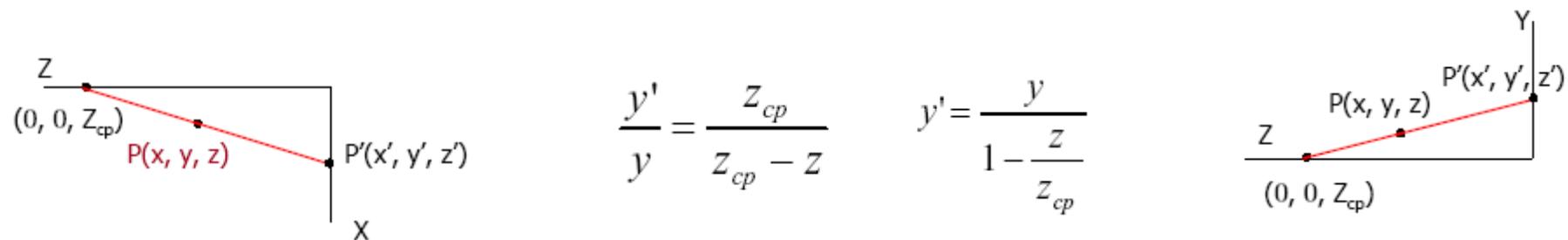
2 tâm chiếu: Mặt chiếu song
song với một trục tọa độ

Mô tả toán học của phép chiếu phối cảnh

- Xác định bởi tâm chiếu và mặt phẳng quan sát.
 - Mặt phẳng quan sát được xác định bởi điểm nhìn tham chiếu (view reference point) R_0 và pháp tuyến của mặt phẳng quan sát (view plane normal) N .
 - P trên đối tượng (object point) được xác định trong tọa độ thế giới thực tại vị trí (x, y, z) .
→ xác định tọa độ điểm ảnh $P'(x', y', z')$ (image point)
- Phép chiếu phối cảnh chuẩn là phép chiếu có mặt phẳng quan sát là xy , tâm chiếu tại $C(0, 0, z_{cp})$ theo chiều dương của trục z .
- Giả sử P là điểm trong không gian, P' là hình chiếu của P trên mặt chiếu. Tính $P'(x', y', z')$?

■ Quan sát trục y về gốc tọa độ:

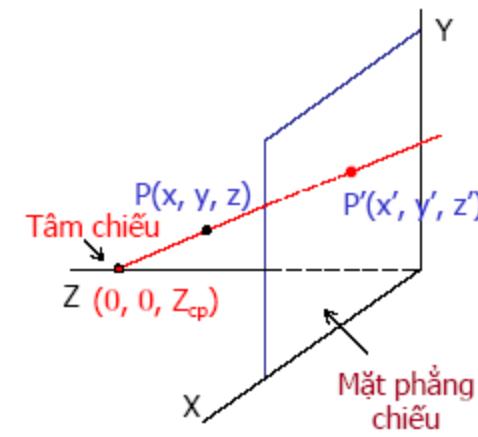
- Xét hai tam giác đồng dạng có



■ Quan sát theo trục x về gốc tọa độ:

$$\frac{x'}{x} = \frac{z_{cp}}{z_{cp} - z}$$

$$x' = \frac{x}{1 - \frac{z}{z_{cp}}}$$

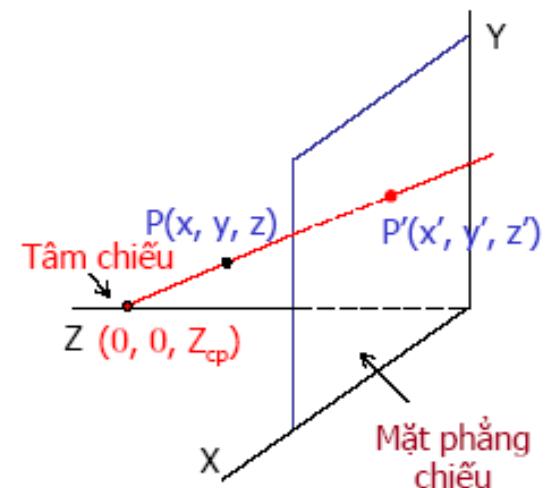


$$P' = \begin{bmatrix} x' & y' & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{1 - \frac{z}{z_{cp}}} & \frac{y}{1 - \frac{z}{z_{cp}}} & 0 & 1 \\ 0 & 0 & 0 & 1 - \frac{z}{z_{cp}} \end{bmatrix} = \begin{bmatrix} x & y & 0 & \left(1 - \frac{z}{z_{cp}}\right) \end{bmatrix}$$

$$= \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{z_{cp}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ma trận biến đổi cho chiếu phôi cảnh sẽ là

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{z_{cp}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Phép chiếu song song

- Người phác thảo, kỹ sư tạo ra các bản vẽ của một đối tượng bảo toàn hình dạng và tỷ lệ.
- Điểm ảnh được xác định là giao của mặt phẳng quan sát với tia chiếu vẽ từ một điểm trên đối tượng và có hướng xác định.
 - Hướng của phép chiếu (direction of projection) xác định cho tất cả các tia chiếu.
 - Phép chiếu trực giao (orthographic projections) được đặc trưng bởi hướng chiếu vuông góc với mặt phẳng chiếu.
 - Hướng chiếu song song với một trục chính tạo ra phép chiếu mặt, bằng, cạnh.
 - Phép chiếu trực lượng (axonometric) là phép chiếu vuông góc trong đó hướng chiếu không song song với bất kỳ trục chính nào.
 - Phép chiếu song song không vuông góc gọi là phép chiếu song song xiên (oblique parallel projections)
- Phép chiếu song song (parallel projective transformation) được xác định bởi vector hướng chiếu V và mặt phẳng quan sát.
 - Mặt phẳng quan sát được xác định bởi điểm nhìn tham chiếu R_0 và pháp tuyến của mặt phẳng quan sát N .

Chiếu song song

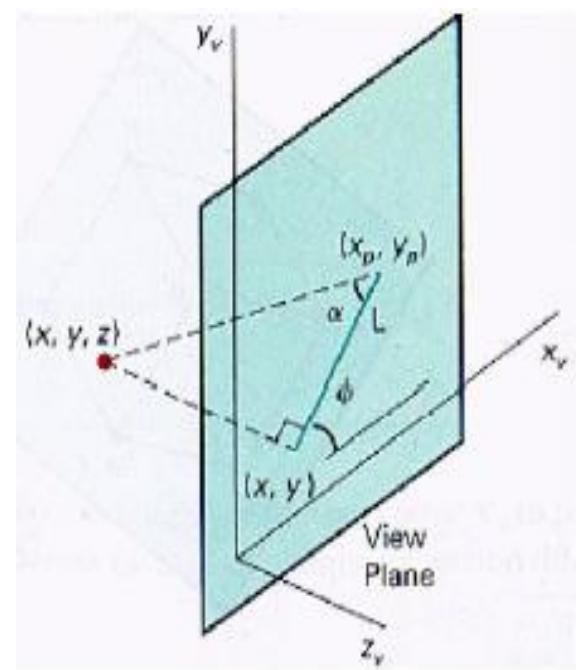
- Phép chiếu xiên (oblique): Tia chiếu không vuông góc với mặt chiếu
 - (x, y): tọa độ chiếu trực giao

$$x_p = x + L \cos \phi$$

$$y_p = y + L \sin \phi$$

$$z_p = 0$$

$$P_{parallel} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ L \cos \phi & L \sin \phi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



- Khi $\alpha=0$ thì $L=0$: có phép chiếu trực giao
 - L_1 : giá trị của L khi $z=1$
 - Khi $\alpha=45^\circ$ hay $L_1=1$: có phép chiếu xiên đều (cavalier)
 - Hình chiếu của đoạn song song hay vuông góc với mặt chiếu đều giữ nguyên độ lớn
 - Khi $\alpha=63.40$ hay $L_1=1/2$: có phép chiếu xiên cân (cabinet)
 - Hình chiếu của đoạn vuông góc với mặt chiếu bằng $\frac{1}{2}$ độ lớn ban đầu
 - Hình chiếu của đoạn song song với mặt chiếu -> giữ nguyên độ lớn

- Xác định ma trận của phép chiếu đối tượng mặt phẳng Q đi qua gốc tọa độ và có vector pháp tuyến $n = i + j + k$

Thực hành đọc ảnh

- System::Drawing::Bitmap^ temp = gcnew System::Drawing::Bitmap(filename);
- int w=temp->Width;
- int h=temp->Height;
- System::Drawing::Rectangle rect =
System::Drawing::Rectangle(0, 0, w, h);
- System::Drawing::Imaging::BitmapData^
bmpData =
- temp->LockBits(rect,
System::Drawing::Imaging::ImageLockMode::ReadWrite, temp->PixelFormat);
■ unsigned char* data= (unsigned char*)
bmpData->Scan0.ToPointer();

Đổi không gian màu

- `for (int i=0;i<w*h*3;i+=3)`
- `{`
- `data[i]=data[i]*0.299+data[i+1]*0.587+data[i+2]*0.114;`
- `data[i+1]=data[i]*0.584-data[i+1]*0.274-data[i+2]*0.31;`
- `data[i+2]=data[i]*0.211-data[i+1]*0.522+data[i+2]*0.311;`
- `}`
- `temp->UnlockBits(bmpData);`

Thực hành lập trình trên máy tính

Bài 1

□ Viết chương trình vẽ đa giác theo mảng tọa độ cho trước. Các cạnh của đa giác được nối với nhau theo thuật toán vẽ đường thẳng DDA

Bài 2

□ Viết chương trình biểu diễn
thuật toán vẽ đường tròn
theo giải thuật trung điểm

Bài 3

- Viết chương trình vẽ tam giác ABC theo các tọa độ cho trước, từ một điểm P bất kỳ (tọa độ cho trước) bên trong tam giác ABC bằng thuật toán loang lõi tô màu ta giác đó theo màu đường biên của tam giác
-

Bài 4

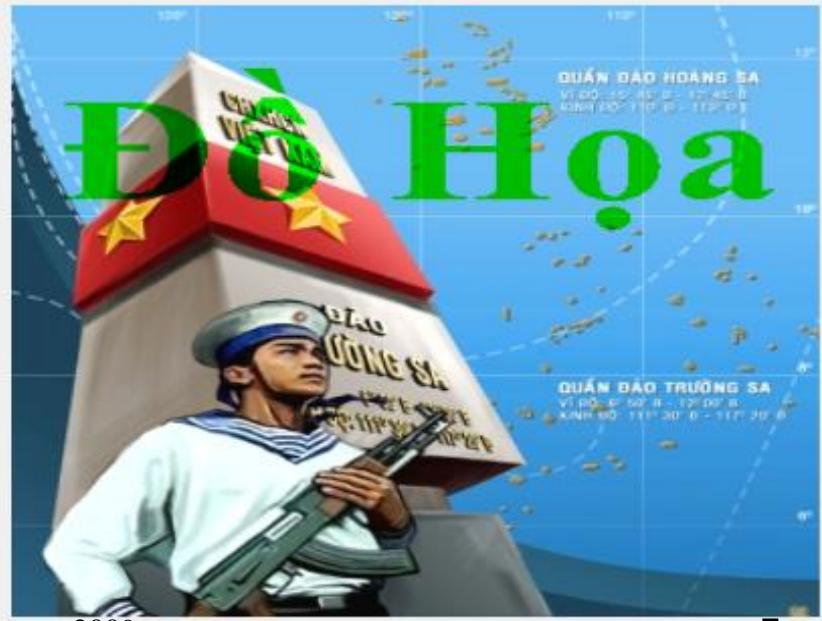
- Viết chương trình biểu diễn các thuật toán xoay, tịnh tiến của đối tượng đồ họa cho dưới dạng một đoạn thẳng AB (tọa độ biết trước)
-

Ví dụ Đề bài thi cho lớp Tin học 41 (thời gian làm bài 15 phút)

- Khi xây dựng các ứng dụng đồ họa việc biểu diễn hình ảnh đa lớp (Multilayers) đôi khi rất cần thiết để tạo hiệu ứng sinh động và có thể biểu diễn nhiều thông tin về hình ảnh hơn. Trong môi trường đồ họa của .Net (C#, VC++.Net, VB.Net) lớp Graphics đã cung cấp sẵn hàm vẽ xâu ký tự lên vùng hình ảnh như ví dụ sau:
- **DrawString("Đồ Họa", new Font("Times New Roman", 50, FontStyle.Bold), new SolidBrush(Color.Red), 10, 40);**
- Tuy nhiên thành phần màu sắc của xâu chữ (trong ví dụ là Color.Red) đòi hỏi phải sử dụng một màu đồng nhất. Để khắc phục hạn chế này hãy sử dụng các kiến thức đã học để viết chương trình thực hiện việc vẽ xâu chữ trong suốt lên ảnh với mục đích giữ lại đặc điểm hình ảnh phía dưới xâu chữ.
- Trong đó xâu chữ được gọi là trong suốt nếu như màu sắc của các chữ cái tại từng điểm ảnh được xác định dựa trên màu nền bằng cách giữ nguyên thành phần màu Green của nền còn hai thành phần màu Blue và Red gán bằng 0

Yêu cầu chi tiết:

- Thủ với cả các trường hợp ảnh có kích thước không chẵn, ví dụ 391x260
- Kết quả thu được phải có dạng như sau



Trần Nguyên Ngọc - 2009

Giới thiệu về Computer vision

Course goals:

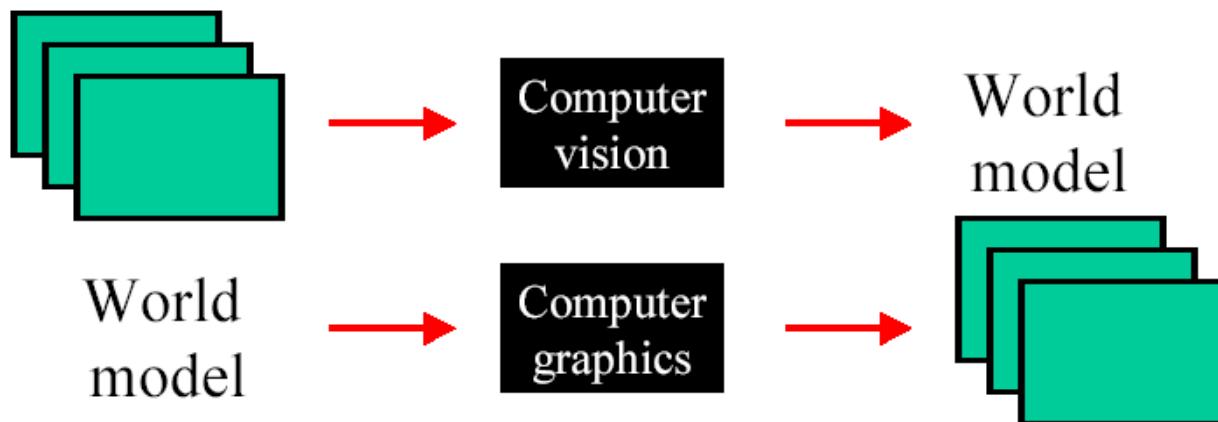
- Cover basic topics of computer vision, and introduce some fundamental approaches for computer vision research.
 - Imaging Geometry
 - Camera Modeling and Calibration
 - Filtering and Enhancing Images
 - Region Segmentation
 - Motion and Optical Flow
 - Color and Texture
 - Line and Curve Detection
 - Shape Analysis
 - Stereopsis
 - Structure from X

Syllabus

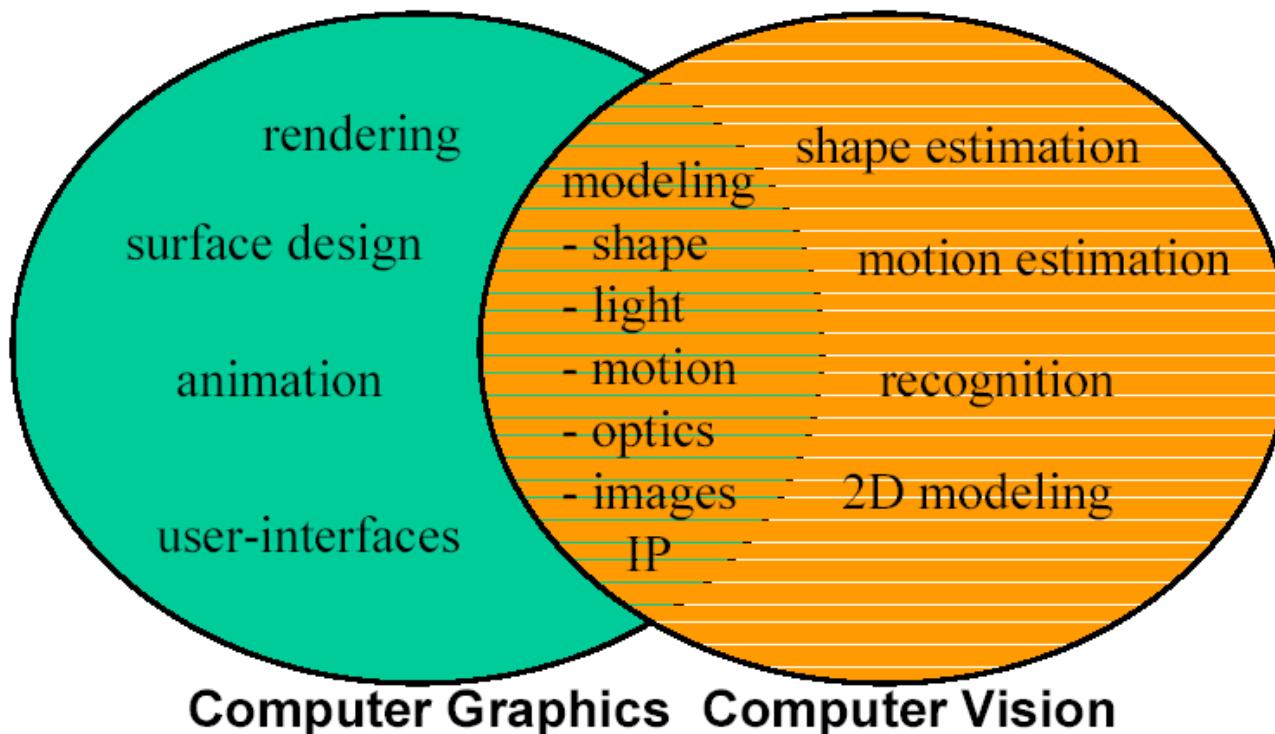
- Reference Text:
 - Emanuele Trucco, Alessandro Verri, "Introductory Techniques for 3-D Computer Vision", Prentice Hall, 1998.
 - Mubarak Shah, "Fundamentals of Computer Vision".
 - David A. Forsyth and Jean Ponce, "Computer Vision: A Modern Approach", Prentice Hall, 2003.

