รายการอ้างอิง

1. David C. Kay, John R. Levin. Graphics File Formats. United States of America : McGraw-Hill, 1992.
2. James M. Anderson , Edward M. Mikhail . Introduction to Surveying : McGraw-Hill,1979.
3. Luse ,Marv Bitmapped Graphics Programming In C++. United States of America : Addison - Wesley , 1993.
4. Peter F. Dale, John D. Mclaughlin. Land Information Management. New York : Oxford University Press , 1988.
5. Rosenfeld A. and Kok A. C. Digital Picture Processing Second Edition. New York : Academic Press,1982.
6. Sukit Viseshsin. Automated Assignment of Hieght Values to Rasterized Contour Data . ,1988.
7. William M. Newman, Robert F. Sproull. Principle of Interactive Computer Graphics Second Edition . Singpore : McGraw-Hill,1979.

ภาคผนวก

ภาคผนวก ก

โครงสร้างข้อมูลแบบ Raster และแบบ Vector

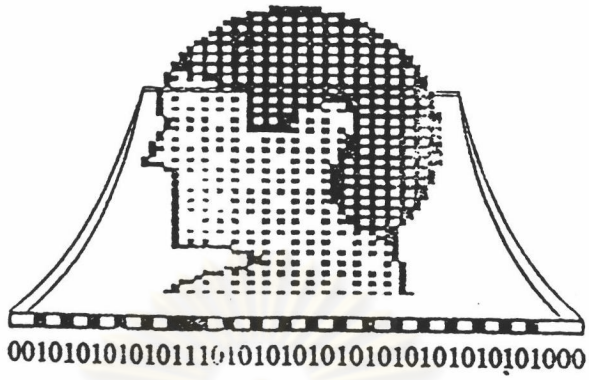
ในการแสดงรูปภาพจะมี 2 วิธีการซึ่งแตกต่างกัน คือ แบบ Raster (บางครั้งเรียก Bitmap หรือ Pixel maps) และ แบบ Vector แฟ้มข้อมูลกราฟิกสามารถใช้หนึ่งหรือทั้งสองวิธีในการแสดงขึ้นอยู่กับความสะดวกและเหมาะสม

โครงสร้างแบบ Raster

David และ John (1992)[1] ได้ให้ความหมาย โครงสร้างข้อมูลแบบ Raster ว่า โครงสร้างข้อมูลแบบ Raster สามารถอธิบายได้โดยการนำภาพมาหนึ่งภาพแล้วตีกริด (grid) เข้าไปในภาพจากนั้นจะบันทึกค่าแสง (สว่าง, มืด, หรือสี) แต่ละช่องของกริดเรียงไป ดังภาพ ก-1 ซึ่งแสดงข้อมูลในหนึ่งบรรทัดของภาพซึ่งมีความละเอียดต่ำ

โครงสร้างข้อมูลแบบนี้ ใช้ได้ดีในภาพที่มีการเปลี่ยนแปลงของสีและรูปร่างมาก เช่น ภาพถ่าย ภาพเขียน และ จอภาพของคอมพิวเตอร์

ข้อเสียของโครงสร้างข้อมูลแบบนี้คือ ขนาดของภาพ และความละเอียดของในแฟ้มข้อมูลภาพ ซึ่งต้องการเนื้อที่เก็บข้อมูลหลายเมกะไบต์ และต้องการหน่วยความจำหลายเมกะไบต์ในกระบวนการแสดงภาพ ด้วยเหตุผลเหล่านี้ทำให้ต้องมีการจัดเก็บข้อมูลในรูปที่ถูกบีบอัดข้อมูลแล้ว



รูป ก-1 ภาพแสดงโครงสร้างข้อมูลแบบ Raster (David และ John ,1992)[1]

<u>โครงสร้างแบบ Vector</u>

โครงสร้างข้อมูลแบบ Vector จะบรรยายภาพด้วยจุดของเส้น หรือรูปทรง หรือขอบเขตที่จะเติมสี

David และ John (1992)[1] ได้อธิบายว่า แฟ้มข้อมูลแบบเวกเตอร์จะมีลักษณะคล้ายโปรแกรมคอมพิวเตอร์ คือ แฟ้มข้อมูลสามารถใช้เขียนเป็นคล้ายชุดคำสั่งภาษาอังกฤษและข้อมูลอยู่ใน ASCII สามารถแก้ไขแฟ้มข้อมูลด้วยโปรแกรม word processor ได้เลย ตัวอย่าง การเก็บข้อมูลแบบโครงสร้าง Vector เช่น DXF

