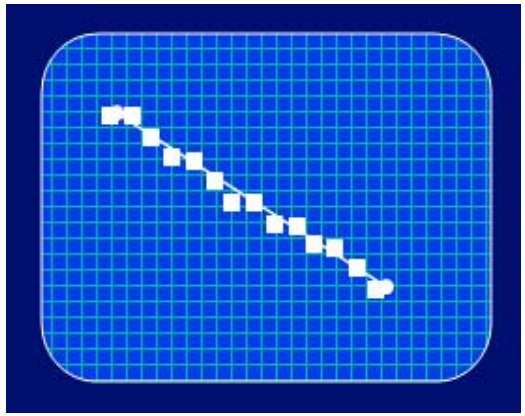
Review Lecture 1B: Graphics System

Raster vs. Vector Graphics

Raster

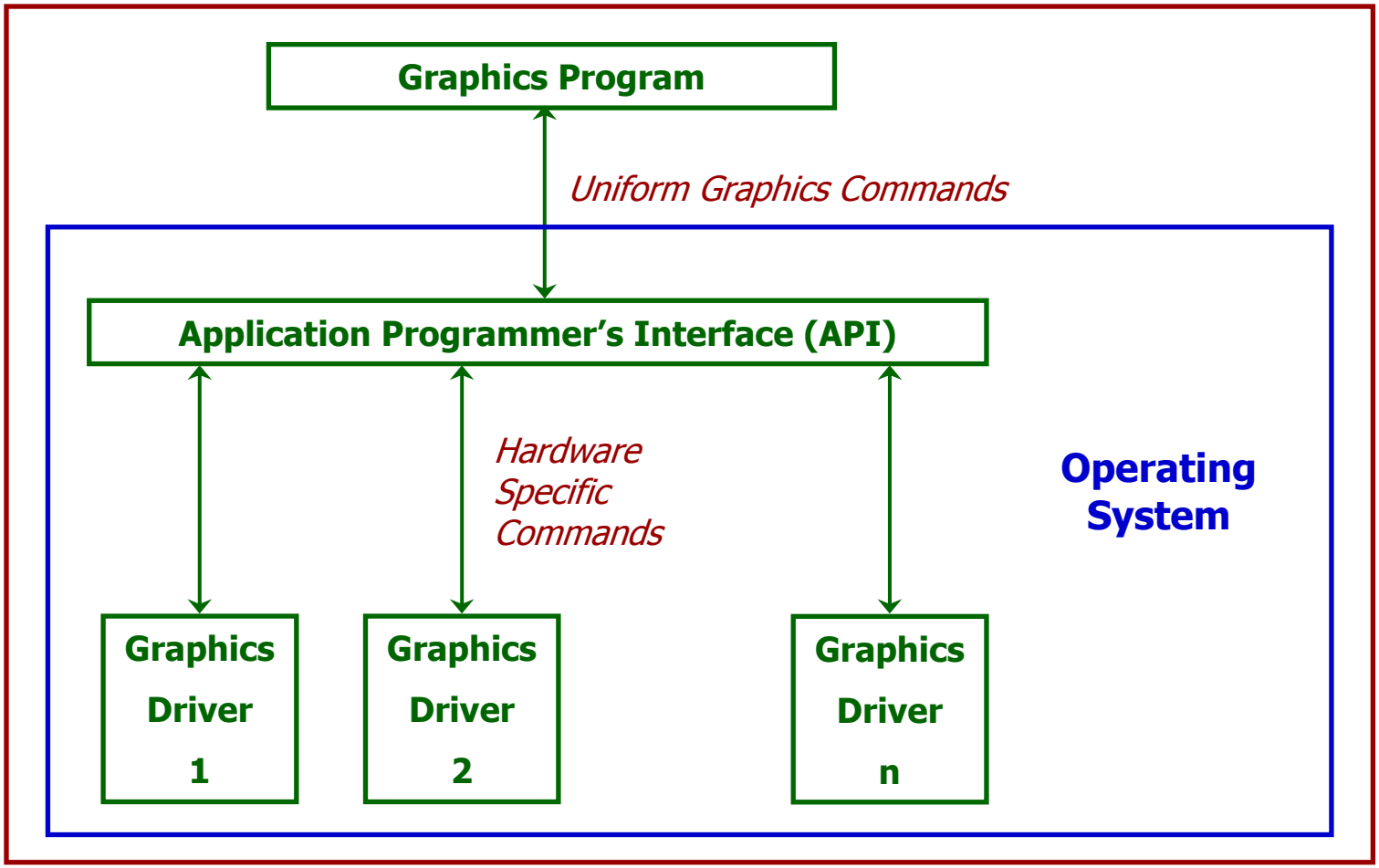


ค่าสีของแต่ละจุด หรือ **Pic**ture **El**ement (Pixel) ซึ่งเก็บอยู่ใน Flame Buffer ประกอบกันขึ้นเป็นภาพบนอุปกรณ์แสดงผล

Vector

ลำดับของคำสั่งที่เก็บอยู่ใน Display File ถูกประมวลผลตามหลักสมการทางคณิตศาสตร์โดย Display Processor แล้วแสดงผลที่จอภาพเฉพาะบริเวณที่รูปภาพแสดง