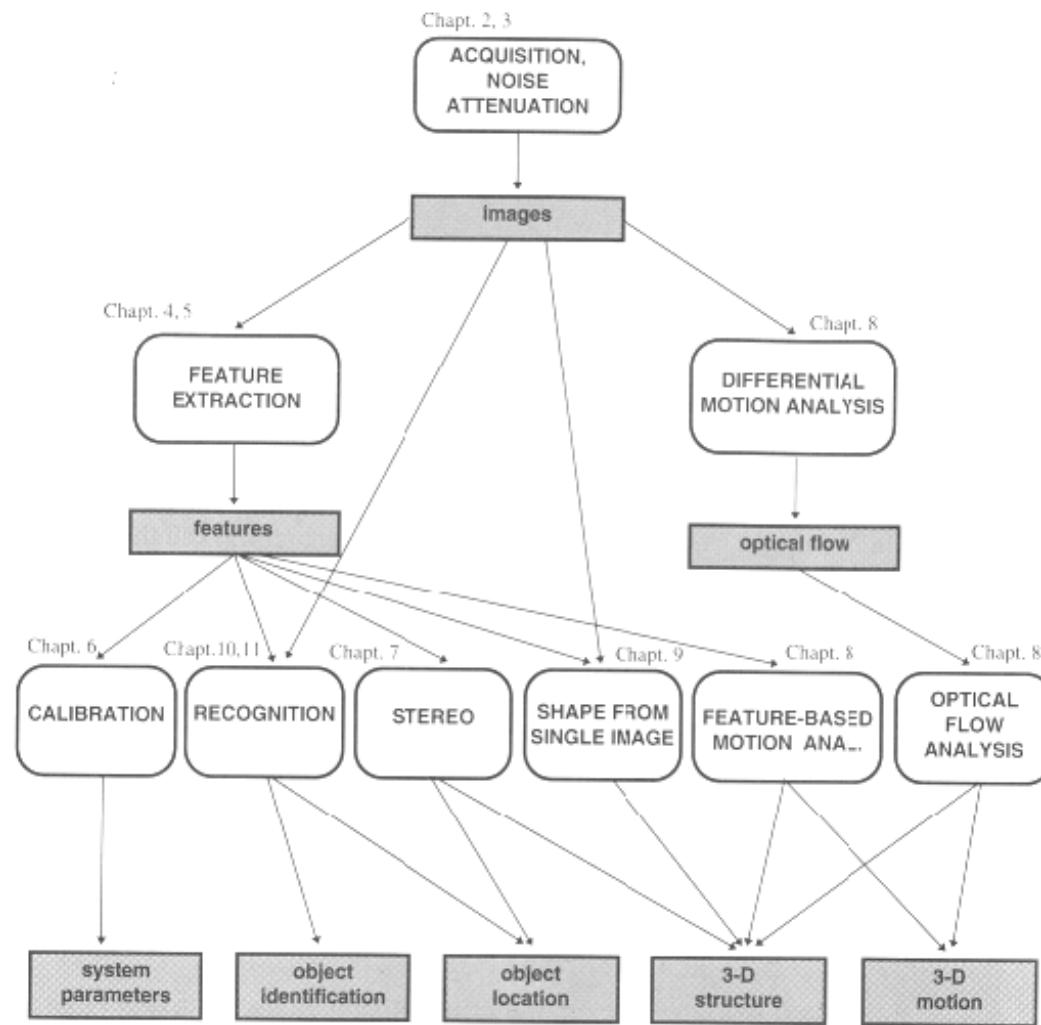
What is Computer Vision?

- Image and video understanding
- Computer emulation of human vision
- Inverse of computer graphics



Intersection Between Vision and Graphics





From Trucco and Verri, 1998

Figure 1.7 The book at a glance: method classes (white boxes), results (grey boxes), their interdependence, and where to find the various topics in this book.

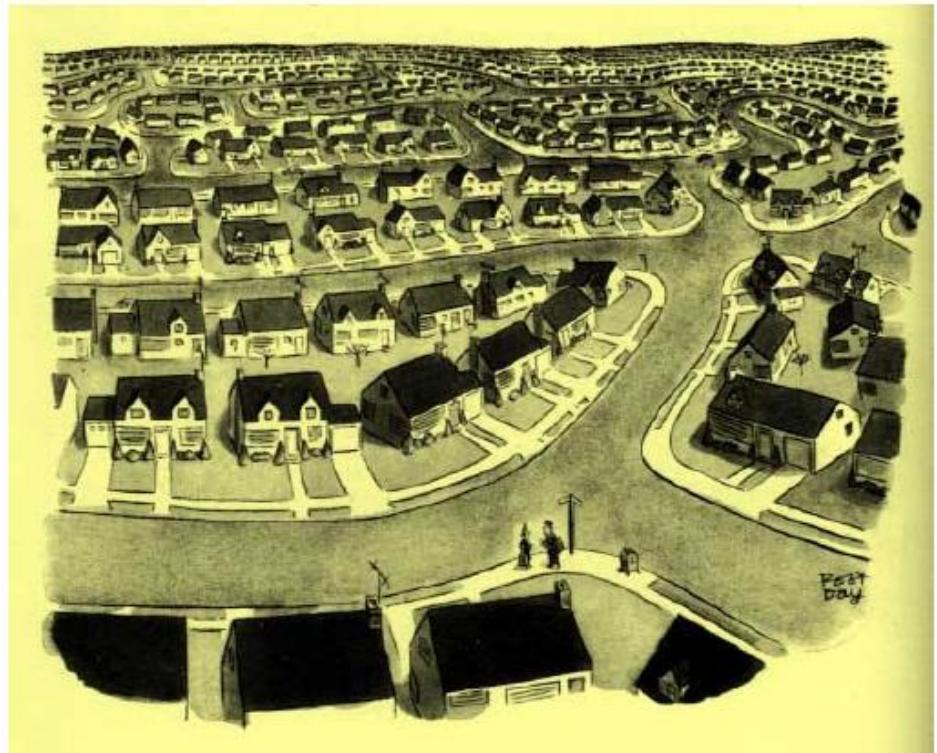
Related Disciplines

- Image process
- Computer graphics
- Pattern recognition
- Artificial intelligence
- Applied mathematics
- Learning
- ...

Course Overview:

What we will cover?

Image Geometry



Camera

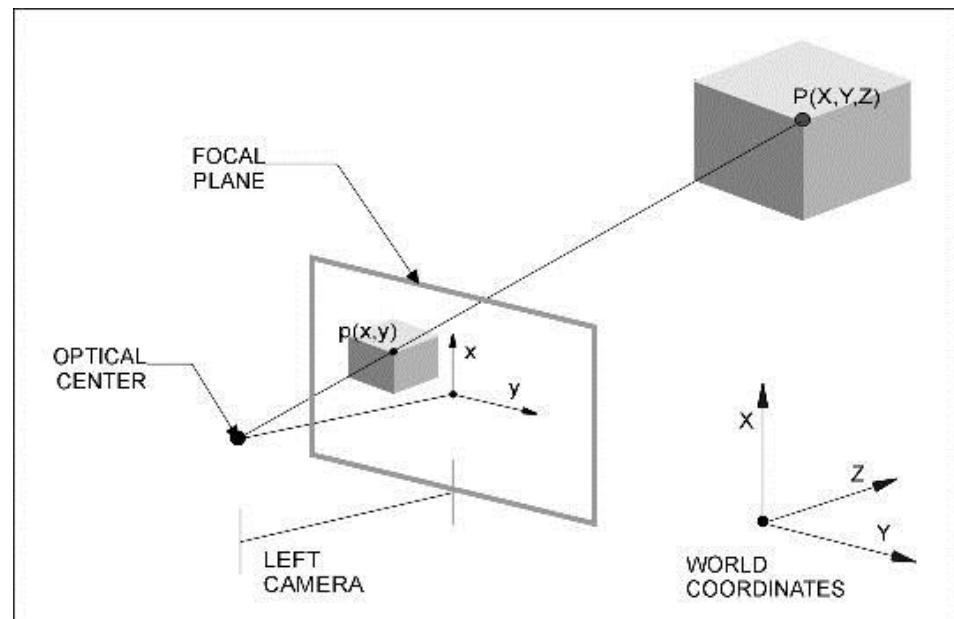
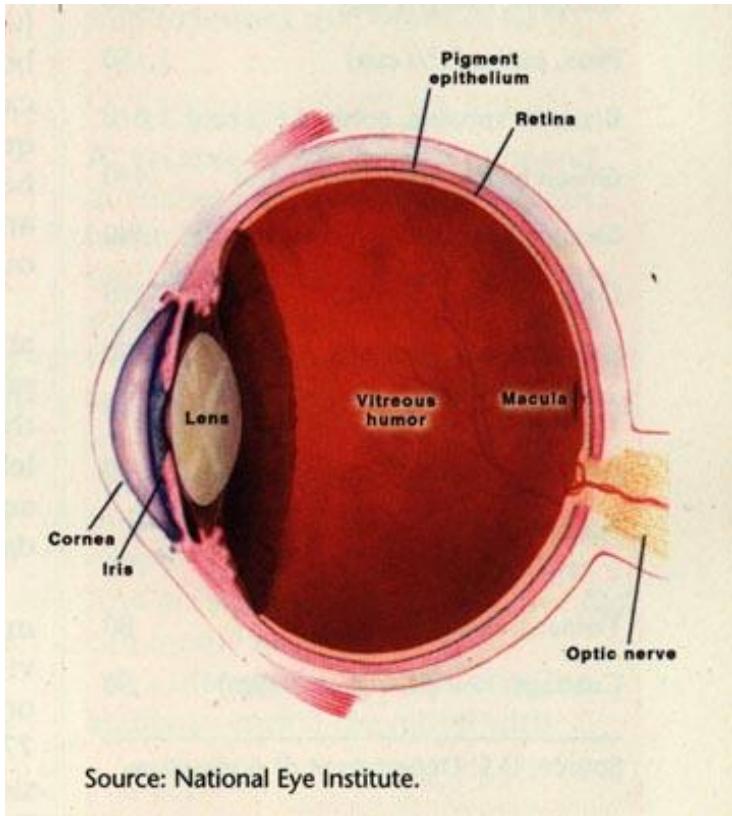


Image Filter and Enhance

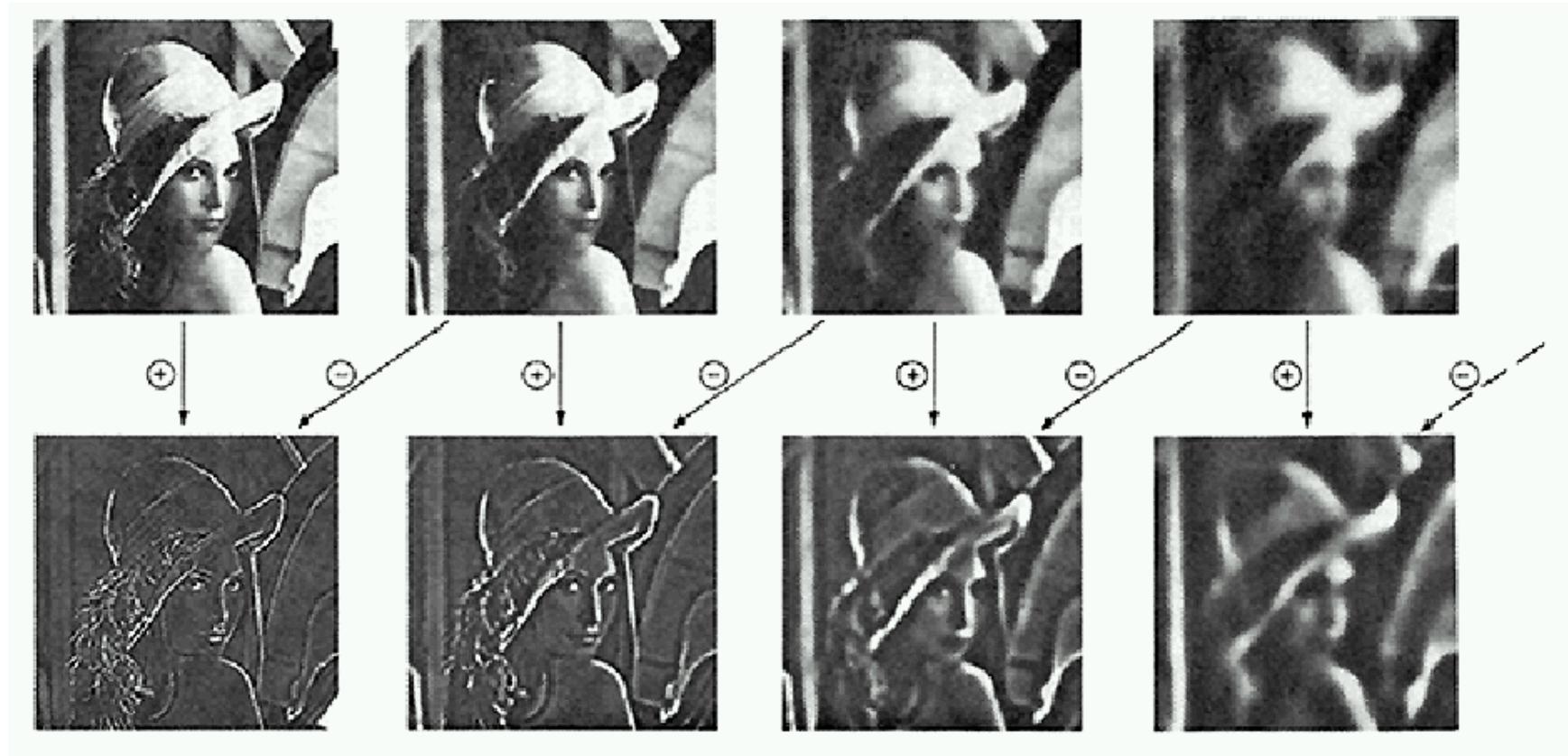


Image Warping



translation



rotation



aspect



affine



perspective



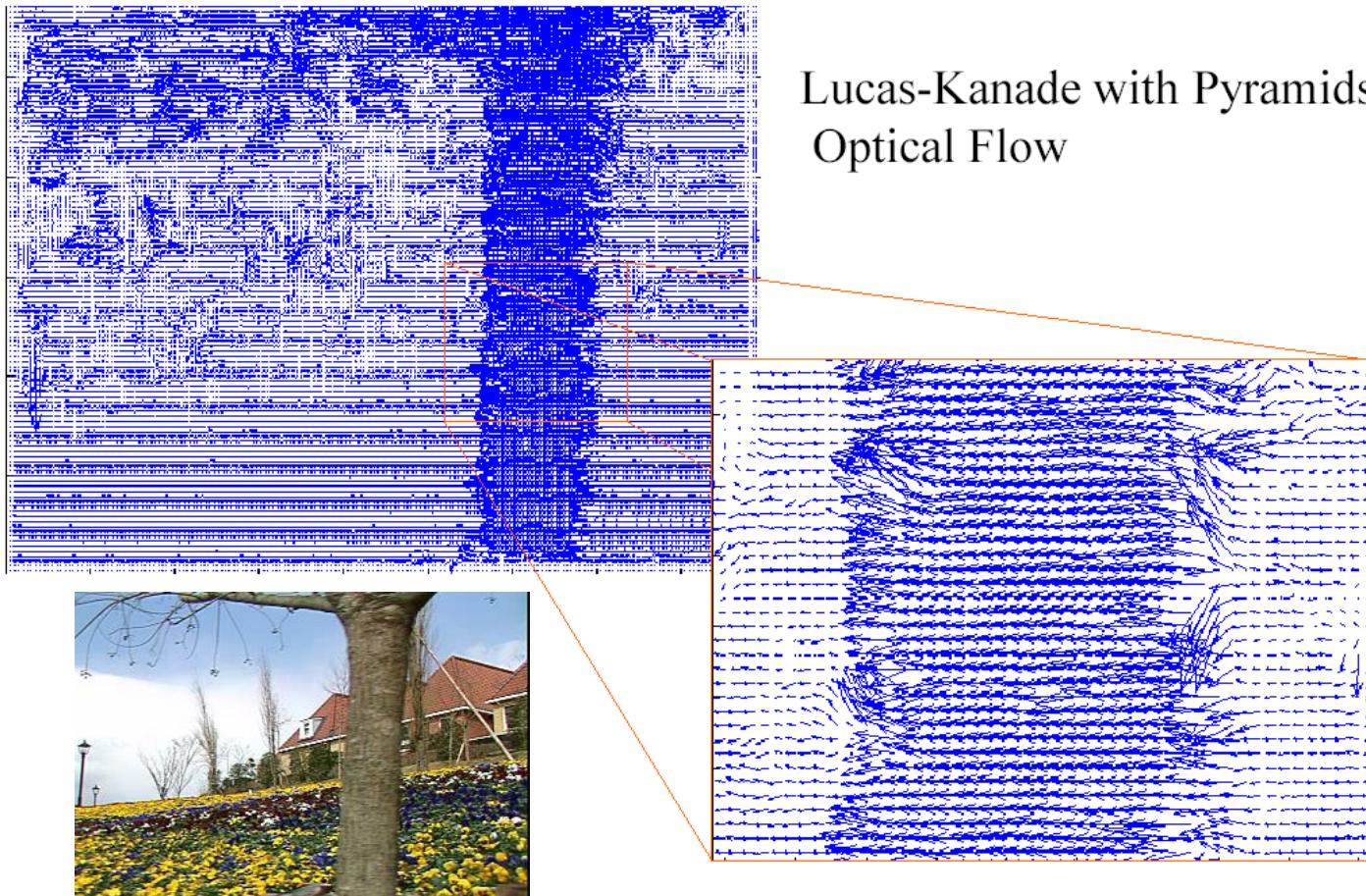
cylindrical

Edge Detector



Elder, J. H. and R. M. Goldberg. "Image Editing in the Contour Domain,"
Proc. IEEE: Computer Vision and Pattern Recognition, pp. 374-381, June, 1998.