โครงสร้างแบบเวกเตอร์จะใช้งานที่เป็นรูปลายเส้น เช่น การออกแบบด้วยคอมพิวเตอร์ (Computer-aided design : CAD) โครงสร้างแบบนี้จะมีข้อจำกัดในการทำมากกว่าแบบ Raster

ภาคผนวก ข

การบีบอัดข้อมูล (Compression Data)

เทคนิคการบีบอัดข้อมูลในปัจจุบันสามารถแบ่งได้เป็น 2 ประเภทใหญ่ๆ คือ

1. Lossy data compression เป็นการบีบอัดข้อมูลที่ยอมให้ความถูกต้องของข้อมูลสูญหายไปบ้าง เพื่อเพิ่มความเร็วของการบีบอัดข้อมูล

2. Lossless data compression เป็นการบีบอัดข้อมูลที่ให้ความสำคัญกับความถูกต้องของข้อมูล คือ ต้องเหมือนต้นฉบับทุกประการ

โดยแต่ละเทคนิคมีรายละเอียดย่อยดังต่อไปนี้

1. Lossy data compression

1.1 Lossy compression

David และ John (1992)[1] ได้อธิบายวิธี Lossy compression ไว้ว่า เป็นวิธีการที่ใช้เทคนิคซึ่งไม่สนใจข้อมูลเดิมว่าจะต้องมีการสูญเสีย เพราะเชื่อว่าข้อมูลเหล่านั้นไม่จำเป็นในการใช้งานภายหลัง เทคนิคนี้จะไม่นำไปใช้งานกับเรื่องที่รายละเอียดของข้อมูลมีความสำคัญ เช่น ภาพในทางการแพทย์

สิ่งที่ไม่จำเป็นต้องให้ความสนใจกับรายละเอียดของภาพมากนัก จะนำวิธี Lossy compression มาใช้งาน อาทิเช่น รายละเอียดบริเวณขอบภาพ หรือรายละเอียดช่องว่างระหว่างสีในการแสดงภาพของโทรทัศน์ เพราะเกินขีดความสามารถของสายตา

2. Lossless data compression

2.1. Binary and symbolic coding

ข้อมูลภาพสามารถเข้ารหัส (Encode) เป็นรูปแบบของสัญลักษณ์ (Symbolic) หรือ ไบนารี(Binary) หรือผสมระหว่างสองรูปแบบ ตัวอย่างคำสั่งและข้อมูลใน Postscript file เช่น การเก็บข้อมูลในรูปแบบสัญญาลักษณ์ สามารถอ่านได้จาก

Word Processor ไฟล์ TIFF เก็บข้อมูลในรูปแบบ ไบนารี ซึ่งง่ายต่อการอ่าน คือ ใช้ Debugger หรือ Binary interpreter ซึ่ง David และ John (1992)[1] ได้อธิบายไว้พอสรุปได้ว่า

การเข้ารหัสสัญลักษณ์ (Symbolic coding) ส่วนมากอยู่ในรูป ASCII จะมีประสิทธิภาพน้อยเมื่อเทียบกับวิธีการอื่น ๆ ส่วนมาก Vector file และ ภาษาที่บรรยายภาพกราฟิก จะเก็บในรูปแบบสัญลักษณ์ (Symbolic Form) อย่างไรก็ตาม ความด้อยประสิทธิภาพของ ASCII แปรผกผันกับ ประสิทธิภาพในการแสดง Vector ที่สูงขึ้น

การเข้ารหัสแบบไบนารี่ (Binary coding) ส่วนใหญ่จะใช้ใน ข้อมูลบิตแม็พ(Bitmap data) ลักษณะเด่น คือ Postscript file ซึ่งโดยทั่วไปจะใช้ ASCII สำหรับบิตแม็พ เพราะฉะนั้น Postscript file ที่เก็บข้อมูลบิตแม็พจะมีขนาดใหญ่กว่าไฟล์บิตแม็พ สำหรับข้อมูลชุดเดียวกัน