Application Programmer's Interface



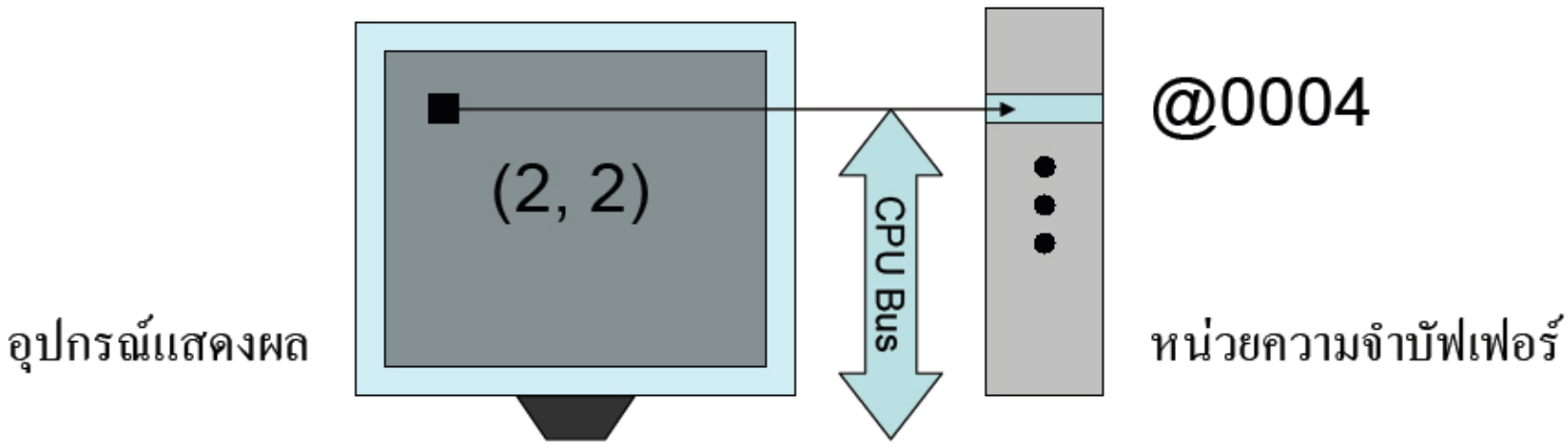
โปรแกรมกราฟิก ในทางปฏิบัติควรจะ เป็นอิสระ ต่ออุปกรณ์แสดงผลให้มากที่สุด (Device Independent) โดยปกติ OS จะมี API ให้สำหรับการทำงานพื้นฐาน

Raster Devices

Raster Devices

คืออุปกรณ์แสดงผล ที่พบได้โดยทั่วไป ได้แก่ จอภาพ และ เครื่องพิมพ์

ลักษณะจำเพาะคือ แต่ละจุด หรือ **Pic**ture **El**ement (Pixel) ซึ่งประกอบกันขึ้นเป็นภาพบนอุปกรณ์แสดงผล จะอ้างถึง (Map) หน่วยความจำ แบบ Random Access (RAM) ซึ่งสามารถเข้าถึงได้ โดย CPU หรือ Register ควบคุม

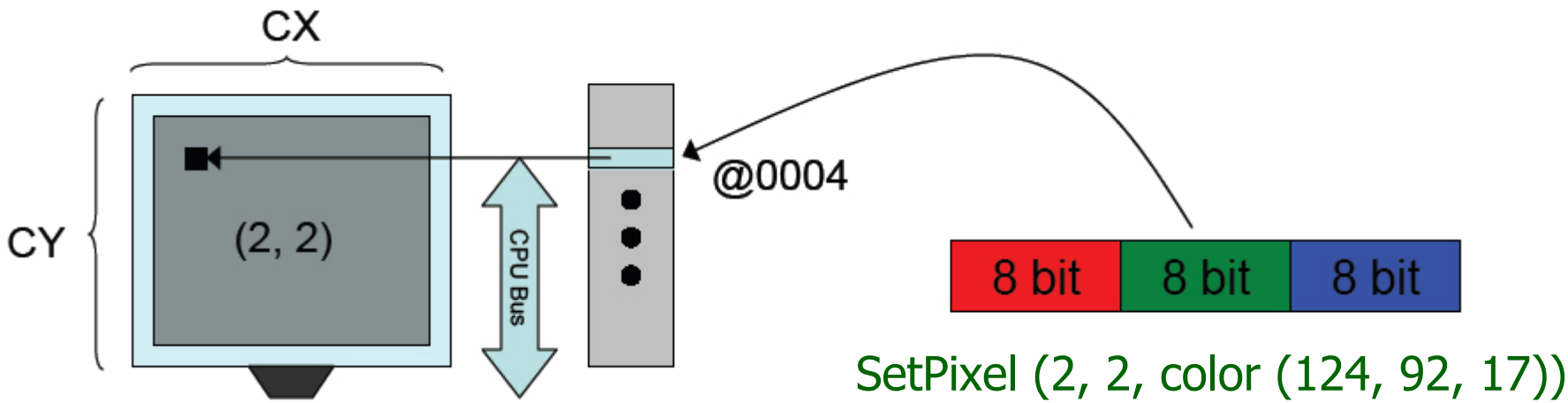


Raster Terminology

Pixel depth คือขนาดของหน่วยความจำแต่ละตำแหน่ง (บิต) ซึ่งกำหนดพิสัยของความเข้มของจุดภาพนั้น เช่น 1 บิต หมายถึงจุดภาพนั้น ดับ หรือ สว่าง 8 บิต หมายถึง จุดภาพมีความเข้มแตกต่างกัน 256 ระดับ

Color คือจำนวนสีพื้นฐาน ที่ประกอบขึ้นเป็นจุดภาพนั้นๆ เช่น จอภาพ CRT มี 3 สี ได้แก่ แดง (R) เขียว (G) น้าเงิน (B) แต่ละสีมี Pixel depth 8 บิต จะแสดงจุดภาพที่มีสีที่สว่างแตกต่างกันได้ 16 ล้านสี หรือ Pixel depth รวม 24 บิต