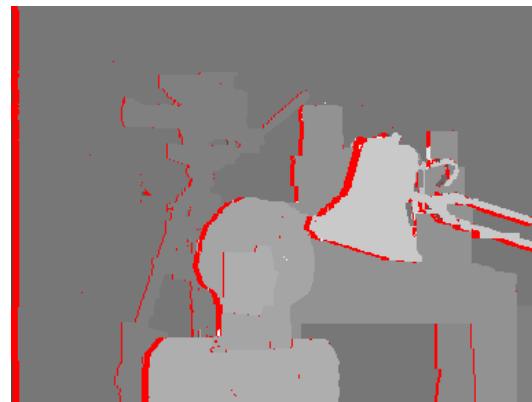
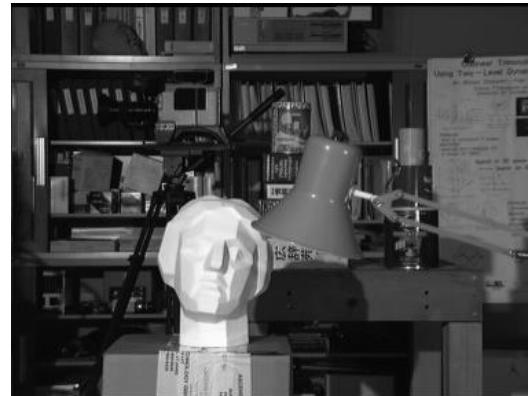
Optical Flow



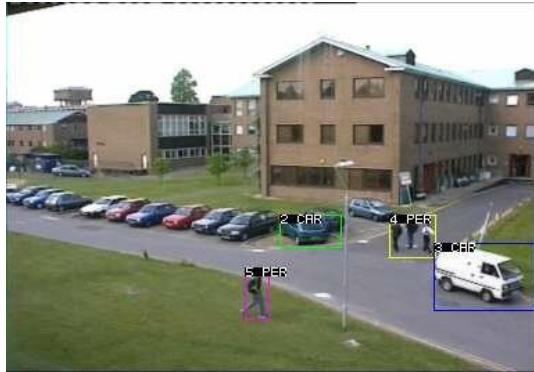
Panoramic Mosaic



Stereo



Tracking

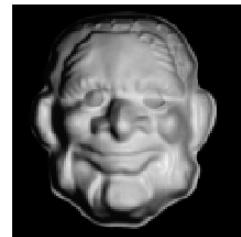


Results from Knight system in UCF



Contour tracking in UCF

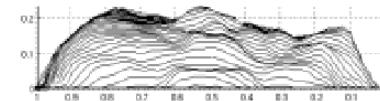
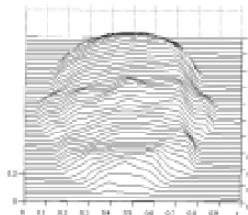
Shape from Shading



a)



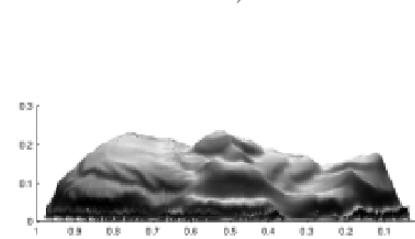
b)



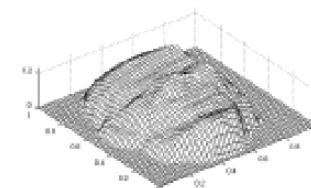
c)



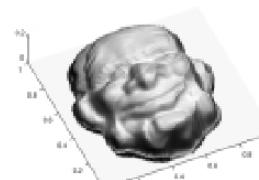
d)



e)

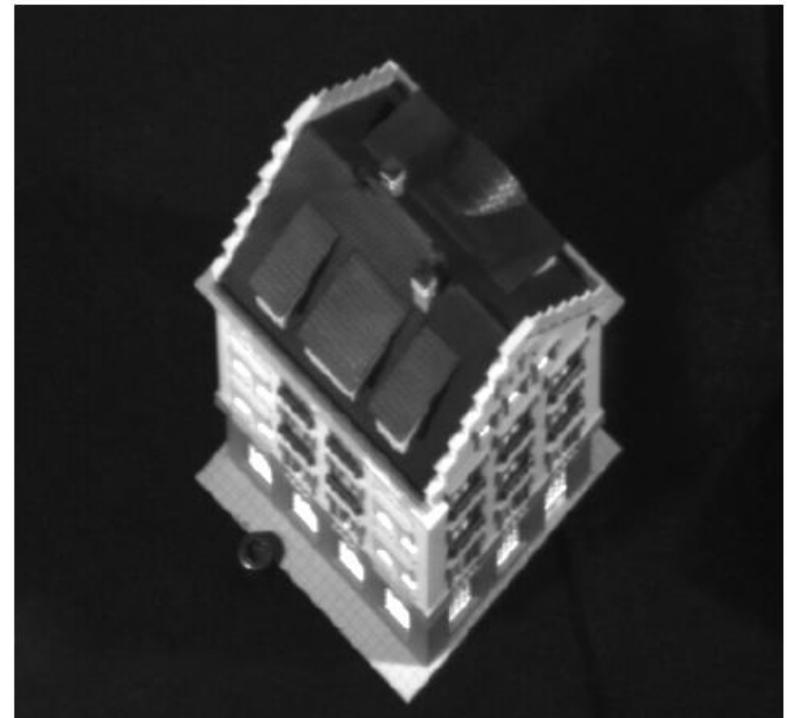


f)

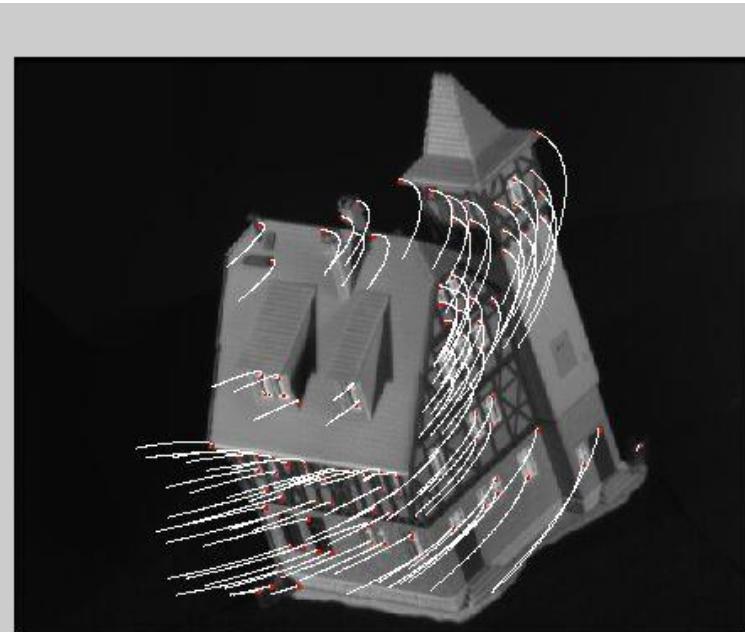
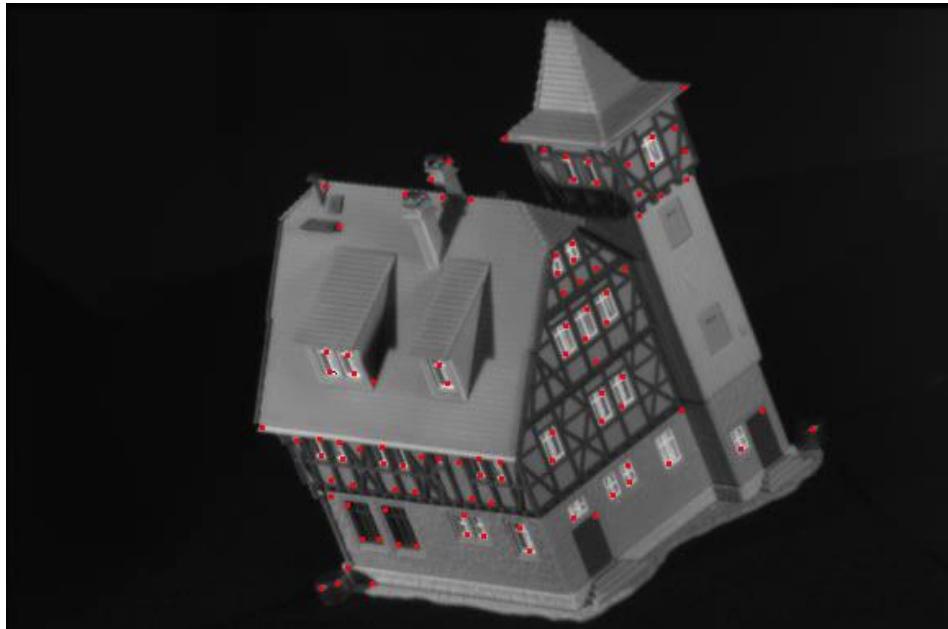


Emmanuel Prados, Olivier Faugeras, and Elisabeth Rouy, "Shape from Shading and Viscosity Solutions", ECCV 2002.

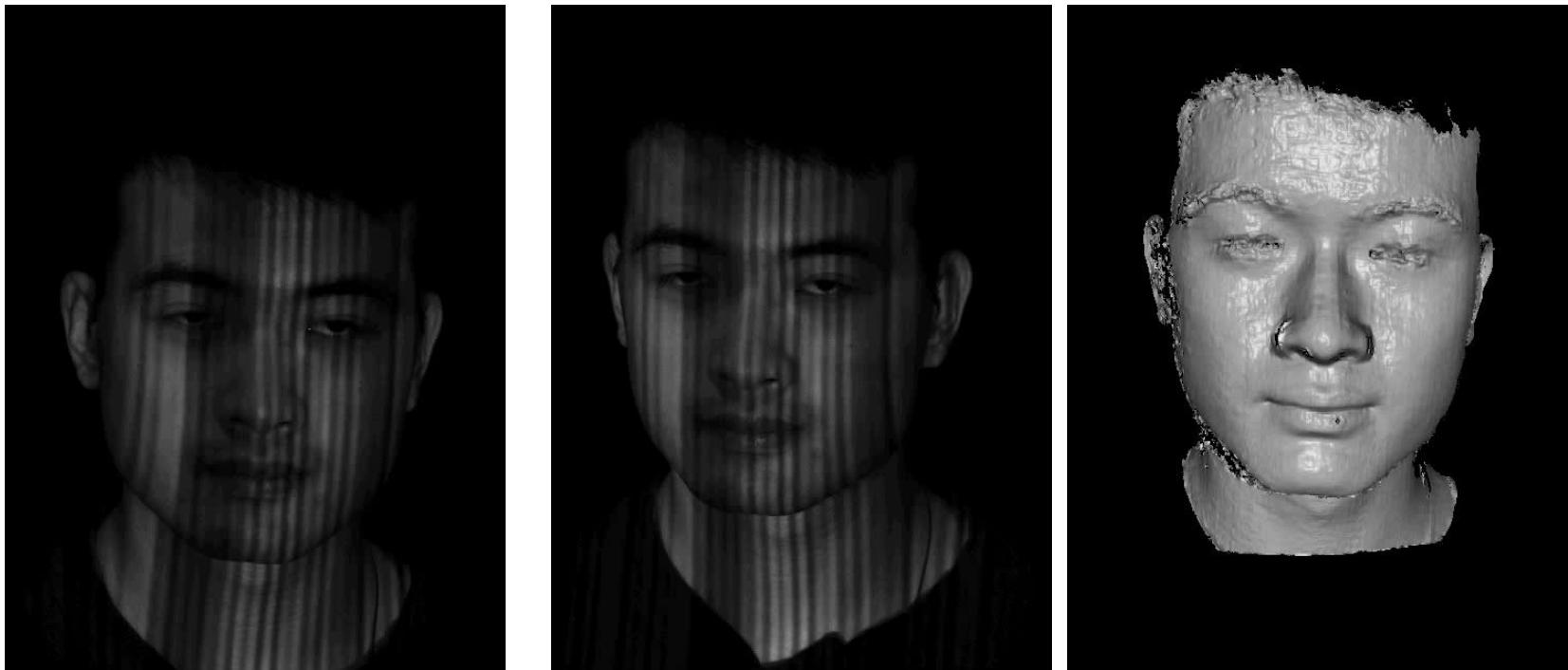
Structure from Motion



Point Correspondences



Structure lighting with Stereo



Li Zhang, Brian Curless, and Steven M. Seitz. Spacetime Stereo: Shape Recovery for Dynamic Scenes. *CVPR 2003*.

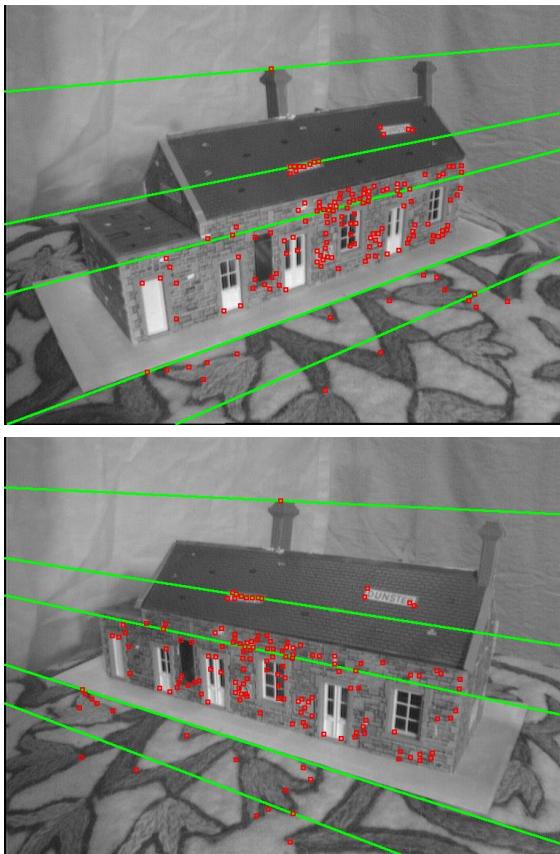
Tracking



Contour-based Tracking



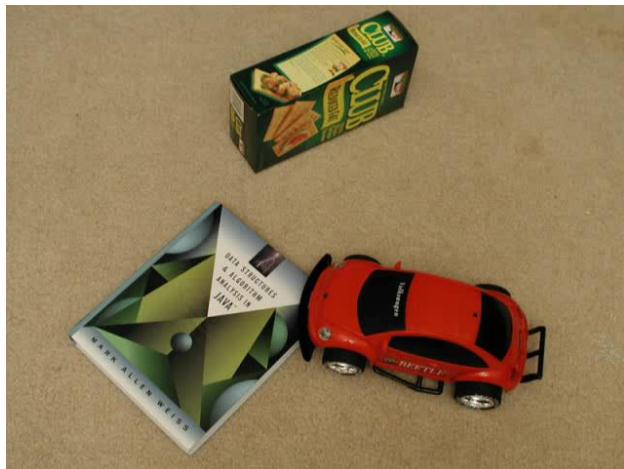
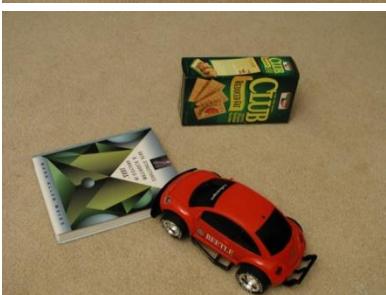
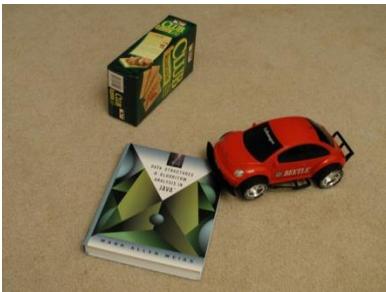
Wide Baseline Matching



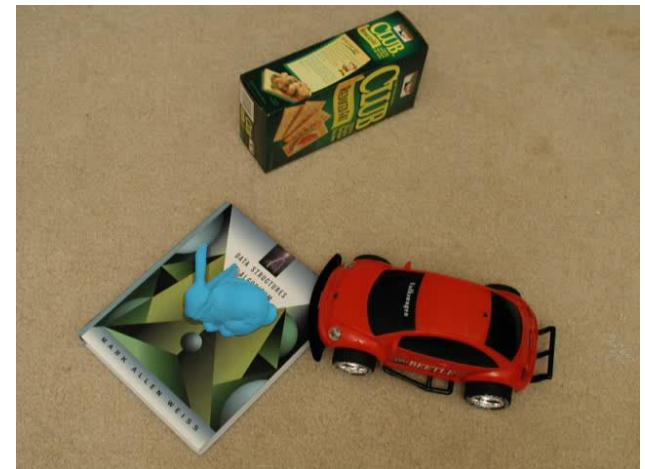
Dynamic View Morphing



Tri-view Morphing



Tri-view synthesis



Augmented reality

Motion Segmentation



Phép chuyển đổi quan sát
hai chiều

Computer Graphics

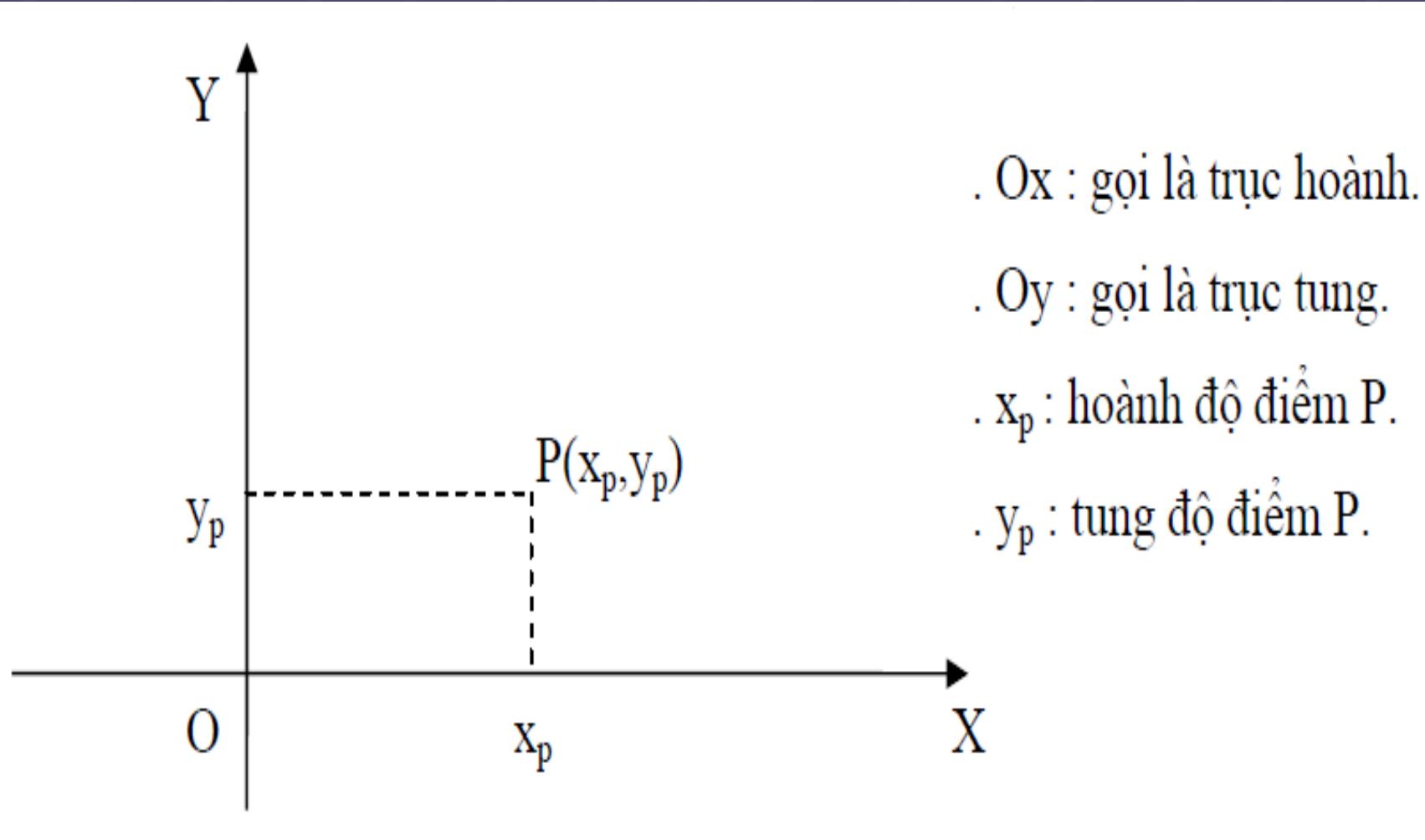
Why?