2.2 Run-length compression

David และ John (1992)[1] ได้อธิบายวิธีการ Run-length compression ว่า เป็นหนึ่งในวิธีการบีบอัดข้อมูลที่ง่าย คือ การเข้ารหัส Run-length ในรูปแบบนี้ชุดของข้อมูลที่ซ้ำ (หรือ ค่าของจุดภาพ) จะถูกแทนด้วย หนึ่งค่า และ จำนวนนับ ตัวอย่างเช่น ใช้ตัวอักษรแสดงแทนค่า โดยชุดของข้อมูลเป็น abbbbbbbccddddeeddd จะถูกแทนด้วย 1 a7b2c4d2e3d กระบวนการนี้จะง่ายต่อการใช้เป็นเครื่องมือ และทำงานได้ดี ในข้อมูลที่มีตัวอักษรซ้ำกันมาก ๆ วิธีการบีบอัดข้อมูลนี้จะดีสำหรับรูปภาพที่มีสีซ้ำ ๆ กันเป็นพื้นที่กว้าง ๆ

วิธีการหนึ่งที่เป็นที่นิยมของการบีบอัดข้อมูลแบบ Run-length คือ "PackBits" ใช้สำหรับข้อมูลบิตแม็พบนเครื่อง Apple MacIntosh การเข้ารหัสแบบ PackBits สำหรับข้อมูล 8-บิต จะใช้เก็บเป็น 2 ไบต์ โดยไบต์แรก จะบรรจุตัวเลข n ระหว่าง -127 กับ -1 ซึ่งได้จากการนับ -n+1 ไบต์ที่สอง จะบรรจุค่าที่ซ้ำ ค่าที่ไม่ซ้ำ เช่น abcde จะถูกแสดงโดย 1 ไบต์ เริ่มด้วยรหัส m ระหว่าง 0 ถึง 127 ซึ่งได้จากความยาวของตัวอักษร m+1

(m=4สำหรับตัวอย่างนี้) การซ้ำของข้อมูลจะต้องยาวไม่เกิน 128 ถ้าเกิน จะถูกแบ่งเป็น 2 ส่วน โดยทั่วไป การบีบอัดข้อมูลแบบนี้จะไม่สามารถข้ามจากบรรทัดหนึ่งไปยังบรรทัดถัดไป

การเข้ารหัสแบบ Run-length ถูกใช้มากในไฟล์รูปแบบบิตแม็พ เช่น TIFF, GEM และ PCX

2.3 Huffman encoding

Huffman encoding (Huffman 1952) เป็นรูปแบบคำสั่งในการบีบอัดข้อมูล กระบวนการนี้ถูกสร้างขึ้นในปี 1952 จากการศึกษาของ David และ John (1992)[1] ได้อธิบายวิธีการ Huffman encoding ว่าใช้วิธีการพื้นฐาน คือ กำหนดรหัสไบนารีลงไปในแต่ละ unique value ด้วยรหัสที่เปลี่ยนแปลงในด้านความยาว รหัสที่สั้นกว่าจะมีความถี่ที่จะถูกใช้มากกว่า การกำหนดค่านี้จะถูกเก็บในตารางการแปลง ซึ่งจะส่งค่าไปยังซอฟแวรท์ ก่อนจะถอดรหัส

ความละเอียดของขั้นตอนวิธี ขึ้นอยู่กับรูปแบบของไฟล์ที่จะใช้ Huffman จะเป็นทางเลือกที่ไม่ดีสำหรับไฟล์ที่มีความยาวของการซ้ำค่ามาก ๆ ซึ่งจะใช้การบีบอัดข้อมูลแบบ Run-length หรือวิธีการเข้ารหัสแบบอื่นดีกว่า

2.4 LZW Compression

LZW จะไม่เหมือนกับ Huffman คือ LZW ไม่ต้องการโครงสร้างตารางรหัสล่วงหน้า แต่เริ่มต้นด้วยตารางง่าย ๆ ซึ่งทำให้ตารางมีประสิทธิภาพมากขึ้นด้วยรหัสที่ยาวซึ่งถูกดัดแปลงให้เหมาะสมกับข้อมูล

Lues และ Marv (1993)[3] ได้อธิบายว่า LZW เริ่มต้นด้วยตารางกับหนึ่งรหัสที่ลงไว้สำหรับค่าที่เป็นไปได้ของข้อมูล เช่น 256 สำหรับข้อมูล 8-บิต และรหัสที่จะใส่ในตาราง แต่ละค่าจะเป็นอิสระไม่ซ้ำกัน จึงจำเป็นต้องกำหนดขนาดใหญ่สุดของตาราง ดังนั้นความยาวของรหัสจึงคงที่