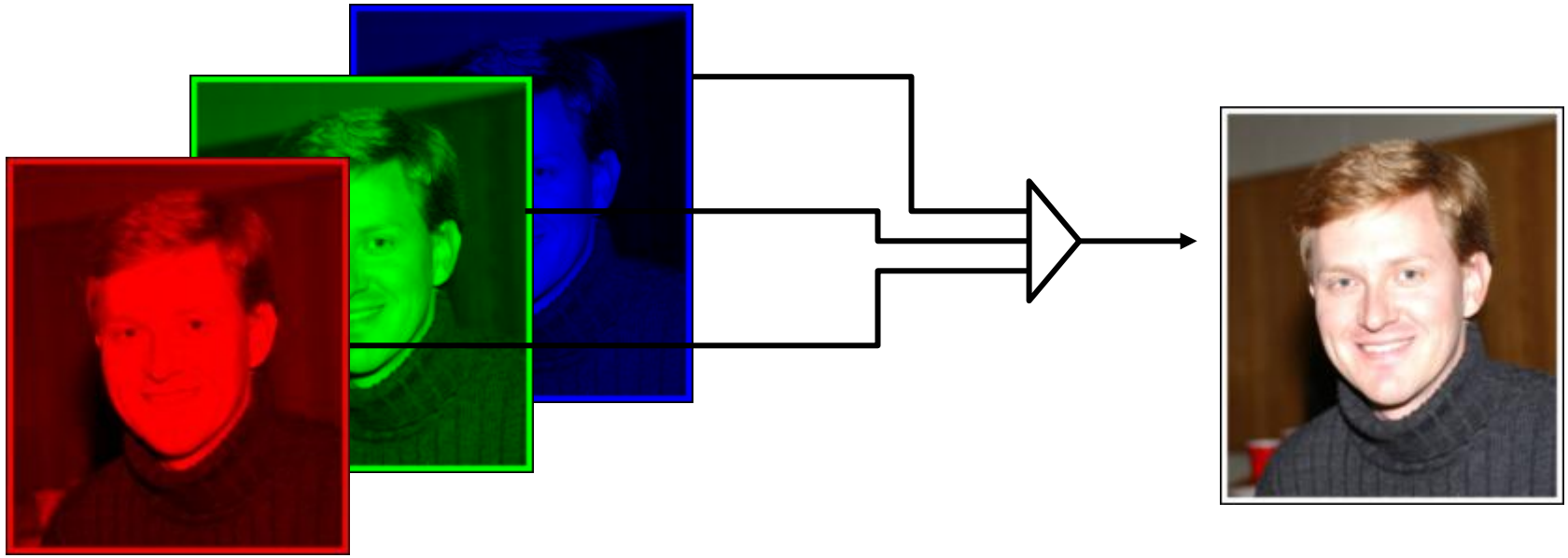
Resolution คือ จำนวนจุดภาพทั้งหมด ที่ปรากฏบนอุปกรณ์แสดงผล



SetPixel (2, 2, color (124, 92, 17))

Raster Terminology

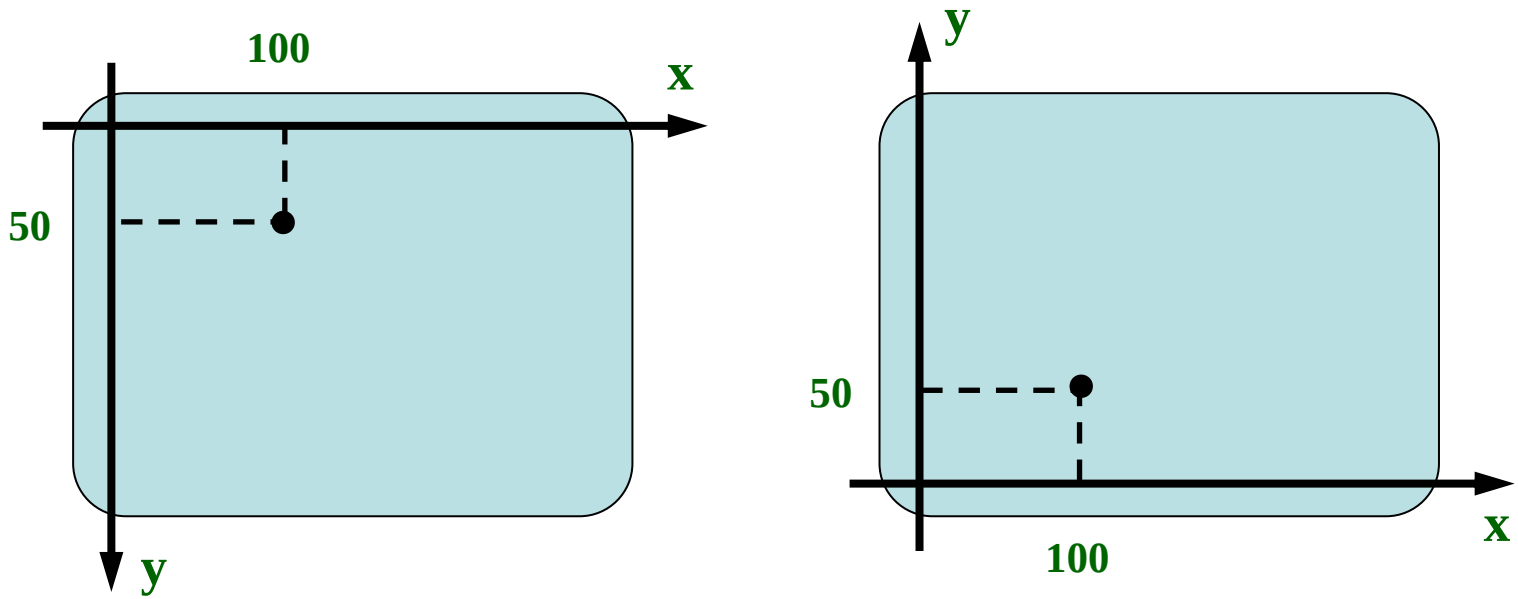
24 bits per pixel =
8 bits red, 8 bits green, 8 bits blue



Lecture 2A:
Overview of Windows Programming and Raster Algorithm (Part I)

- Coordinate System
- Input Device, Event Mouse
- Windows Programming
- Raster Algorithm: Line Drawing Algorithm