- Để biểu diễn hình ảnh của một vật, phải ánh xạ tọa độ của các điểm và đường tạo nên vật đó thành tọa độ tương ứng của thiết bị hoặc trạm làm việc nơi hình ảnh được hiển thị → phép chuyển đổi tọa độ - chuyển đổi quan sát (viewing transformation).

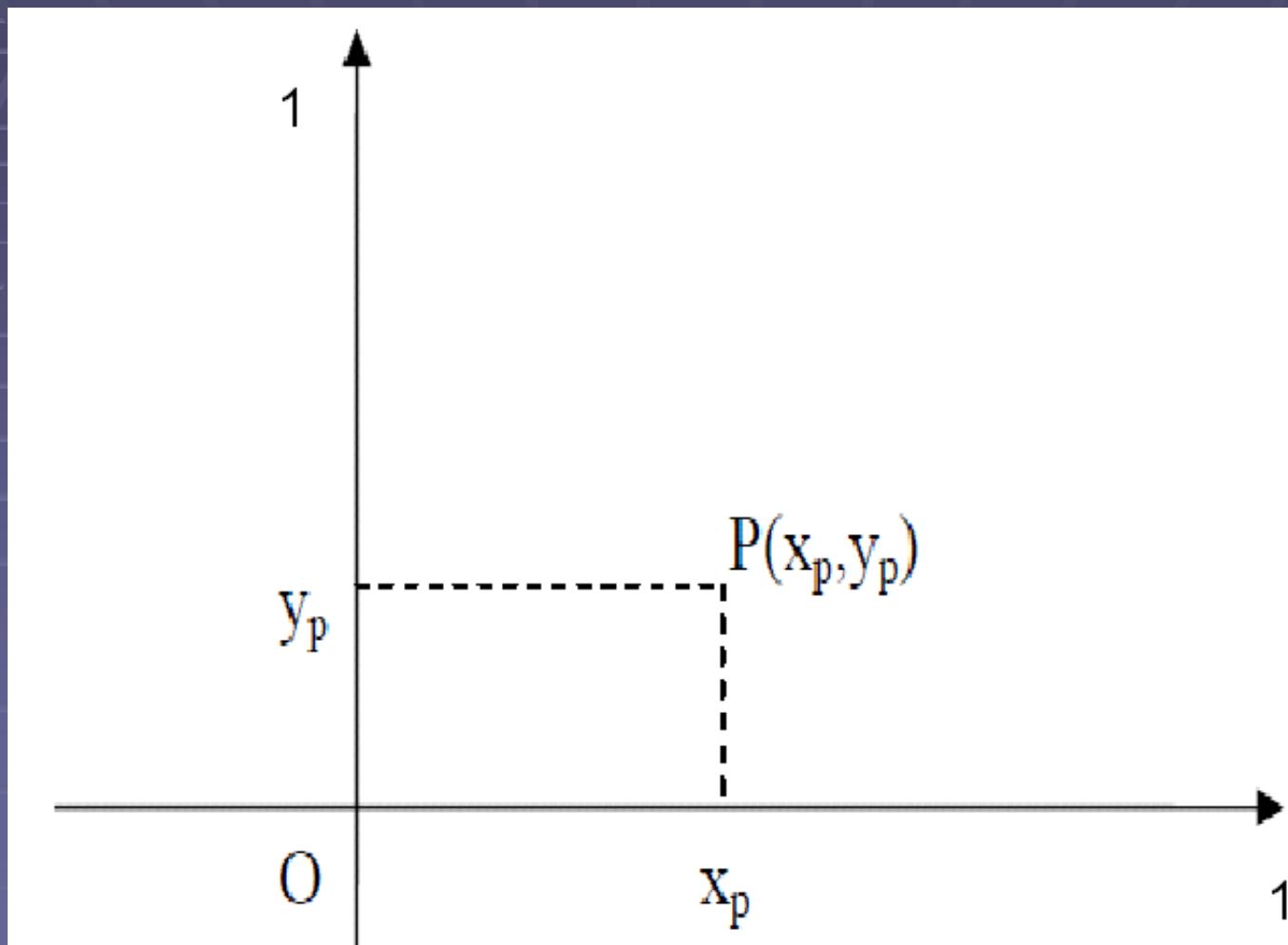
Các hệ tọa độ

- hệ tọa độ thế giới thực (World Coordinate System-WCS) là hệ tọa độ Decard để biểu diễn tọa độ của một vật.
- hệ tọa độ thiết bị vật lý (Physical device Coordinate System-PDCS) là hệ tọa độ tương ứng với thiết bị trong đó ảnh của vật hoặc mô hình được hiển thị.
- hệ tọa độ thiết bị chuẩn hóa (normalized device Coordinate System-NDCS) là hệ tọa độ tay phải trong đó vùng hiển thị của thiết bị tương ứng 1×1 với một hình chữ nhật đơn vị mà góc trên bên trái là gốc của hệ tọa độ.

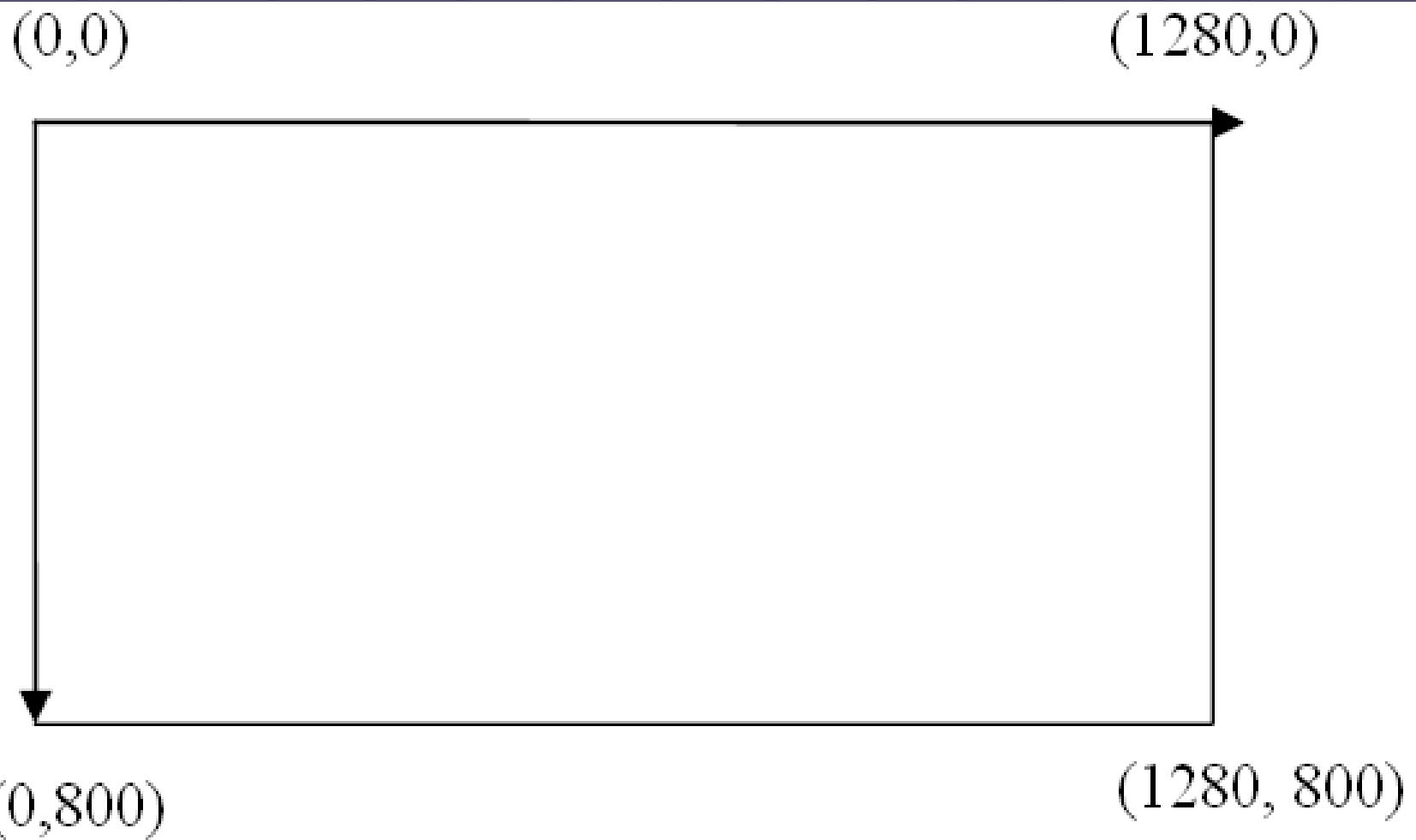
Hệ tọa độ thực



Hệ tọa độ chuẩn hóa



Hệ tọa độ màn hình

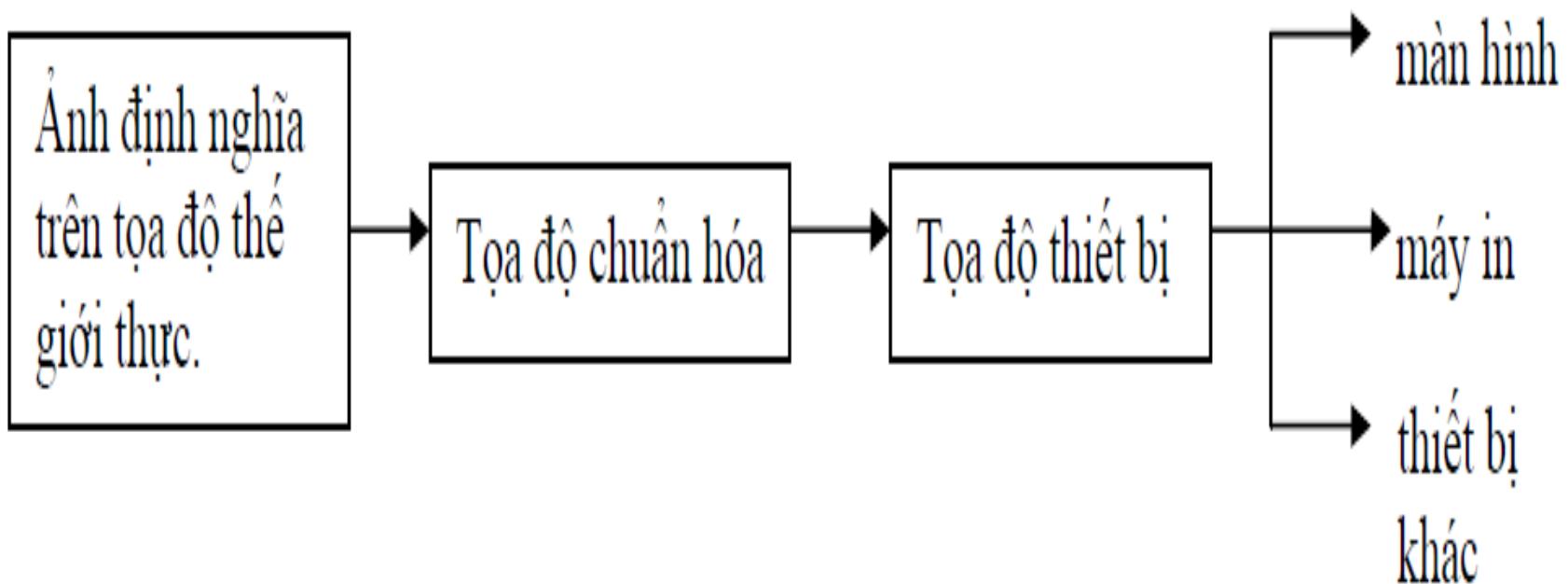


Chuyển đổi quan sát

- Chuyển đổi quan sát vật trong WCS thành PDCS:
 - Phép chuyển đổi chuẩn hóa (normalization transformation:N) ánh xạ tọa độ thế giới thực thành tọa độ thiết bị chuẩn hóa
 - Phép chuyển đổi trạm làm việc (workstation transformation: W) ánh xạ tọa độ thiết bị chuẩn hóa thành tọa độ thiết bị vật lý

$$V = W \cdot N$$

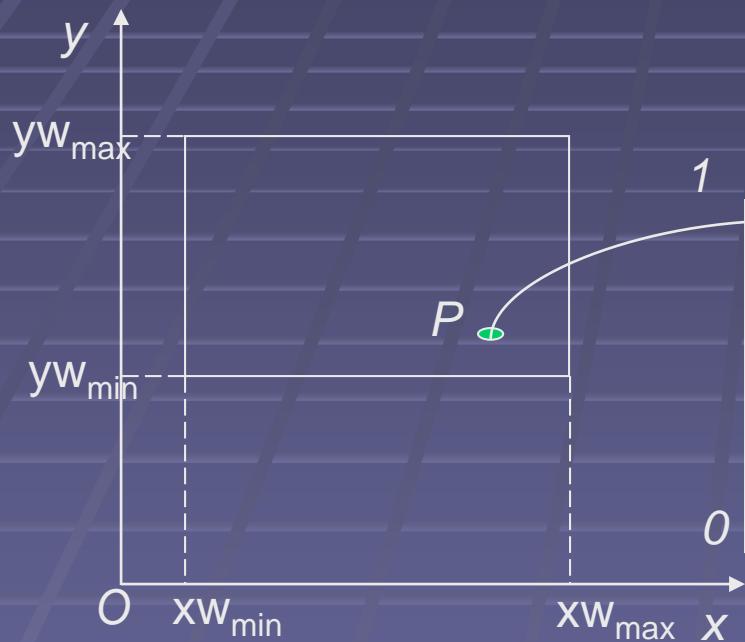
Quá trình chuyển đổi



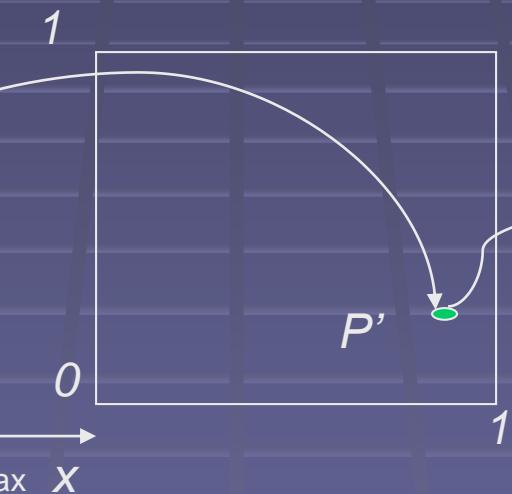
Cửa sổ và khung nhìn

- Drawback:
 - WCS là vô hạn (đầu phẩy động)
 - Vùng hiển thị của thiết bị có giới hạn
- chỉ làm việc trong vùng giới hạn trong WCS gọi là cửa sổ (window)
- Vùng hiển thị của thiết bị gọi là khung nhìn (viewport)

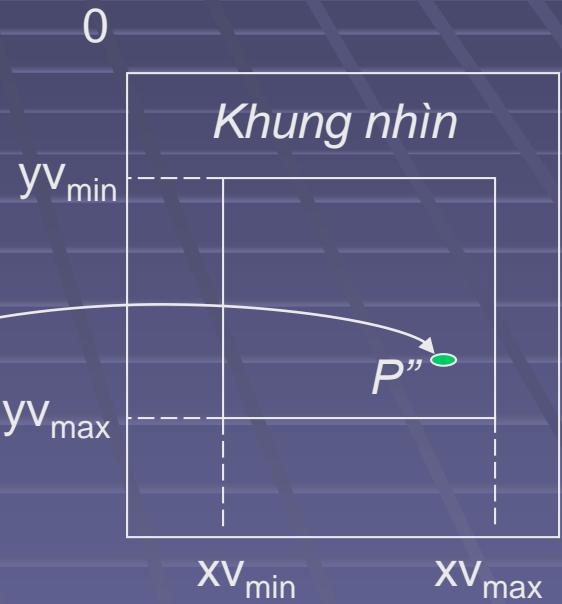
Cửa sổ và khung nhìn



Tọa độ thế giới thực



Tọa độ thiết bị chuẩn hóa



Tọa độ màn hình

Ma trận của phép biến đổi

- N (hoặc W)=

$$\begin{bmatrix} S_x & 0 & -S_x \cdot xw_{\min} + xv_{\min} \\ 0 & S_y & -S_y \cdot yw_{\min} + yv_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

- Trong đó

$$S_x = \frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}, S_y = \frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}}$$