ตัวอย่างตารางรหัส เช่น ข้อมูลเป็น abaaaaccaaad ให้ใช้ค่าของข้อมูลเป็น 2 บิต การเข้ารหัสและถอดรหัส ของLZW จะเริ่มต้นด้วยตารางเดียวกัน คือ

a : 00

b : 01

c : 10

d : 11

การเข้ารหัส LZW จะมีอัตราส่วนการบีบอัดอยู่ระหว่าง 1:1 ถึง 3:1 ถ้ารูปที่มีรูปแบบมาก ๆ บางครั้งอาจมีอัตราส่วนการบีบอัดมากถึง 10:1 ได้ สำหรับภาพที่มีสิ่งลบกวน (Noisy) หรือ การกระจายของข้อมูลแบบสุ่มอิสระจะเป็นการยากต่อการบีบอัดข้อมูลแบบ LZW การบีบอัดข้อมูลแบบ LZW จะถูกใช้ในไฟล์ GIF และ TIFF

2.5 Arithmetic compression

David และ John (1992)[1] ได้อธิบาย Arithmetic compression ว่าจะเหมือนกับ Huffman coding ที่ใช้รหัสสั้นๆ สำหรับข้อมูลที่มีความถี่ในการซ้ำกัน และใช้รหัสยาวสำหรับข้อมูลที่ไม่ซ้ำ อย่างไรก็ตามวิธีนี้จะมีประสิทธิภาพมากเหมือนกับวิธี LZW ถ้าข้อมูลเรียงเป็นระบบ

Arithmetic compression จะลดขนาดของไฟล์ได้มากน้อยขึ้นอยู่กับความละเอียดในการทำ Statistical model คือ เป็นการอ่านข้อมูลและเข้ารหัสตัวอักษรในเวลาเดียวกัน ซึ่งรหัสที่ได้จะขึ้นกับความน่าจะเป็นของการซ้ำตัวอักษร

ภาคผนวก ค.

การสร้างเส้นตรงระหว่างจุด

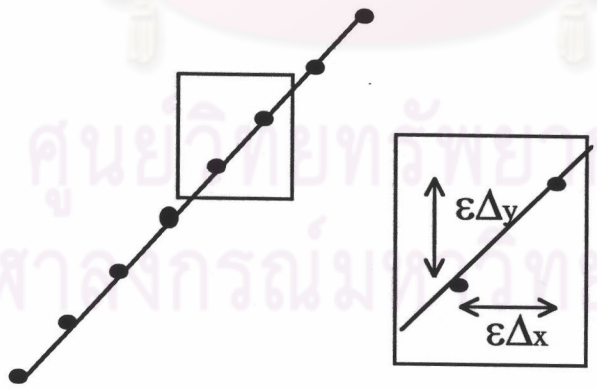
จากการศึกษาการสร้างภาพเส้นตรงบนจอภาพคอมพิวเตอร์ของ William และ Robert (1979)[7] มีหลายวิธีพอสรุปได้ดังนี้

1. The Symmetrical DDA

DDA (The Digital Differential Analyzer) จะสร้างเส้นจากสมการ Differential สมการคือ

$$\frac{dy}{dx} = \frac{\Delta y}{\Delta x}$$

DDA จะทำงานบนหลักการที่ว่า เราจะคำนวณการเพิ่มของ X และ Y พร้อมๆ กันทีละขั้น ถ้าเป็นจอภาพในจินตนาการ คือมีความละเอียดสูง ค่า X และ Y จะเพิ่มด้วย $\varepsilon\Delta X$ และ $\varepsilon\Delta Y$ ซึ่งค่า ε มีค่าน้อยมาก ดังรูปที่ ค-1