Addressing Conventions



Pixel addressing conventions vary from system to system

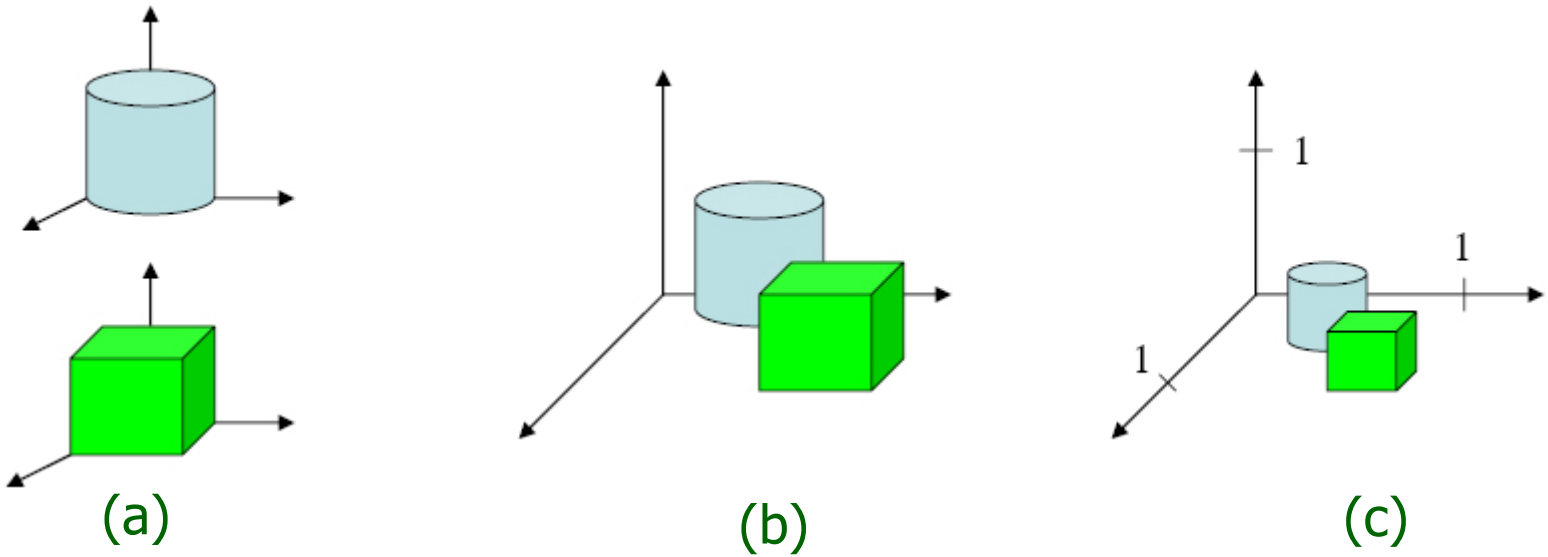
การเขียนโปรแกรมจำเป็นจะต้องคำนึงถึงการเปลี่ยนแปลง Resolution ด้วย

Coordinate Systems

(a) Object or Model or Local Coordinates
คือระบบปริภูมิที่กำหนดสำหรับการวาดวัตถุแต่ละชิ้น

(b) World or Physical Coordinates
คือระบบปริภูมิที่ เลื่อนวัตถุใดๆ ไปอยู่ในตำแหน่งที่เหมาะสมในฉาก

(c) Normalized or Device or Screen Coordinates
คือระบบปริภูมิที่ใช้สำหรับวาดวัตถุที่ปรากฏ บนอุปกรณ์แสดงผล



(a) (b) (c)

World Coordinate System

การเขียนโปรแกรม แบบ Device Independent จำเป็นต้องมีการนิยามระบบปริภูมิกายภาพ (World Coordinate System) ซึ่งจะเป็นการอ้างอิงการแสดงผลกับหน่วยวัดทางกายภาพ ให้เหมาะสมกับการประยุกต์ใช้งานได้แก่ mm, cm หรือ yard

SetWindowWorldCoords(WXMin,WYMin,WXMax,WYMax);

Drawing Graphics Primitives

หลังจากนิยามปริภูมิทางกายภาพแล้ว เราสามารถวาด Graphics Primitive บนบริเวณที่กำหนดได้ ตัวอย่างเช่น

SetPixel (x, y, color);
DrawLine (x1, y1, x2, y2);
DrawCircle (x, y, radius);
DrawPolygon (List of Points (x, y));
DrawText (x1, y1, "Suranaree University of Technology");

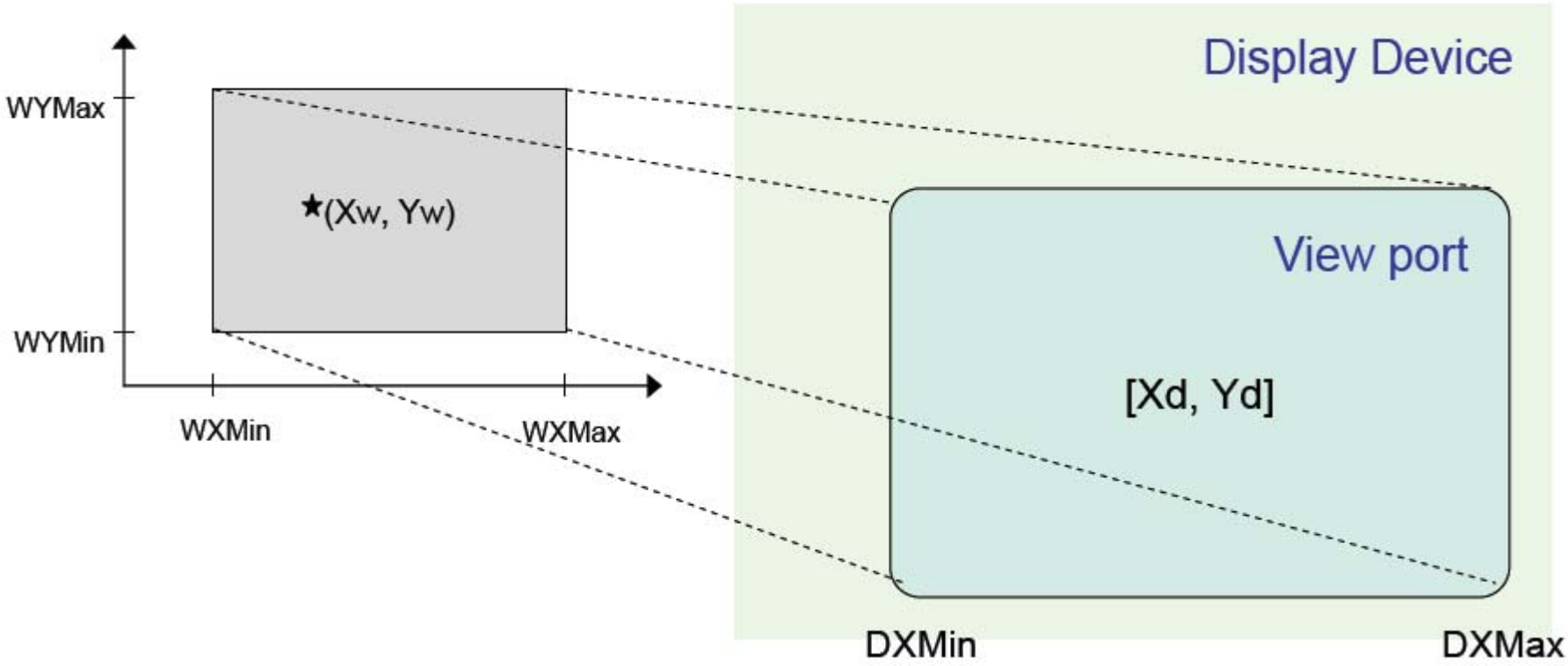
นอกจากนี้ เรายังสามารถกำหนดรูปแบบ ของการแสดงผลได้โดย การกำหนดค่าให้กับ Attributes ที่เกี่ยวข้อง เช่น