- xv, yv chỉ tọa độ khung nhìn còn xw, yw chỉ tọa độ của sổ

Tỷ lệ co dãn

- Chuyển đổi quan sát bao gồm cả co dãn
 - Hình bị biến dạng hoặc co dãn không đều:
 - Hình tròn → ellipse
 - Hình vuông → hcn
 - Tỷ lệ co dãn a_w của cửa sổ = tỷ lệ co dãn của khung nhìn thì ko có biến dạng

$$a = \frac{x_{\max} - x_{\min}}{x_{\max} + x_{\min}}$$

Công cụ thực hành môn học

VC++ .Net

Visual Studio 2005

How to: Create Graphics Objects for Drawing

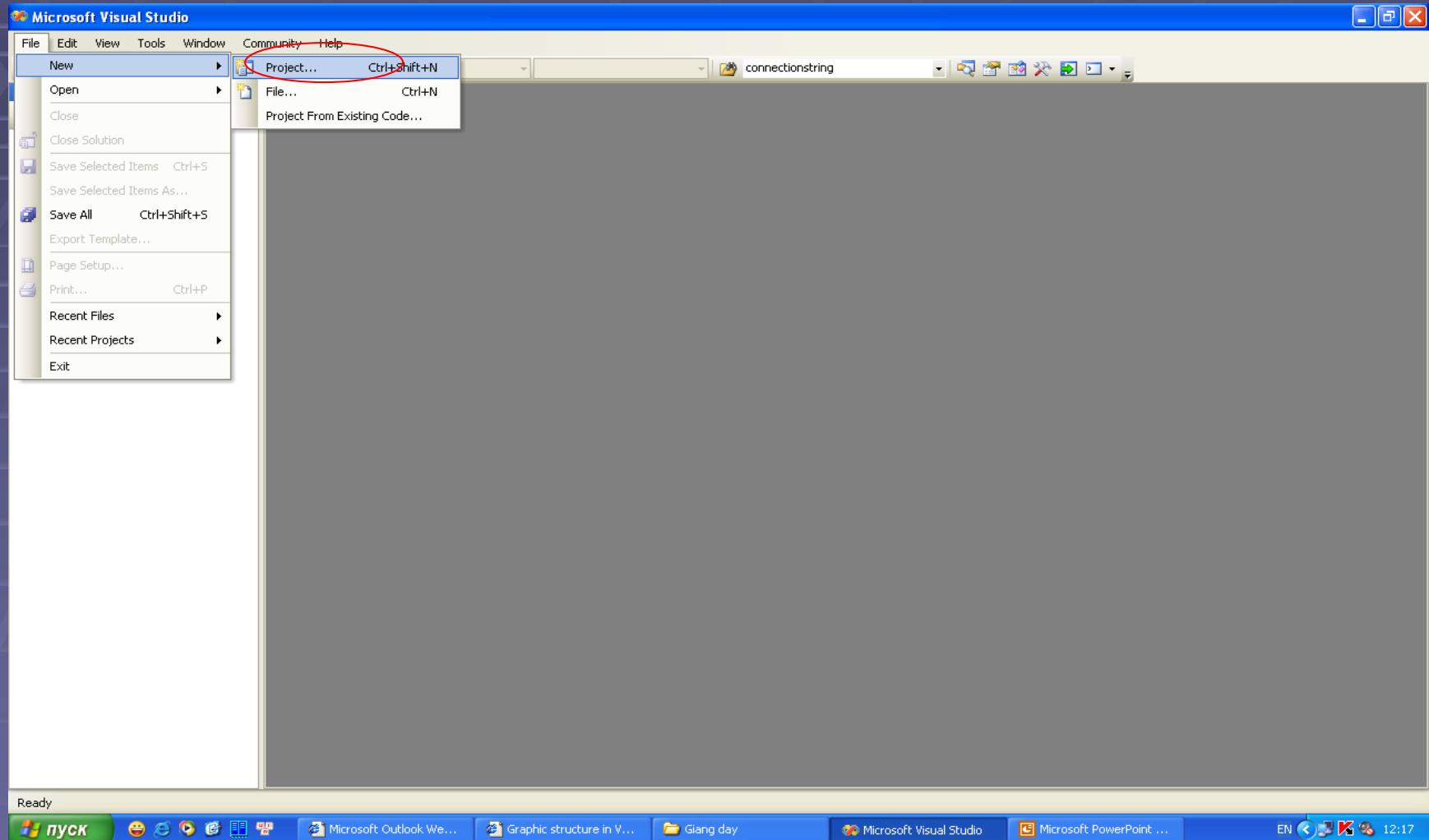
- Để tạo môi trường vẽ đồ họa trong .Net trước hết cần có đối tượng **Graphics** object. Đối tượng này có thể gắn vào các đối tượng điều khiển khác. Có 2 cách để làm việc với đối tượng đồ họa:
- Tạo mới **Graphics** object.
- Sử dụng **Graphics** object sẵn có để vẽ.

Tạo graphics object

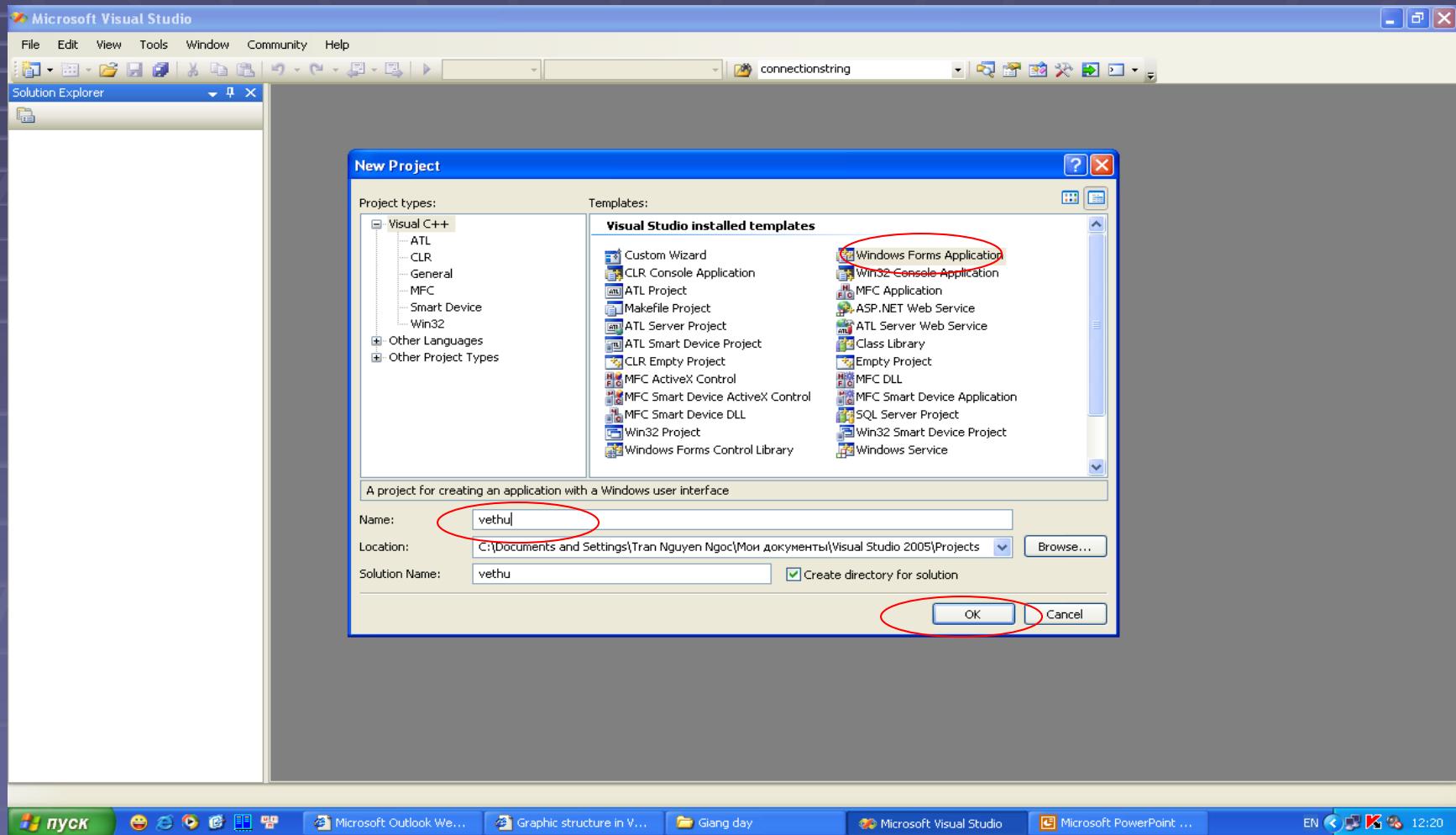
- Thừa kế lớp PaintEventArgs từ sự kiện Paint của Form hoặc lớp Control.
- Gọi phương thức CreateGraphics của Control or Form để vẽ lên các đối tượng đó.
- Tạo Graphics object từ bất kỳ đối tượng nào thừa kế từ lớp Image.

```
private: System::Void  
Form1_Paint(System::Object^ sender,  
System::Windows::Forms::PaintEventArgs^ e)  
{  
    Graphics^ ve= e->Graphics;  
    System::Drawing::Pen^ but =gcnew  
    System::Drawing::Pen(Color::Red,5);  
    ve->DrawLine(but,1,1,100,200);  
}
```