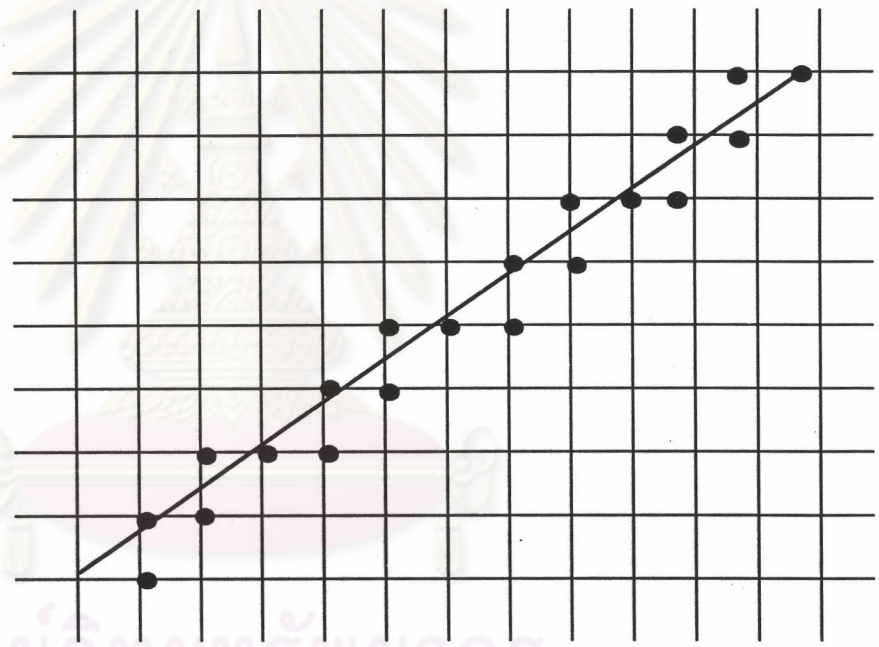


รูปที่ ค-1 วิธีการคิดค่าเพิ่มสำหรับการสร้างเส้นตรง (William และ Robert ,1979)[7]

ในความเป็นจริง จอภาพจะมีความละเอียดจำกัด ทำให้เรามักจะสร้างเฉพาะจุดที่มีตำแหน่ง ซึ่งทำได้โดยใช้เลขจำนวนเต็มที่ใกล้เคียงที่สุด หลังจากได้คำนวณการเพิ่มขึ้นในแต่ละขั้น หลังจากปัดเศษแล้ว เราจะได้เส้นจุดผลลัพธ์ของ X และ Y ดังรูป ค-2

การสร้างเส้นตรงด้วยวิธี DDA จะขึ้นอยู่กับการเลือกใช้ ค่า ε ในกรณีของ Symmetrical DDA เราจะใช้

$$2^{n-1} \leq \max|\Delta X, \Delta Y| < 2^n$$



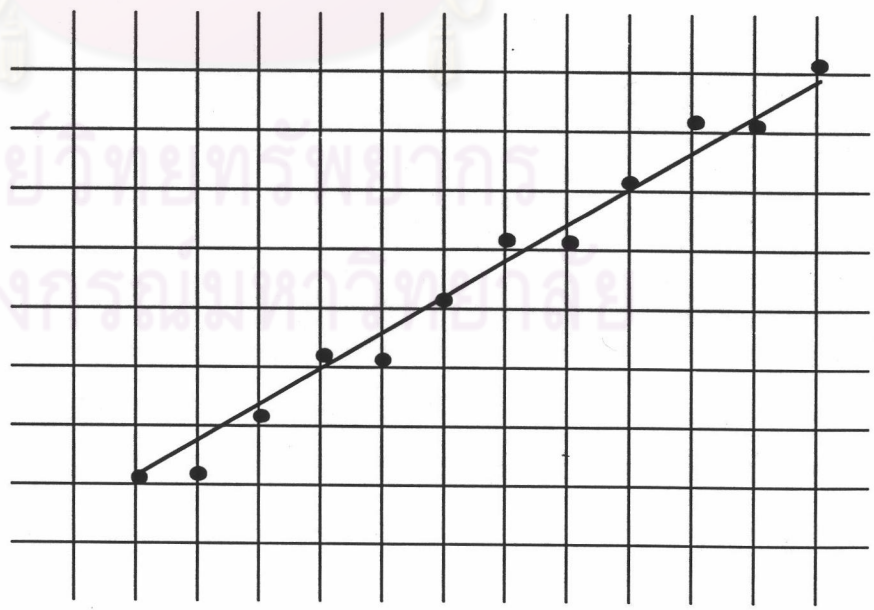
รูป ค-2 การสร้างเส้นตรงโดยวิธี The Symmetrical DDA (William และ Robert ,1979) [7]