Line ได้แก่ Style (Solid/Dash หรือ Dot), Thickness (in หรือ Pixels)
ColorText ได้แก่ Font, Size, Color

Normalization

คือขั้นตอนการแปลง World Coordinate ไปเป็น Normalized หรือ Device Coordinates เพื่อที่จะนำไปแสดงที่อุปกรณ์แสดงผลต่อไป

ขั้นตอนนี้สามารถ คำนวณได้โดยใช้หลักสมการเชิงเส้นพื้นฐาน



Normalization

การกำหนดค่าของ View port สามารถทำได้ในทำนองเดียวกัน

SetDisplayCoords (DXMin,DYMin,DXMax,DYMax);

ความสัมพันธ์ระหว่าง World และ Device coordinates เขียนได้เป็น

$$(Xw-WXmin)/(WXMax-WXMin) = (Xd-DXMin)/(DXMax-DXMin)$$

จัดข้างสมการแล้วจะได้ว่า $Xd = Xw*A + B$

โดยที่ $A = (DXmax-DXmin)/(WXmax-WXmin)$

$B = -WXmin(DXmax-DXmin)/(WXmax-WXmin)+DXmin$

Input Device

สำหรับ Computer Graphics มีอุปกรณ์รับข้อมูลหลายชนิดได้แก่

Mouse, Joystick, Keypad, Digitization Tablet, Light Pen, etc

ในที่นี้เราจะพิจารณาเฉพาะ Mouse เท่านั้น

Mouse คืออุปกรณ์ที่ส่งข้อมูล อย่างน้อย 3 ชนิดไปยัง Operating System

- ตำแหน่งสัมพัทธ์ที่เลื่อนไปตามแกน x
- ตำแหน่งสัมพัทธ์ที่เลื่อนไปตามแกน y
- สถานะของปุ่ม

หมายเหตุ เครื่องหมาย Mouse ที่ปรากฏบนจอภาพเป็นสิ่งที่สร้างขึ้นโดย software ซึ่งอาศัยข้อมูลดังกล่าวข้างต้น

Event Mouse

Mouse Event จะเกิดขึ้นก็ต่อเมื่อ ข้อมูลใดข้อมูลหนึ่งใน 3 ชนิดได้เปลี่ยนสถานะ

ในขั้นตอนนี้ Mouse จะส่งสัญญาณ Interrupt ไปยังระบบปฏิบัติการ พร้อมกับข้อมูลของสถานะในปัจจุบัน

หลังจากนั้นระบบปฏิบัติการจะปรับตำแหน่งของเครื่องหมาย Mouse บนอุปกรณ์แสดงผล พร้อมกับ ส่งข้อมูล Mouse ไปยัง โปรแกรมกราฟิกเพื่อประมวลผลต่อไป

สำหรับระบบปฏิบัติการ Windows จะเรียก Mouse Event ว่า Message
ซึ่งจะส่งไปโปรแกรมประยุกต์ พร้อมกับ Interrupt อื่นๆ

Mouse Message ประกอบด้วย Message ID ซึ่งระบุการเปลี่ยนแปลงที่เกิดขึ้น
เช่น WM_MOUSEMOVE, WM_LBUTTONUP และ
lParam และ wParam (Parameter ขนาด 32 และ 16 บิต ตามลำดับ)
ซึ่งโดยปกติ lParam จะเก็บค่าพิกัด X และ Y ขณะ นั้นไว้ที่ตำแหน่ง 16 บิตล่าง และ 16 บิตบน ตามลำดับ

Event Loop

ในระบบปฏิบัติการ Windows โปรแกรมกราฟิก จำเป็นจะต้องมีการวนลูป (Event Loop) เพื่อรอรับเหตุการณ์ดังนี้

```
while (GetMessage (&msg, NULL, 0, 0))
{
        TranslateMessage(&msg);
        DispatchMessage(&msg);
}
```

Windows API Reference

```
BOOL GetMessage(lpmsg, hwnd, uMsgFilterMin, uMsgFilterMax)

MSG FAR* lpmsg;        /* address of structure with message   */
HWND hwnd;             /* handle of the window                */
UINT uMsgFilterMin;    /* first message                       */
UINT uMsgFilterMax;    /* last message                        */
```

เมื่อโปรแกรมกราฟิกรับ MouseEvent (Message) ได้จาก DispatchMessage ก็จะส่งไปยังฟังก์ชันประมวลผล Message ต่อไป (Callback Function) ดังนี้

```
LRESULT wnddisp2d :: WndProc (UINT iMessage, WPARAM wParam, LPARAM lParam)
{
        switch (iMessage)
        {
                case WM_LBUTTONDOWN: /* Process Mouse Event when the Left Button is pressed*/
                                     SetCapture(m_hWnd);
                                     m_px= LOWORD (lParam);
                                     m_py= HIWORD (lParam);
                                     break;
        }
}
```

Overview of Windows Programming (I)