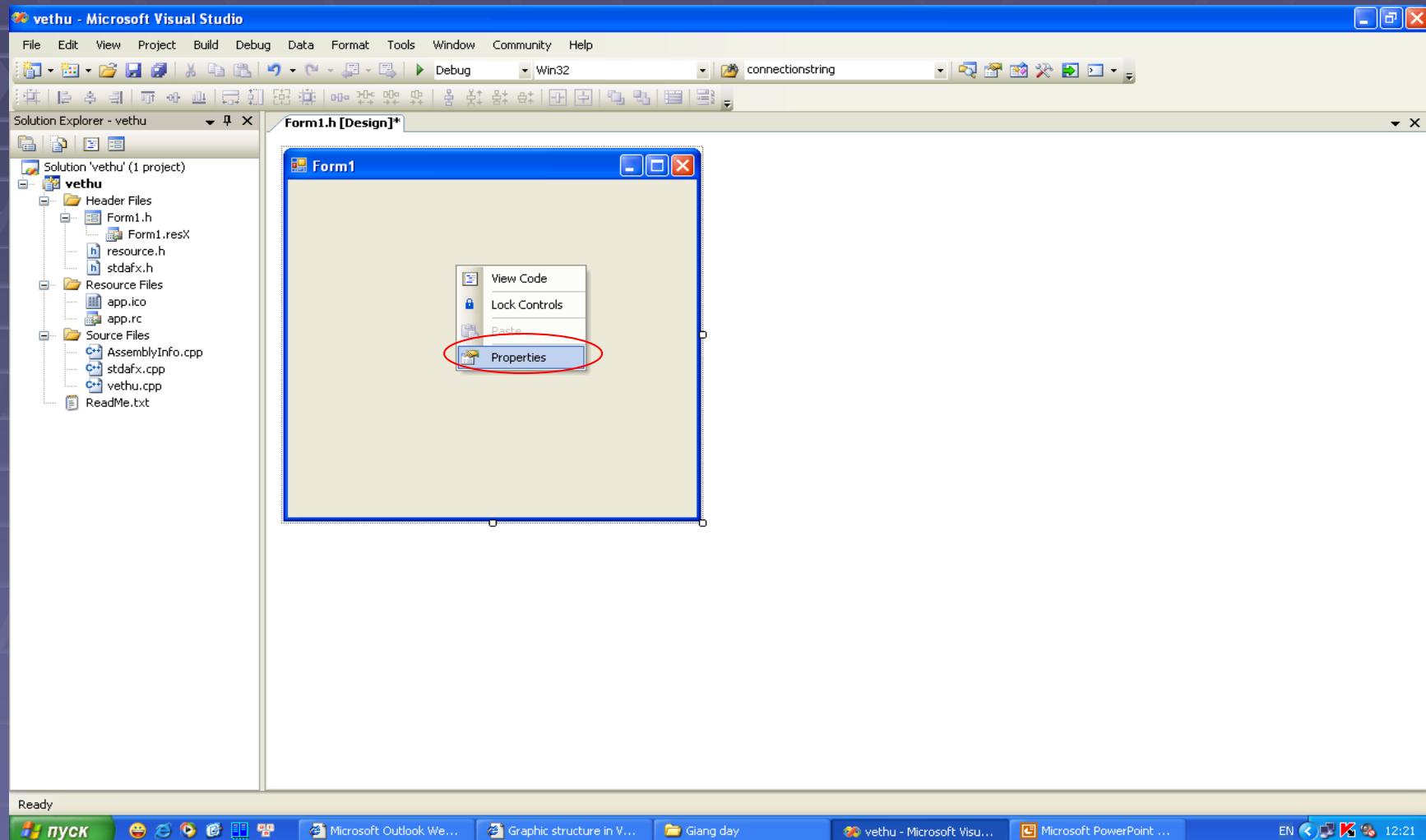
Bước 1-Chọn new Project



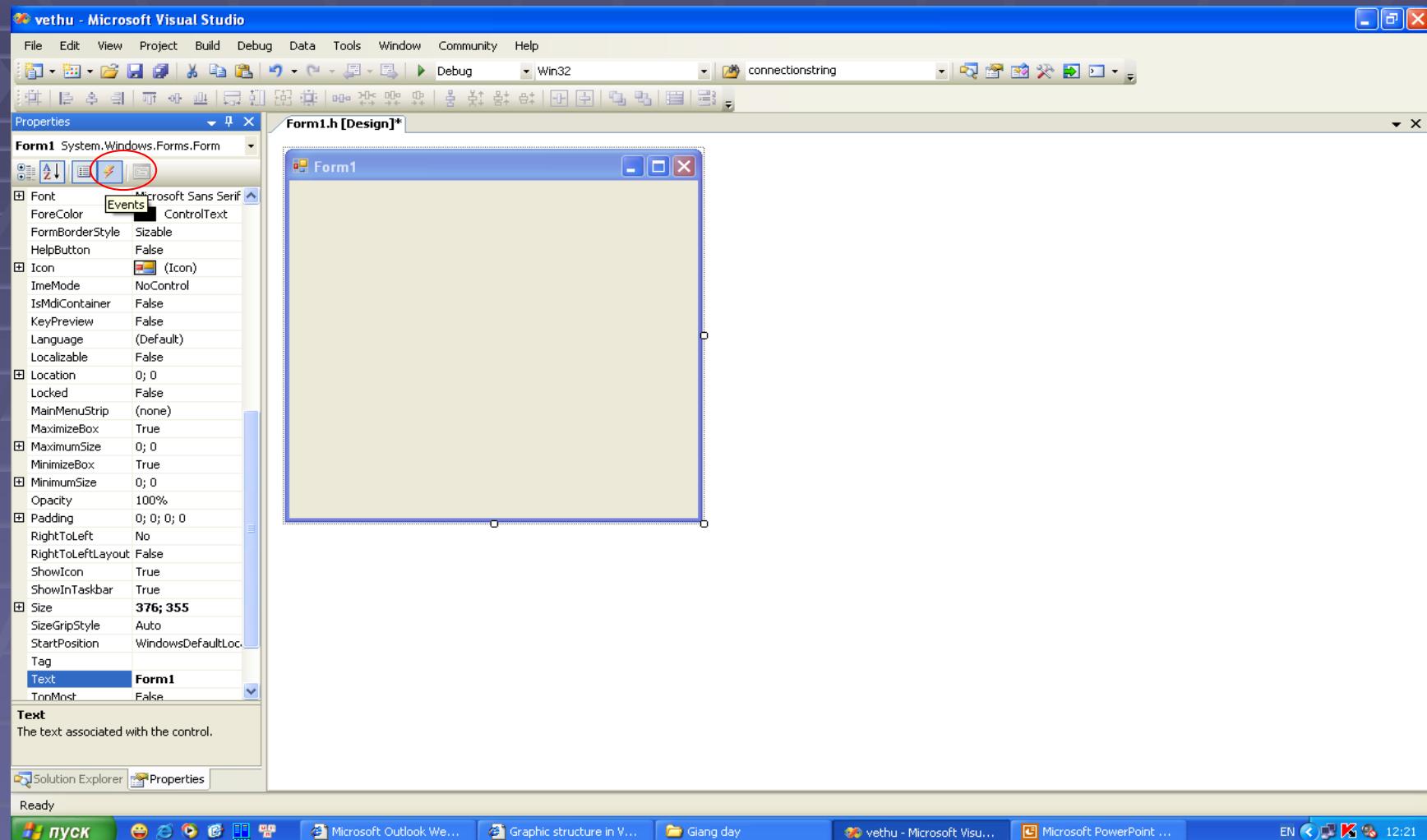
Bước 2 – Đặt tên cho Project



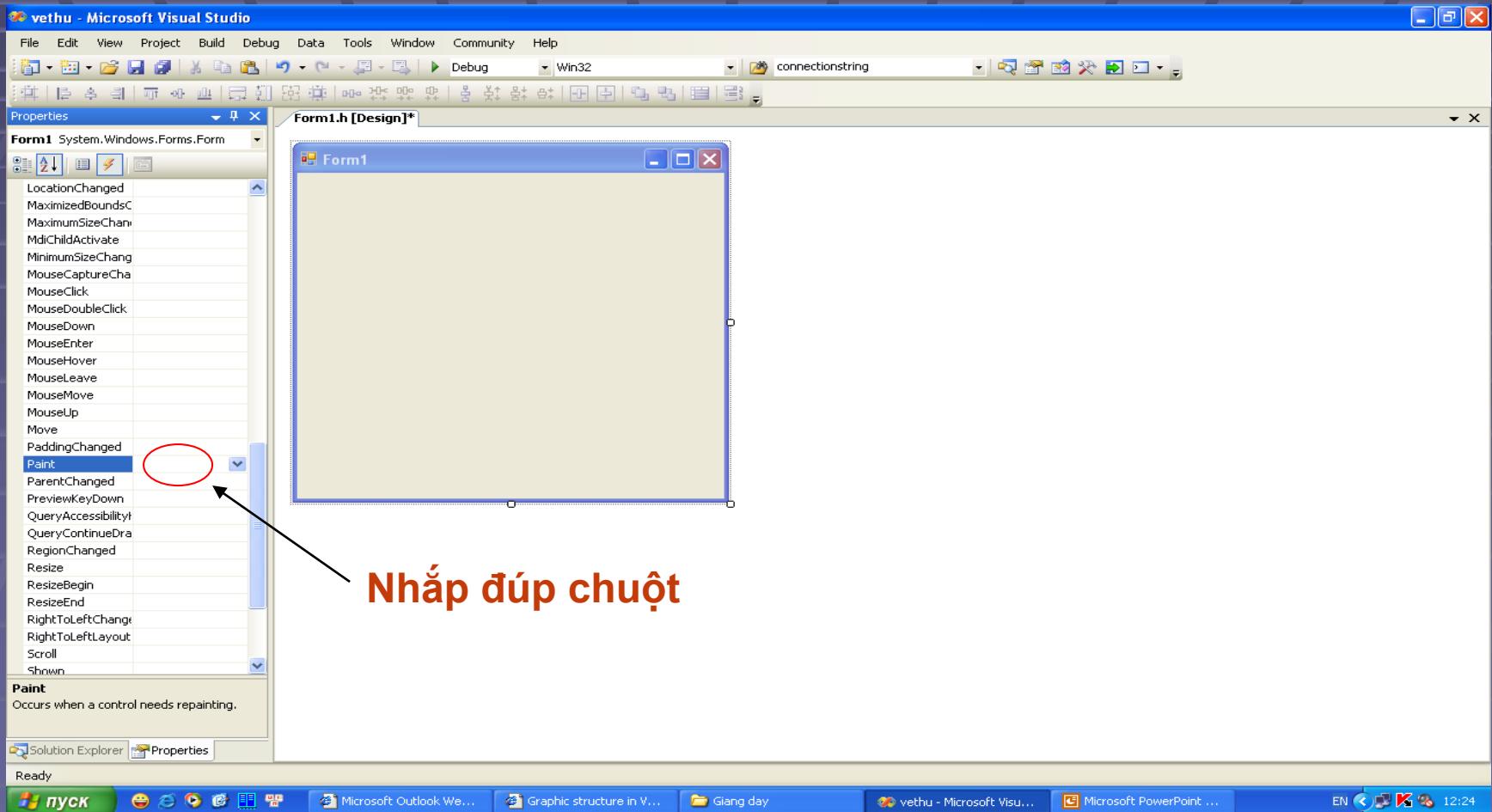
Bước 3 – Chọn thuộc tính của Form



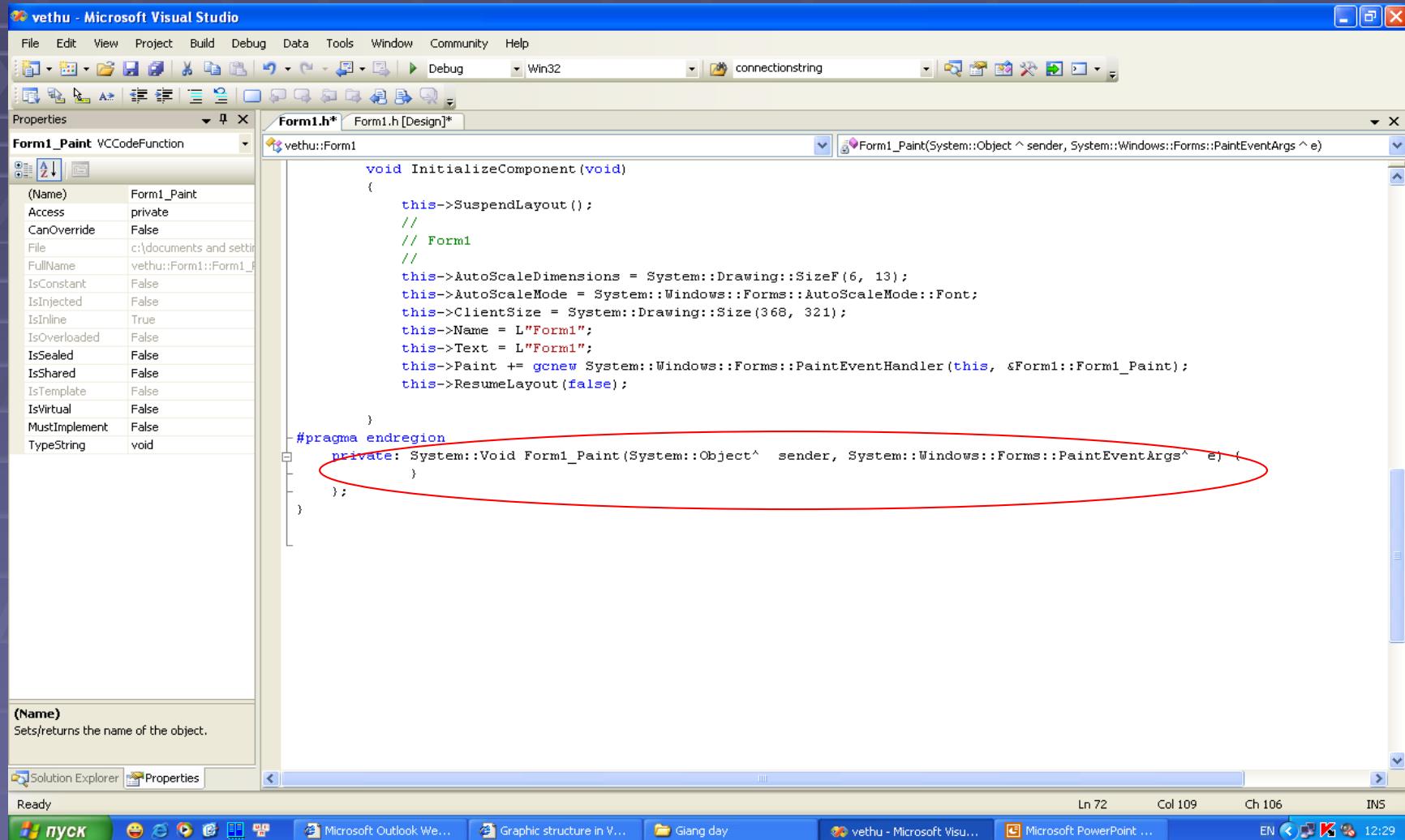
Bước 4 – Lựa chọn sự kiện của Form



Bước 5 – Chọn sự kiện vẽ



Bước 6 – Bổ sung các thao tác vẽ



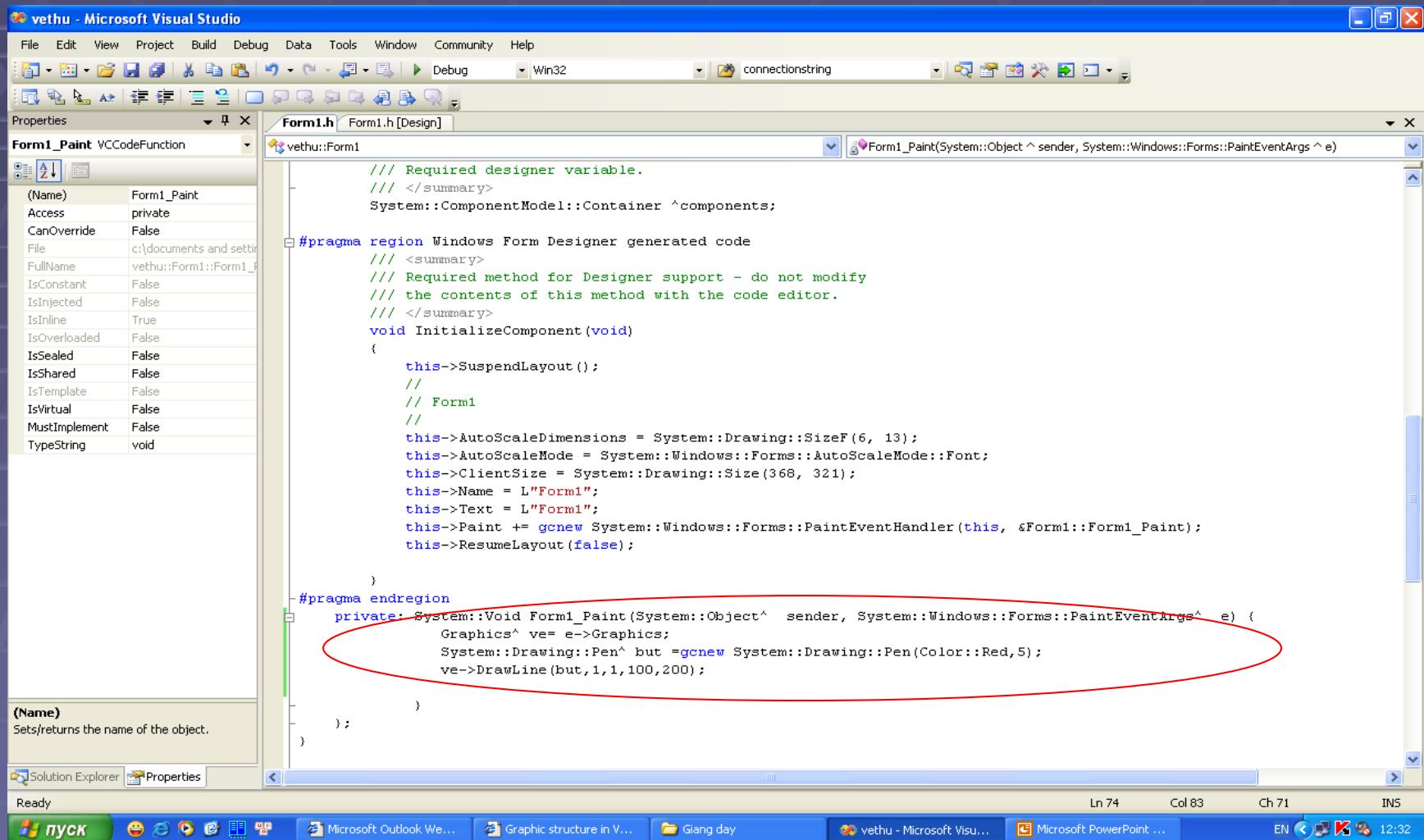
The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "vethu - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Data, Tools, Window, Community, and Help. The toolbar has various icons for file operations like Open, Save, Print, and Build. The solution explorer shows a project named "connectionstring". The properties window shows details for "Form1_Paint" which is a "VCCodeFunction". The code editor displays the following C++ code:

```
void InitializeComponent(void)
{
    this->SuspendLayout();
    // 
    // Form1
    //
    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
    this->AutoSizeMode = System::Windows::Forms::AutoSizeMode::Font;
    this->ClientSize = System::Drawing::Size(368, 321);
    this->Name = L"Form1";
    this->Text = L"Form1";
    this->Paint += gcnew System::Windows::Forms::PaintEventHandler(this, &Form1::Form1_Paint);
    this->ResumeLayout(false);
}

#pragma endregion
private: System::Void Form1_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) {
}
};
```

A large red oval highlights the entire private: block, specifically the line "private: System::Void Form1_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) {".

Bước 7



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "vethu - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Data, Tools, Window, Community, and Help. The toolbar has various icons for file operations. The solution explorer shows a project named "connectionstring". The properties window shows settings for "Form1_Paint" (VCCodeFunction). The code editor displays the header file "Form1.h" with the following content:

```
// Required designer variable.
// <summary>
System::ComponentModel::Container ^components;

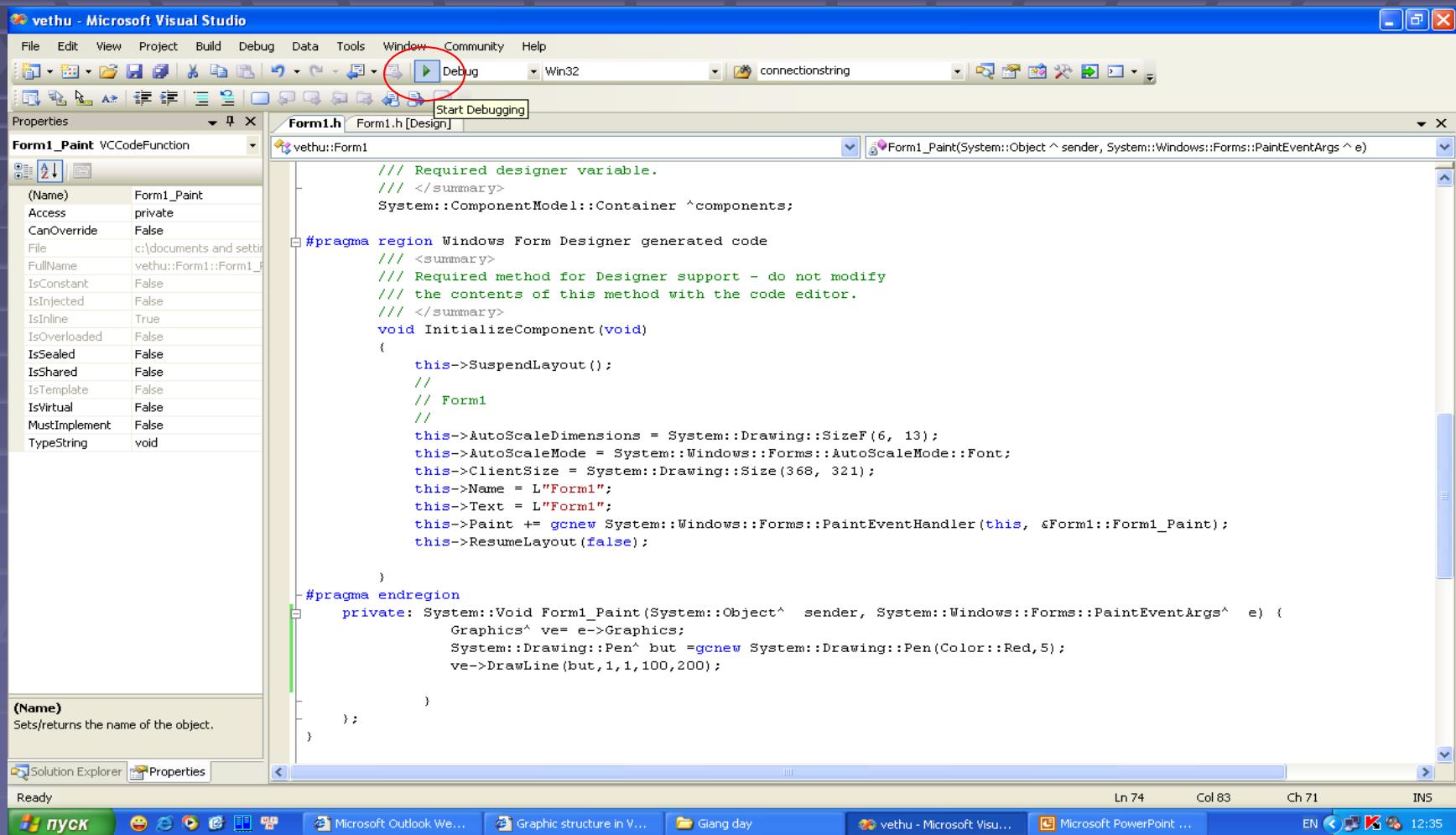
#pragma region Windows Form Designer generated code
// <summary>
// Required method for Designer support - do not modify
// the contents of this method with the code editor.
// </summary>
void InitializeComponent(void)
{
    this->SuspendLayout();
    //
    // Form1
    //
    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
    this->AutoSizeMode = System::Windows::Forms::AutoSizeMode::Font;
    this->ClientSize = System::Drawing::Size(368, 321);
    this->Name = L"Form1";
    this->Text = L"Form1";
    this->Paint += gcnew System::Windows::Forms::PaintEventHandler(this, &Form1::Form1_Paint);
    this->ResumeLayout(false);

}

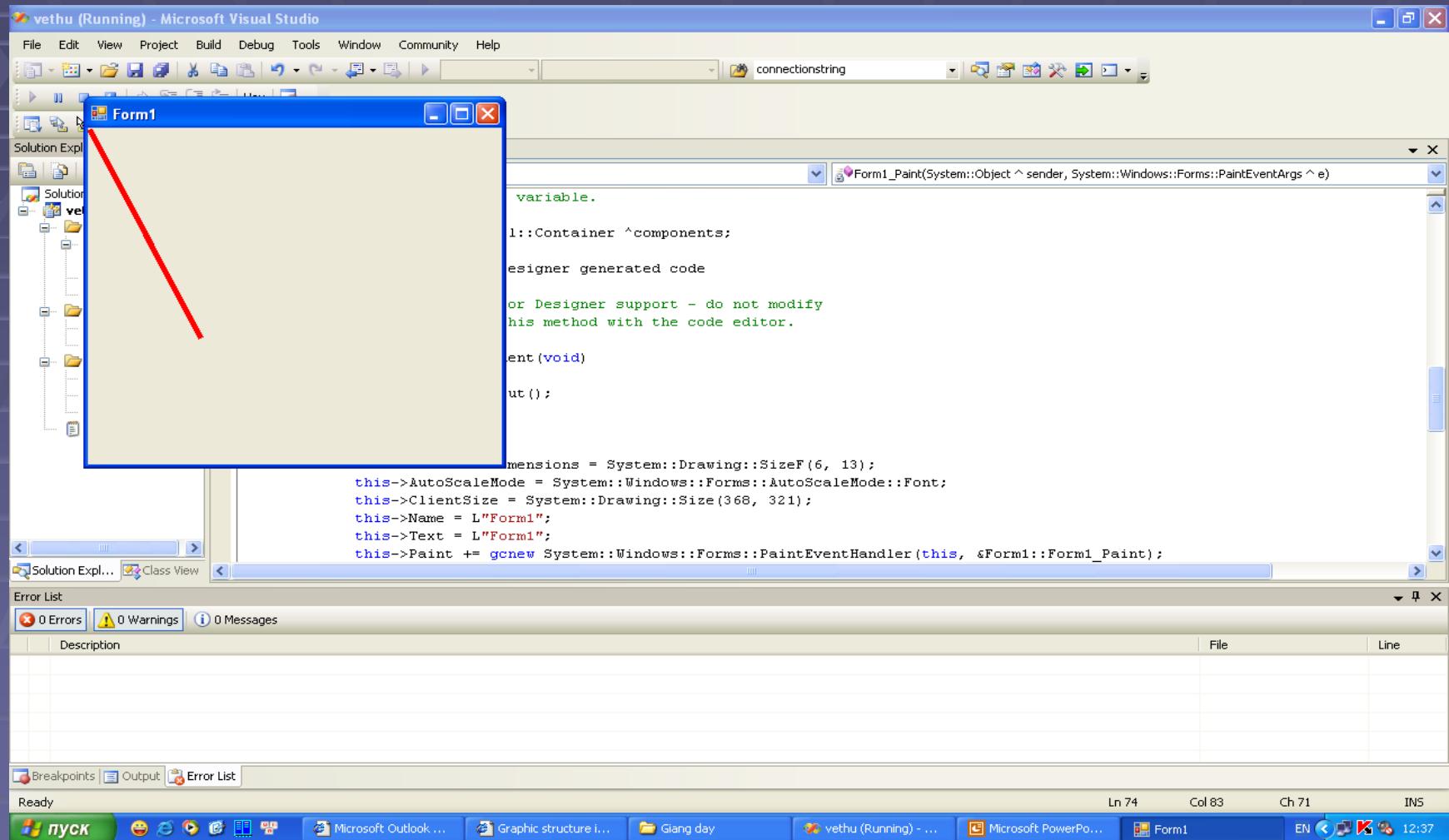
#pragma endregion
private: System::Void Form1_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) {
    Graphics^ ve= e->Graphics;
    System::Drawing::Pen^ but =gcnew System::Drawing::Pen(Color::Red,5);
    ve->DrawLine(but,1,1,100,200);
}
```

A red oval highlights the entire content of the "Form1_Paint" event handler.

Bước 8



Kết quả



Đối tượng Graphic của môi trường .Net

	Name	Description
	Clip	Gets or sets a Region that limits the drawing region of this Graphics .
	ClipBounds	Gets a RectangleF structure that bounds the clipping region of this Graphics .
	CompositingMode	Gets a value that specifies how composited images are drawn to this Graphics .
	CompositingQuality	Gets or sets the rendering quality of composited images drawn to this Graphics .
	DpiX	Gets the horizontal resolution of this Graphics .
	DpiY	Gets the vertical resolution of this Graphics .
	InterpolationMode	Gets or sets the interpolation mode associated with this Graphics .
	IsClipEmpty	Gets a value indicating whether the clipping region of this Graphics is empty.
	IsVisibleClipEmpty	Gets a value indicating whether the visible clipping region of this Graphics is empty.
	PageScale	Gets or sets the scaling between world units and page units for this Graphics .
	PageUnit	Gets or sets the unit of measure used for page coordinates in this Graphics .
	PixelOffsetMode	Gets or set a value specifying how pixels are offset during rendering of this Graphics .
	RenderingOrigin	Gets or sets the rendering origin of this Graphics for dithering and for hatch brushes.
	SmoothingMode	Gets or sets the rendering quality for this Graphics .
	TextContrast	Gets or sets the gamma correction value for rendering text.
	TextRenderingHint	Gets or sets the rendering mode for text associated with this Graphics .
	Transform	Gets or sets the world transformation for this Graphics .
	VisibleClipBounds	Gets the bounding rectangle of the visible clipping region of this Graphics .