Main() or WinMain()

```
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmd, int nCmdShow)
{
   static char          szAppName[] = "423406 Computer Graphics" ;
   HWND                 hwnd;
   MSG                  msg;
   WNDCLASS             wndclass;
   HANDLE               hAccel;

   if (!hPrevInstance)
   {
      wndclass.style                = CS_HREDRAW | CS_VREDRAW ;
      wndclass.lpfnWndProc          = (WNDPROC)WndProc ;
      wndclass.cbClsExtra           = 0 ;
      wndclass.cbWndExtra           = 0 ;
      wndclass.hInstance            = hInstance ;
      wndclass.hIcon                = LoadIcon (NULL, IDI_APPLICATION);
      wndclass.hCursor              = LoadCursor (NULL, IDC_ARROW) ;
      wndclass.hbrBackground        = (HBRUSH)(COLOR_APPWORKSPACE + 1);
      wndclass.lpszMenuName         = MAKEINTRESOURCE (PROGRAM_MENU);
      wndclass.lpszClassName        = szAppName ;

      RegisterClass (&wndclass);
   }

   RegisterChildWindows (hInstance);
```

Overview of Windows Programming (II)

```
hwnd = CreateWindow (szAppName,
                     "Computer Graphics: Chapter 1",
                     WS_OVERLAPPEDWINDOW,
                     CW_USEDEFAULT, CW_USEDEFAULT,
                     640, 480,
                     NULL, NULL,
                     hInstance, NULL);

  ShowWindow (hwnd, nCmdShow);
  UpdateWindow (hwnd);

  hAccel      = LoadAccelerators (hInstance, MAKEINTRESOURCE (PROGRAM_ACCEL));
  m_hInstance  = hInstance;

  while (GetMessage (&msg, NULL, 0, 0))
  {
     if (!TranslateAccelerator (hwnd, (HACCEL) hAccel, &msg))
     {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
     }
  }

  UnRegisterChildWindows (hInstance);

  return msg.wParam ;
}
```

```
LPCSTR lpszClassName;   /* address of registered class name */
LPCSTR lpszWindowName;  /* address of window text   */
DWORD dwStyle;  /* window style */
int x;  /* horizontal position of window    */
int y;  /* vertical position of window  */
int nWidth; /* window width */
int nHeight;    /* window height    */
HWND hwndParent;    /* handle of parent window  */
HMENU hmenu;    /* handle of menu or child-window identifier   */
HINSTANCE hinst;    /* handle of application instance   */
void FAR* lpvParam; /* address of window-creation data  */
```

Callback Function

```
LRESULT  FAR  PASCAL WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   int ret;

   switch (message)
   {
      case WM_CRETE        :  HDC        hdc;
                              hdc = GetDC (hwnd)
                              Ellipse (hdc, 0, 20, 400, 300);
                              ReleaseDC (hwnd, hdc);
                              break;

       case WM_DESTROY     :  PostQuitMessage (0);
                              return 0;
   }

   return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

ฟังก์ชัน Callback นี้จะวาดวงรี ซึ่งมีแกนขนาด 10 คูณ 20 จุดภาพ ณ ตำแหน่งมุมบนซ้ายของจอ (0, 0)

Raster Algorithm

- Frame Buffer Class
- Simple Graphic Primitive Algorithms
  - Line Drawing Algorithms (Direct v.s. Integer Arithmetic)
  - Circle Drawing Algorithms
  - Rasterization of Arbitrary Curves
  - Polynomial Curve and Spline Drawing Algorithms
- Filled-Area Primitives
  - Polygon Filling
  - Flood-Fill Algorithm
  - Inside-Outside Tests
- Picture Approximation using Halftone
- Text Generation

Device Independent

เทคนิคหนึ่งที่พยายามทำให้โปรแกรมกราฟิก ขึ้นอยู่กับความสามารถของอุปกรณ์ให้น้อยที่สุด คือ พยายามสร้างฟังก์ชันกราฟิกพื้นฐานขึ้นมาเอง

MFC Library Reference
CDC::Ellipse

Draws an ellipse.

```
BOOL Ellipse(
    int x1,
    int y1,
    int x2,
    int y2
);
BOOL Ellipse(
    LPCRECT lpRect
);
```

ถึงแม้ว่าจะมีผลทำให้ความเร็วของการทำงานลดลง แต่โปรแกรมจะมีความยืดหยุ่นสูง เทคนิคนี้ ใช้กันมาก โดย ผู้เขียนโปรแกรม Micro-controller สำหรับแสดงผล และ ผู้ผลิตระบบพัฒนาโปรแกรม (Integrated Development Environment : IDE) เช่น ชุดคำสั่ง GDI ของ Microsoft Foundation Class เพื่อ อำนวยความสะดวกให้กับ ผู้เขียนโปรแกรม ได้เรียกใช้

http://msdn.microsoft.com/developercenters/

Flame Buffer Class

บทนี้เน้นแนวคิด โครงสร้างข้อมูล และขั้นตอนวิธี การสร้างชุดคำสั่งกราฟิกประเภท Device Independent โดยเฉพาะอย่างยิ่ง การนำ array ชนิด byte (unsigned character) มาสร้างหน่วยความจำแสดงผล (Frame Buffer) ซึ่งออกแบบในลักษณะ OOP ดังนี้

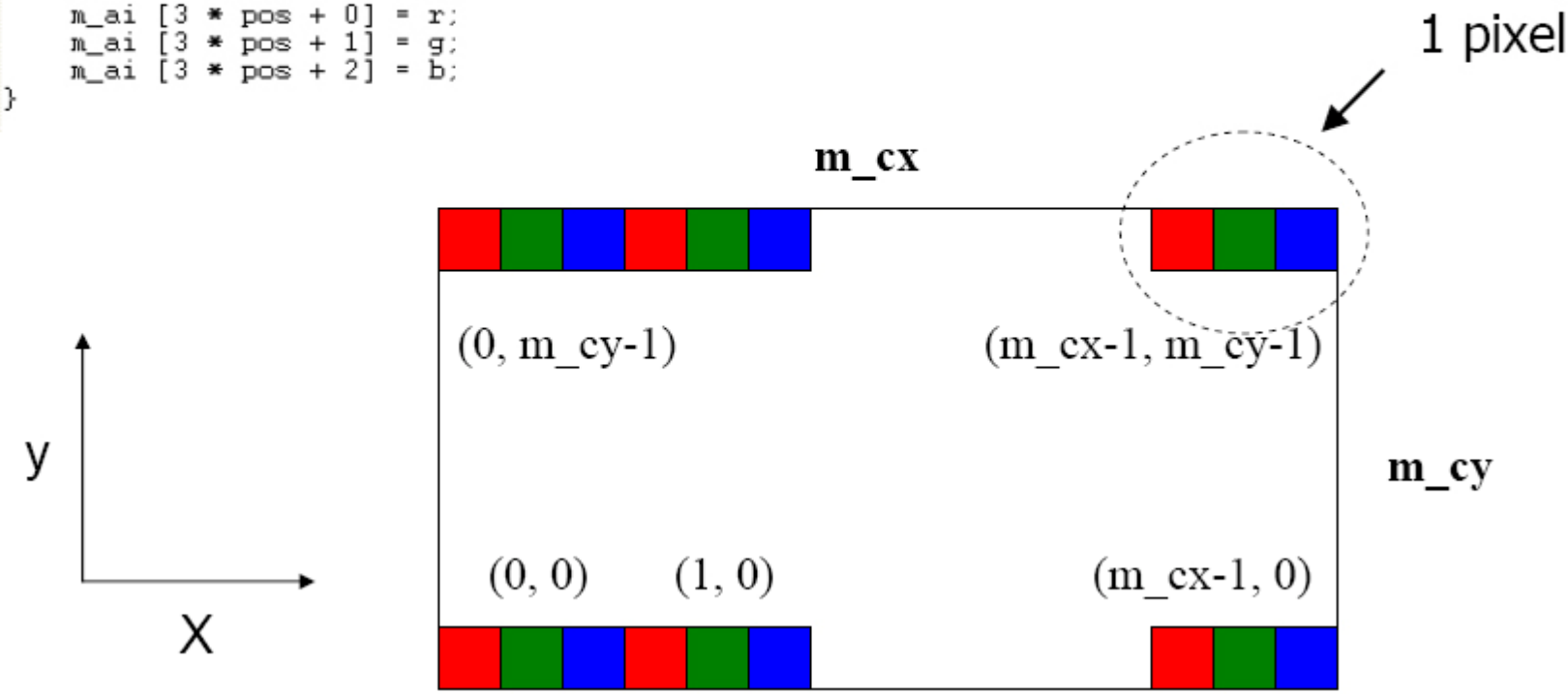
```
#include <windows.h>

class cfbuffer
{
public:
     cfbuffer ();
    ~cfbuffer ();

public:
    void    init (long cx, long cy);
    void    clearresource (void);

public:
    void    setpixel (long ix, long iy, unsigned char i);
    void    setpixel (long ix, long iy, unsigned char  r, unsigned char  g, unsigned char  b);

    void    getpixel (long ix, long iy, unsigned char *i);
    void    getpixel (long ix, long iy, unsigned char *r, unsigned char *g, unsigned char *b);

    void    clrscr (void);
    void    display (HDC hdc);

protected:
    unsigned char   *m_ai;      //  rgb components of size 3 * m_cx * m_cy

    long             m_cx;      //  frame buffer size
    long             m_cy;
};
```

Pixel Coordinates

การอ้างถึงตำแหน่งใน frame buffer โดยกำหนดค่าพิกัด (x, y) สามารถทำได้ ด้วยการคำนวณ ตำแหน่งใน array ขนาด 1 มิติ ดังนี

```
void cfbuffer::setpixel (long ix, long iy, unsigned char r, unsigned char g, unsigned char b)
{
    long    pos;

    pos = ix + (m_cy - iy - 1) * m_cx;

    m_ai [3 * pos + 0] = r;
    m_ai [3 * pos + 1] = g;
    m_ai [3 * pos + 2] = b;
}
```

มาได้ยังไง !!!



Device Dependent

จากการประกาศ class จะเห็นว่ามีเพียงฟังก์ชันเดียว ที่อ้างอิงกับระบบปฏิบัติการ Windows นั้นคือ การเรียกใช้ pointer (handle) ไปยัง Device Context (DC)

```
void cfbuffer::display (HDC hdc)
{
    BITMAPINFO  bmi;

    ::memset (&bmi, 0, sizeof (bmi));

    bmi.bmiHeader.biSize            = sizeof (BITMAPINFOHEADER);
    bmi.bmiHeader.biWidth           = (int) m_cx;
    bmi.bmiHeader.biHeight          = (int) m_cy;
    bmi.bmiHeader.biPlanes          = 1;
    bmi.bmiHeader.biBitCount        = 24;
    bmi.bmiHeader.biCompression     = BI_RGB;
    bmi.bmiHeader.biSizeImage       = m_cx * m_cy * 3;
    bmi.bmiHeader.biXPelsPerMeter   = 0;
    bmi.bmiHeader.biYPelsPerMeter   = 0;
    bmi.bmiHeader.biClrUsed         = 0;
    bmi.bmiHeader.biClrImportant    = 0;

    ::SetDIBitsToDevice (hdc, 0, 0, m_cx, m_cy, 0, 0, 0, m_cy,
                         m_ai, &bmi, DIB_RGB_COLORS);
}
```

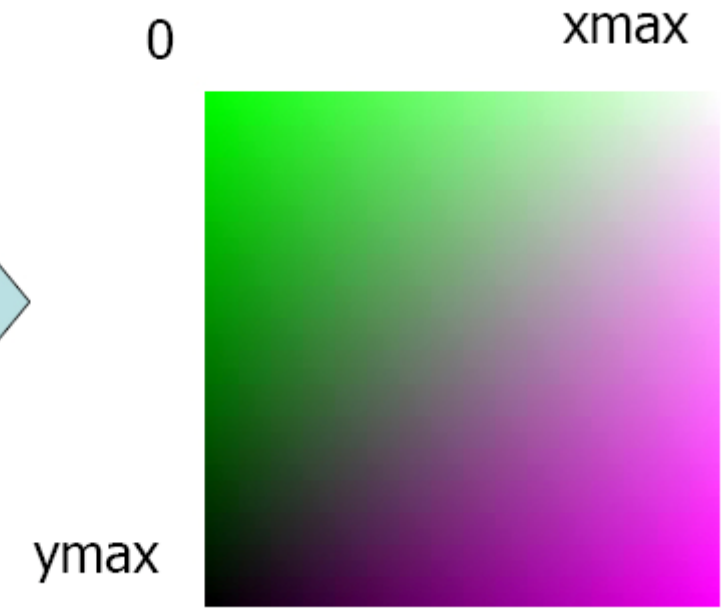
การส่งข้อมูลใน Frame Buffer ออกทางจอภาพทำได้โดยเรียกฟังก์ชันของ OS

Implementation Example

การเรียกใช้ class cfbuffer สามารถทำได้ดังนี้

1) กำหนดขนาดของหน่วยความจำ (init)
2) ตั้งค่าปริยายให้กับหน่วยความจำเป็นศูนย์ (clrscr)
3) กำหนดค่าสีของแต่ละจุด (setpixel)
4) แสดงผลออกหน้าจอ (display)