	Name	Description
	AddMetafileComment	Adds a comment to the current Metafile .
	BeginContainer	Overloaded. Saves a graphics container with the current state of this Graphics and opens and uses a new graphics container.
	Clear	Clears the entire drawing surface and fills it with the specified background color.
	CopyFromScreen	Overloaded. Performs a bit-block transfer of color data from the screen to the drawing surface of the Graphics .
	CreateObjRef	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object. (inherited from MarshalByRefObject)
	Dispose	Releases all resources used by this Graphics .
	DrawArc	Overloaded. Draws an arc representing a portion of an ellipse specified by a pair of coordinates, a width, and a height.
	DrawBezier	Overloaded. Draws a Bézier spline defined by four Point structures.
	DrawBeziers	Overloaded. Draws a series of Bézier splines from an array of Point structures.
	DrawClosedCurve	Overloaded. Draws a closed cardinal spline defined by an array of Point structures.
	DrawCurve	Overloaded. Draws a cardinal spline through a specified array of Point structures.
	DrawEllipse	Overloaded. Draws an ellipse defined by a bounding rectangle specified by a pair of coordinates, a height, and a width.
	DrawIcon	Overloaded. Draws the image represented by the specified Icon at the specified coordinates.
	DrawIconUnstretched	Draws the image represented by the specified Icon without scaling the image.
	DrawImage	Overloaded. Draws the specified Image at the specified location and with the original size.
	DrawImageUnscaled	Overloaded. Draws the specified image using its original physical size at the location specified by a coordinate pair.

DrawImageUnscaledAndClipped	Draws the specified image without scaling and clips it, if necessary, to fit in the specified rectangle.
DrawLine	Overloaded. Draws a line connecting the two points specified by the coordinate pairs.
DrawLines	Overloaded. Draws a series of line segments that connect an array of Point structures.
DrawPath	Draws a GraphicsPath .
DrawPie	Overloaded. Draws a pie shape defined by an ellipse specified by a coordinate pair, a width, a height, and two radial lines.
DrawPolygon	Overloaded. Draws a polygon defined by an array of Point structures.
DrawRectangle	Overloaded. Draws a rectangle specified by a coordinate pair, a width, and a height.
DrawRectangles	Overloaded. Draws a series of rectangles specified by Rectangle structures.
DrawString	Overloaded. Draws the specified text string at the specified location with the specified Brush and Font objects.
EndContainer	Closes the current graphics container and restores the state of this Graphics to the state saved by a call to the BeginContainer method.
EnumerateMetafile	Overloaded. Sends the records in the specified Metafile , one at a time, to a callback method for display at a specified point.
Equals	Overloaded. Determines whether two Object instances are equal. (inherited from Object)
ExcludeClip	Overloaded. Updates the clip region of this Graphics to exclude the area specified by a Rectangle structure.
FillClosedCurve	Overloaded. Fills the interior of a closed cardinal spline curve defined by an array of Point structures.
FillEllipse	Overloaded. Fills the interior of an ellipse defined by a bounding rectangle specified by a pair of coordinates, a width, and a height.
FillPath	Fills the interior of a GraphicsPath .
FillPie	Overloaded. Fills the interior of a pie section defined by an ellipse specified by a pair of coordinates, a width, a height, and two radial lines.

Nhận xét

- Lớp Graphic đã chứa rất nhiều các thao tác (hàm, thủ tục dạng thư viện) cơ bản phục vụ cho việc vẽ đồ họa
- Tuy nhiên bản chất các thư viện đó là gì?
Cơ chế hoạt động như thế nào?

Thực hành vẽ đồ họa trong môi trường

C# Visual Studio 2008

How to: Create Graphics Objects for Drawing

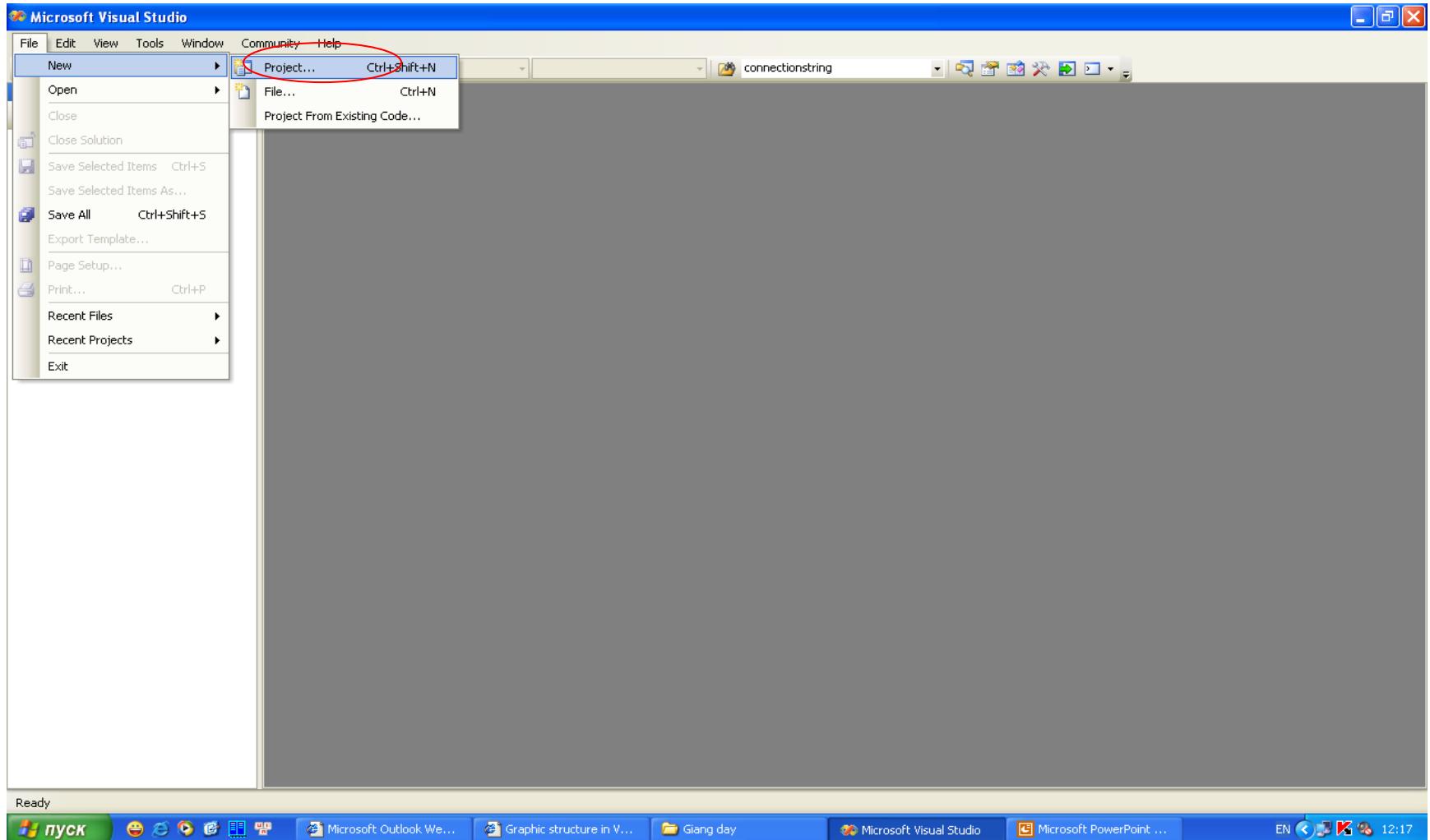
- Để tạo môi trường vẽ đồ họa trong .Net trước hết cần có đối tượng **Graphics** object. Đối tượng này có thể gắn vào các đối tượng điều khiển khác. Có 2 cách để làm việc với đối tượng đồ họa:
 - Tạo mới Graphics object.
 - Sử dụng Graphics object sẵn có để vẽ.

Tạo graphics object

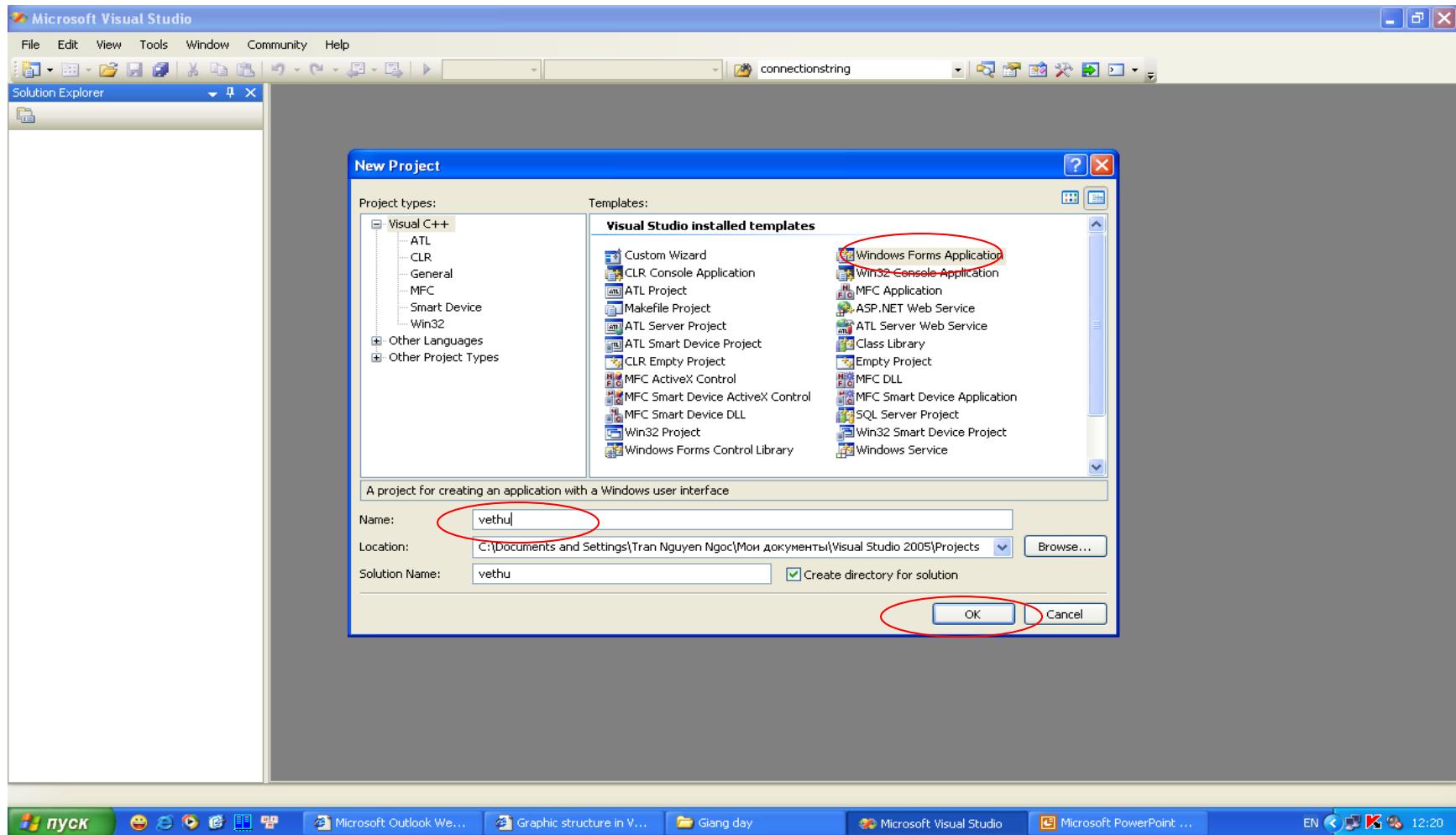
- Thừa kế lớp [PaintEventArgs](#) từ sự kiện [Paint](#) của Form hoặc lớp Control.
- Gọi phương thức [CreateGraphics](#) của Control or Form để vẽ lên các đối tượng đó.
- Tạo Graphics object từ bất kĩ đối tượng nào thừa kế từ lớp [Image](#).

```
private void Form1_Paint(object sender, PaintEventArgs  
e)  
{  
    •     Graphics ve = e.Graphics;  
    •     Pen but = new Pen(Color.Red,5);  
    •     ve.DrawLine(but, 1, 1, 100, 200);  
  
    •     }  
}
```

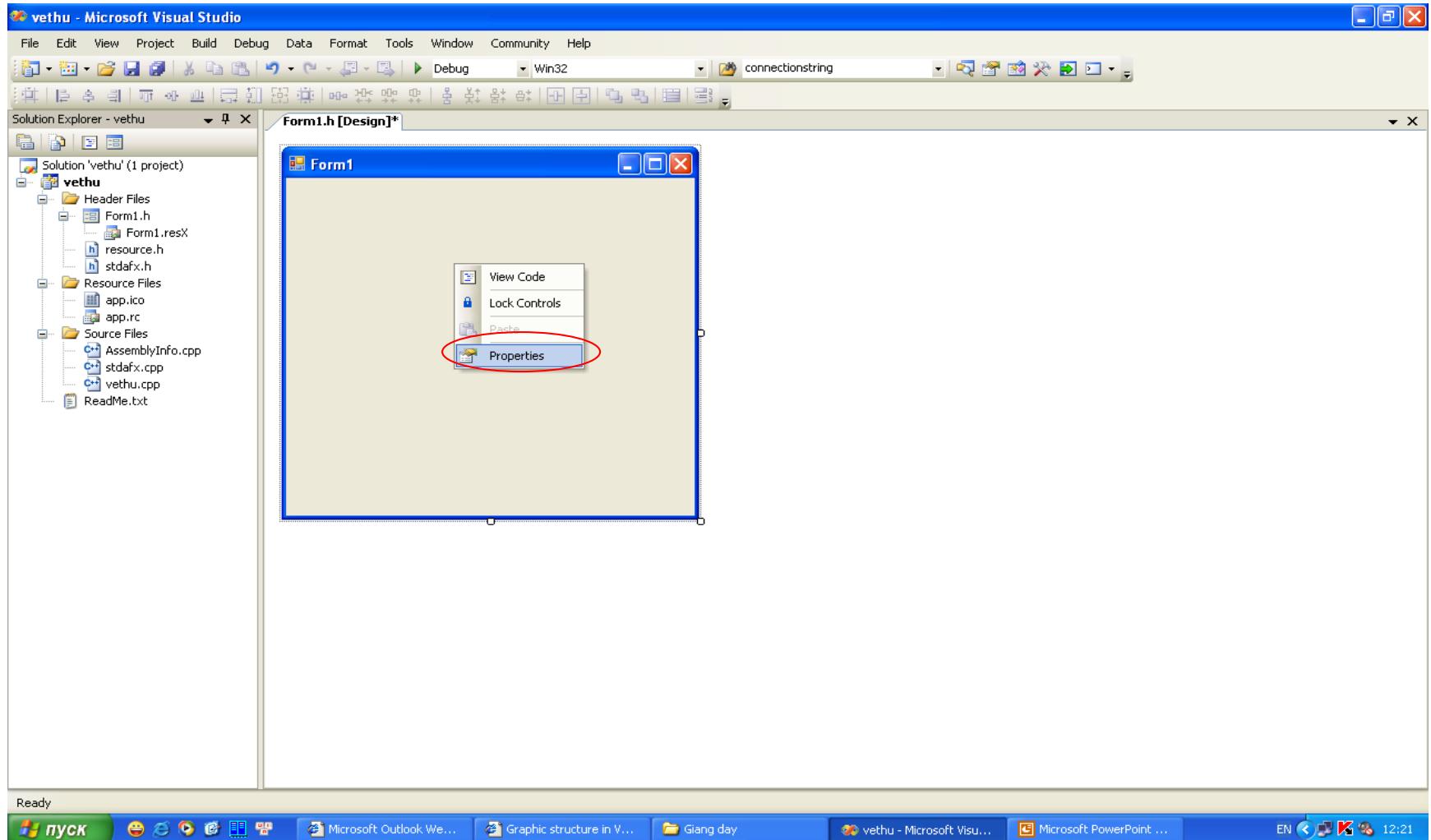
Bước 1-Chọn new Project



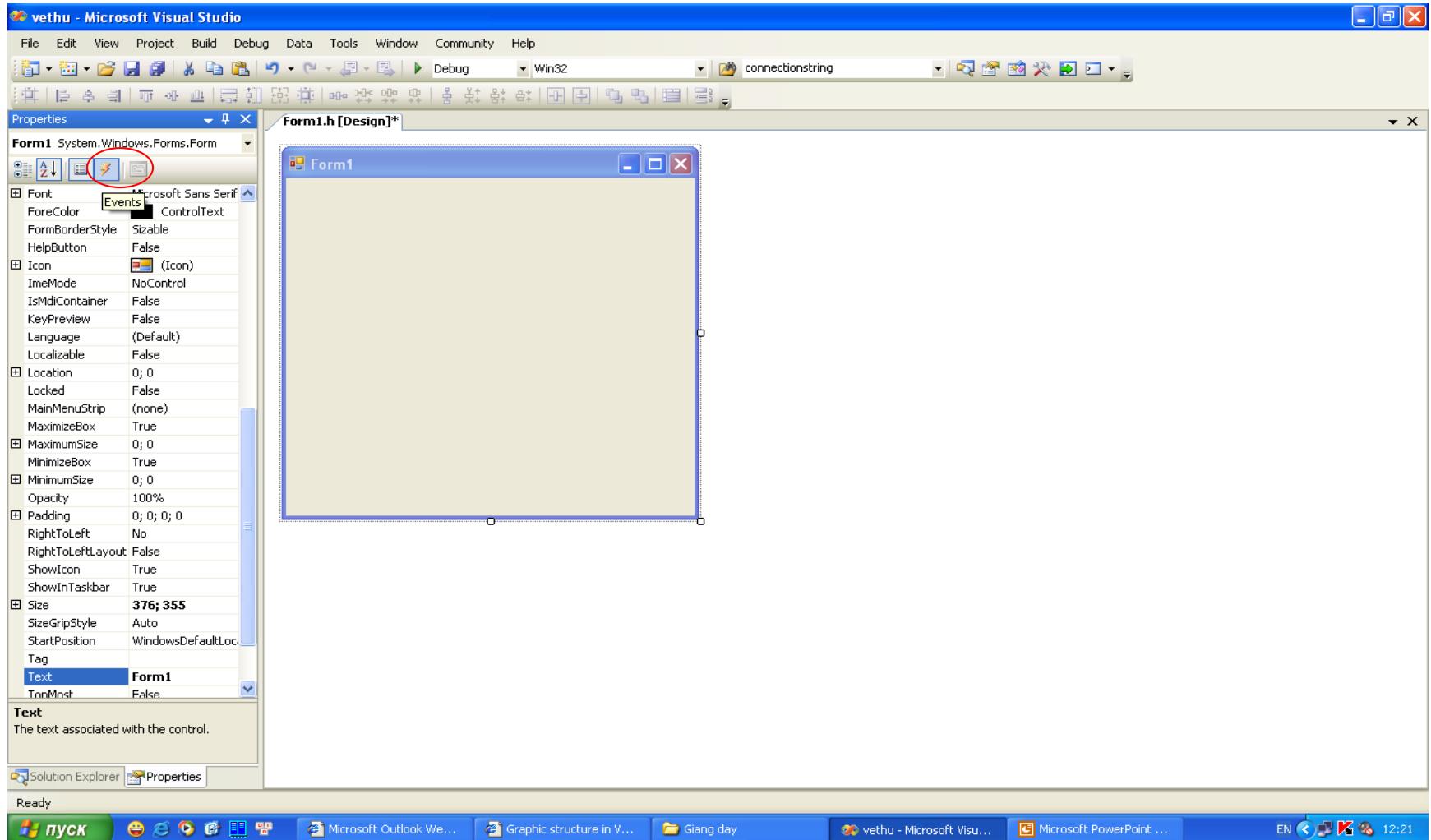
Bước 2 – Đặt tên cho Project



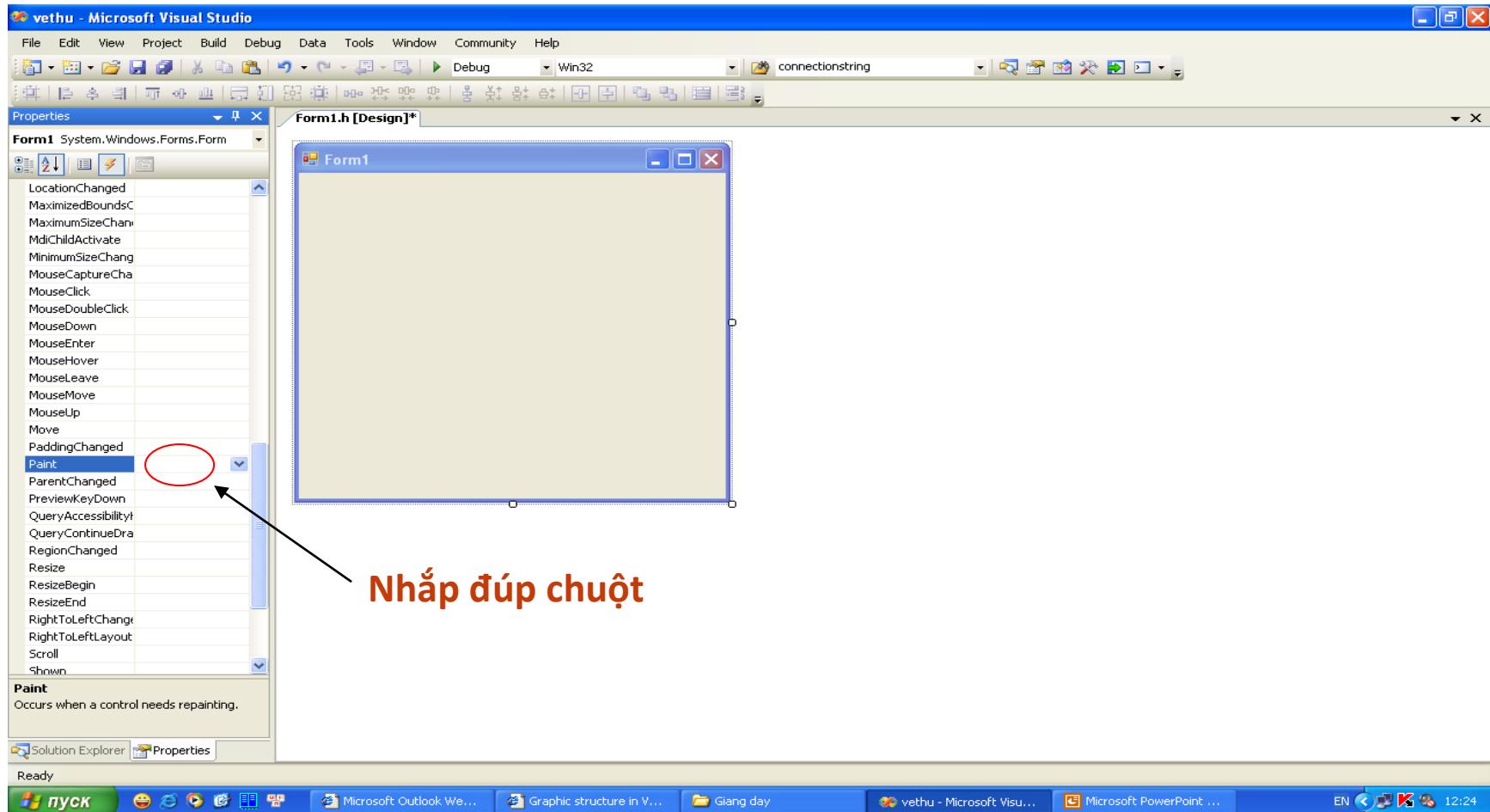
Bước 3 – Chọn thuộc tính của Form



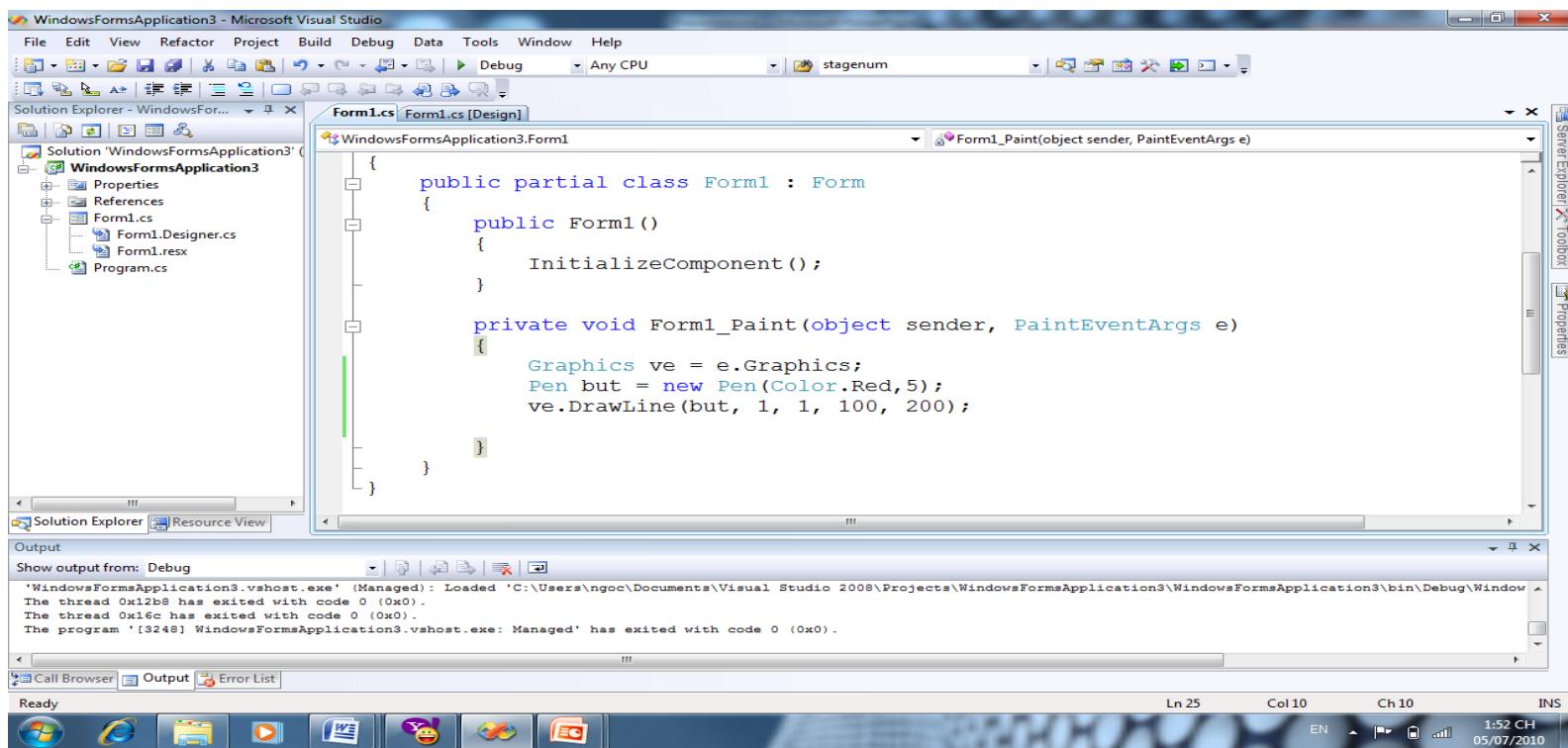
Bước 4 – Lựa chọn sự kiện của Form



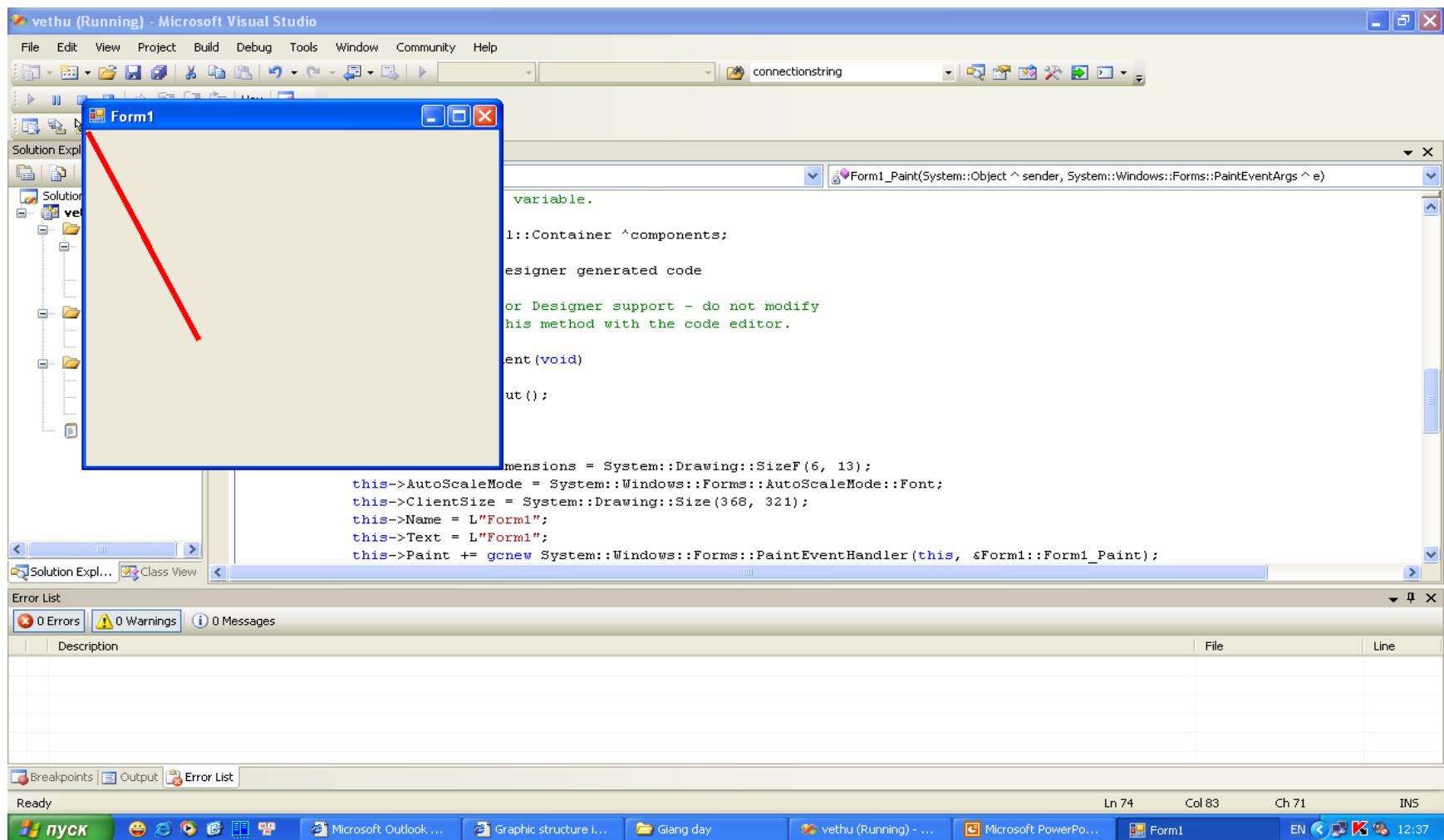
Bước 5 – Chọn sự kiện vẽ



Bước 6



Kết quả



Đối tượng Graphic của môi trường .Net

	Name	Description
	Clip	Gets or sets a Region that limits the drawing region of this Graphics .
	ClipBounds	Gets a RectangleF structure that bounds the clipping region of this Graphics .
	CompositingMode	Gets a value that specifies how composited images are drawn to this Graphics .
	CompositingQuality	Gets or sets the rendering quality of composited images drawn to this Graphics .
	DpiX	Gets the horizontal resolution of this Graphics .
	DpiY	Gets the vertical resolution of this Graphics .
	InterpolationMode	Gets or sets the interpolation mode associated with this Graphics .
	IsClipEmpty	Gets a value indicating whether the clipping region of this Graphics is empty.
	IsVisibleClipEmpty	Gets a value indicating whether the visible clipping region of this Graphics is empty.
	PageScale	Gets or sets the scaling between world units and page units for this Graphics .
	PageUnit	Gets or sets the unit of measure used for page coordinates in this Graphics .
	PixelOffsetMode	Gets or set a value specifying how pixels are offset during rendering of this Graphics .
	RenderingOrigin	Gets or sets the rendering origin of this Graphics for dithering and for hatch brushes.
	SmoothingMode	Gets or sets the rendering quality for this Graphics .
	TextContrast	Gets or sets the gamma correction value for rendering text.
	TextRenderingHint	Gets or sets the rendering mode for text associated with this Graphics .
	Transform	Gets or sets the world transformation for this Graphics .
	VisibleClipBounds	Gets the bounding rectangle of the visible clipping region of this Graphics .

	Name	Description
	AddMetafileComment	Adds a comment to the current Metafile .
	BeginContainer	Overloaded. Saves a graphics container with the current state of this Graphics and opens and uses a new graphics container.
	Clear	Clears the entire drawing surface and fills it with the specified background color.
	CopyFromScreen	Overloaded. Performs a bit-block transfer of color data from the screen to the drawing surface of the Graphics .
	CreateObjRef	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object. (inherited from MarshalByRefObject)
	Dispose	Releases all resources used by this Graphics .
	DrawArc	Overloaded. Draws an arc representing a portion of an ellipse specified by a pair of coordinates, a width, and a height.
	DrawBezier	Overloaded. Draws a Bézier spline defined by four Point structures.
	DrawBeziers	Overloaded. Draws a series of Bézier splines from an array of Point structures.
	DrawClosedCurve	Overloaded. Draws a closed cardinal spline defined by an array of Point structures.
	DrawCurve	Overloaded. Draws a cardinal spline through a specified array of Point structures.
	DrawEllipse	Overloaded. Draws an ellipse defined by a bounding rectangle specified by a pair of coordinates, a height, and a width.
	DrawIcon	Overloaded. Draws the image represented by the specified Icon at the specified coordinates.
	DrawIconUnstretched	Draws the image represented by the specified Icon without scaling the image.
	DrawImage	Overloaded. Draws the specified Image at the specified location and with the original size.
	DrawImageUnscaled	Overloaded. Draws the specified image using its original physical size at the location specified by a coordinate pair.

DrawImageUnscaledAndClipped	Draws the specified image without scaling and clips it, if necessary, to fit in the specified rectangle.
DrawLine	Overloaded. Draws a line connecting the two points specified by the coordinate pairs.
DrawLines	Overloaded. Draws a series of line segments that connect an array of Point structures.
DrawPath	Draws a GraphicsPath .
DrawPie	Overloaded. Draws a pie shape defined by an ellipse specified by a coordinate pair, a width, a height, and two radial lines.
DrawPolygon	Overloaded. Draws a polygon defined by an array of Point structures.
DrawRectangle	Overloaded. Draws a rectangle specified by a coordinate pair, a width, and a height.
DrawRectangles	Overloaded. Draws a series of rectangles specified by Rectangle structures.
DrawString	Overloaded. Draws the specified text string at the specified location with the specified Brush and Font objects.
EndContainer	Closes the current graphics container and restores the state of this Graphics to the state saved by a call to the BeginContainer method.
EnumerateMetafile	Overloaded. Sends the records in the specified Metafile , one at a time, to a callback method for display at a specified point.
Equals	Overloaded. Determines whether two Object instances are equal. (inherited from Object)
ExcludeClip	Overloaded. Updates the clip region of this Graphics to exclude the area specified by a Rectangle structure.
FillClosedCurve	Overloaded. Fills the interior of a closed cardinal spline curve defined by an array of Point structures.
FillEllipse	Overloaded. Fills the interior of an ellipse defined by a bounding rectangle specified by a pair of coordinates, a width, and a height.
FillPath	Fills the interior of a GraphicsPath .
FillPie	Overloaded. Fills the interior of a pie section defined by an ellipse specified by a pair of coordinates,

Nội dung sinh viên tự thực hành

- Tìm hiểu các hàm khác của lớp Graphics:
Drawlines; DrawPath; DrawRectangle;
DrawString; FillRectangle; FillPolygon(Brush,
Point[]); RotateTransform(Single)

Bài tập lập trình vận dụng

- Viết chương trình vẽ tam giác có cạnh màu đỏ ABC theo các tọa độ A(10,10); B(10,200);C(200,200), sau đó từ điểm P(50,100) bên trong tam giác ABC bằng thuật toán tô màu theo đường biên hãy tô màu đa giác đó sao cho tất cả các điểm bên trong tam giác có màu xanh.

Ôn tập

Phương pháp truy cập các điểm ảnh bằng cơ chế Lockbits.

- Bitmap bm = new Bitmap(openFileDialog1.FileName);
- Rectangle rec = new Rectangle(0, 0, bm.Width, bm.Height);
- BitmapData bmData = bm.LockBits(rec,
ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);
- int stride = bmData.Stride;
- int nOffset = stride - bm.Width * 3;

Tiếp theo

- unsafe
- {
- byte* p = (byte*)bmData.Scan0;
- for (int y = 0; y < bm.Height; y++)
- {
- for (int x = 0; x < bm.Width; x++)
- {
- //các thao tác trên con trỏ p
- }
- }
-
- bm.UnlockBits(bmData);
- }