```
void CMainFrame::TestFrameBuffer (HWND hwnd)
{
    HDC         hdc;
    cfbuffer    fbuffer;
    long        x, y;

    hdc     = ::GetDC (hwnd);

    fbuffer.init (256, 256);
    fbuffer.clrscr ();

    for (y = 0; y < 256; y ++)
    {
        for (x = 0; x < 256; x ++)
        {
            fbuffer.setpixel (x, y, x, 255-y, x);
        }
    }

    fbuffer.display (hdc);

    ::ReleaseDC (hwnd, hdc);
}
```



Line Drawing Algorithm

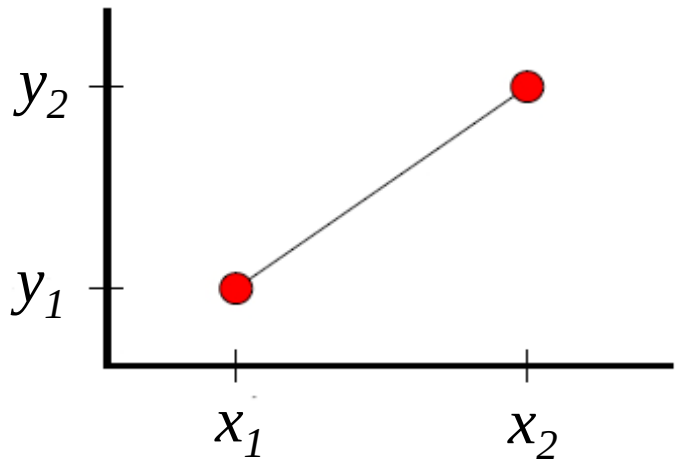
การวาดเส้นตรงบนจอภาพสามารถแสดงได้ โดยเริ่มจากการศึกษาสมการเส้นตรง

$$y = m \cdot x + b$$

โดยที่ m และ b คือ ความชัน และ จุดตัดแกน y ของเส้นตรง ตามลำดับ
ถ้ากำหนด จุดปลายสองจุด คือ (x_1, y_1) และ (x_2, y_2) ความสัมพันธ์เขียนได้เป็น

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - m \cdot x_1$$



Digital Differential Analyzer (DDA)

จากการสังเกตพบว่า ความแตกต่างตามแกน x ($\Delta x = x_2 - x_1$) และ แกน y ($\Delta y = y_2 - y_1$) จะสัมพันธ์กับค่าความชัน m ดังนี้

จาก $$m = \frac{y_2 - y_1}{x_2 - x_1}$$ นั่นคือ $$m = \frac{\Delta y}{\Delta x}$$

จะได้ว่า $$\Delta y = m \cdot x$$ $$\Delta x = \frac{m}{\Delta y}$$

Digital Differential Analyzer (DDA)

เนื่องจากใน Computer Graphics เราจะพิจารณา Frame Buffer ซึ่งมีระบบปริภูมิ แบบ Discrete นั่นคือ Δ จะมีค่าเป็นจำนวนเต็มหน่วยเท่านั้น (pixels) ดังนั้น แบ่ง การพิจารณาเป็น 2 กรณี **เมื่อ m และ Δx มีค่าเป็นบวก**

กรณีที่ 1) m น้อยกว่าหรือเท่ากับ 1

$$y_{k+1} = y_k + m$$

กรณีที่ 2) m มากกว่า 1

$$x_{k+1} = x_k + \frac{1}{m}$$

ถ้า Δx มีค่าเป็นลบก็เพียงแต่เปลี่ยนเครื่องหมายหน้า m จาก + เป็น -

Digital Differential Analyzer (DDA)

ในทำนองเดียวกัน กรณีที่ m มีค่าเป็นลบ จะพิจารณาจากค่าสัมบูรณ์ของ m

กรณีที่ 1)|m| น้อยกว่าหรือเท่ากับ 1

$$y_{k+1} = y_k + m$$

กรณีที่ 2)|m| มากกว่า 1

$$x_{k+1} = x_k + \frac{1}{m}$$

DDA Algorithm จะเร็วกว่าทำการคำนวณจากสมการตรงๆ โดยการตัดกระบวนการคูณ (พิกัด x กับความชัน m) ทิ้ง แล้วแทนที่ด้วยการบวก จำนวนจริง

อย่างไรก็ดี ข้อควรระวังคือ หากเส้นตรงมีความยาวมาก ความผิดพลาดจะสะสม จนกระทั่งเบี่ยงเบนไปจาก เส้นตรงที่ต้องการอย่างเด่นชัด อาจแก้ไขโดยการบวกค่าสะสมเป็นจำนวนจริง แล้วปัดเศษเป็นจำนวนเต็ม ก่อนจะวาดจุดภาพ ทว่าวิธีนี้จะใช้เวลา CPU ค่อนข้างมาก

Digital Differential Analyzer (DDA)

DDA Algorithm สามารถเขียนด้วย C++ บน class cfbuffer ได้ดังนี

```
void cfbuffer::lineDDA (int x1, int y1, int x2, int y2)
{
    int     dx, dy, steps, k;
    double  xincrement, yincrement;
    double  x, y;

    dx  = x2 - x1;
    dy  = y2 - y1;

    steps = (abs (dx) > abs (dy)) ? abs (dx) : abs (dy);

    xincrement  = (double) dx / steps;
    yincrement  = (double) dy / steps;

    x   = x1;
    y   = y1;

    setpixel ((int) (x + 0.5), (int) (y + 0.5), 0, 0, 0);

    for (k = 0; k <= steps; k ++)
    {
        x   = x + xincrement;
        y   = y + xincrement;

        setpixel ((int) (x + 0.5), (int) (y + 0.5), 0, 0, 0);
    }
}
```