2. The Simple DDA

วิธี Symmetrical DDA ดังตัวอย่างรูป ค-3 จะใช้ ค่ากำลังของสองแทนค่าความยาวโดยประมาณ จากหลักการของ DDA จะใช้ความยาวของเส้นโดยประมาณและค่า ε เพื่อคำนวณค่า $\varepsilon\Delta X$ และ $\varepsilon\Delta X$ โดยไม่ให้เกินหนึ่งหน่วยสำหรับวิธีการ Simple DDA เรา

จะเลือกความยาวของเส้น โดยประมาณเท่ากับค่ามากสุด ระหว่าง ΔX กับ ΔY เพื่อให้ค่า $\varepsilon\Delta x$ และ $\varepsilon\Delta x$ อยู่ในหนึ่งหน่วย วิธีการนี้จะได้

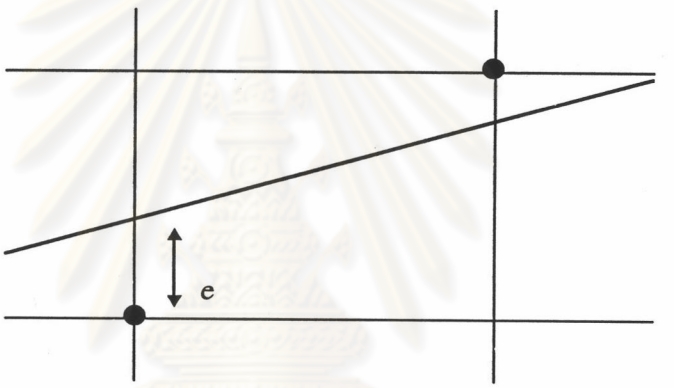
```
DDA (int x1,int y1,int x2,int y2)
{  int  length,i;
   float  x,y,xincrement,yincrement;
   length = abs(x2-x1);
   if (abs(y2-y1)>length)  length=abs(y2-y1);
   xincrement=(x2-x1)/length;
   yincrement=(y2-y1)/length;
   x=x1+0.5;y=y1+0.5;
   for (i=1;i<=lenght;i++)
   {   plot(x,y);
       x=x+xincrement;
       y=y+yincrement;
   }
}
```



รูป ค-3 การสร้างเส้นตรงโดยวิธี The Simple DDA(William และ Robert ,1979)[7]

3. Bresenham's Algorithm

วิธีการนี้เป็นวิธีการที่น่าสนใจ และงานวิจัยนี้ก็ใช้วิธีการนี้ในการสร้างเส้นตรง เพราะให้ความถูกต้องใกล้เคียงมากกว่า วิธีการนี้จะคล้ายกับวิธี Simple DDA คือ การทำงานจะวนซ้ำ โดยแต่ละรอบจะเปลี่ยนค่าพิกัดที่ ± 1 ส่วนค่าพิกัดอีกค่าจะเปลี่ยนแปลงหรือไม่ขึ้นอยู่กับค่า Error Term (e) เป็นค่าระยะที่วัดตั้งฉากกับแกนที่มีการเคลื่อนที่มากระหว่างแนวที่เส้นผ่านกับจุดภาพที่สร้างไว้ ตามรูป ค-4 ซึ่งแกน X เป็นแกนที่มีการเคลื่อนที่มาก ดังนั้นค่า (e) จึงวัดขนานกับแกน Y เส้นผ่านไปยังจุดภาพที่สร้าง



รูปที่ ค-4 Bresenhem's algorithm : Error Term e (William และ Robert ,1979) [7]

เป็นการวัดระยะจากแนวที่ขึ้นในการวนซ้ำแต่ละรอบ ค่า Error Term (e) จะถูกบวกด้วยความชันของเส้น ($\Delta X/\Delta Y$) เครื่องหมายของ e เป็นตัวบอกการเพิ่มของค่าพิกัด Y ของจุดภาพที่จะแสดง ซึ่งสามารถบรรยายเป็นชุดคำสั่งบนภาษาซีได้ดังนี้

```
int  x,y,deltax,deltay,i;
float  e;
e=(deltay/deltax)-0.5;
for (i=0;i<deltax;i++)
{   plot (x,y);
    if e>0
```

```
    {  y=y+1;
       e=e-1;}
    x=x+1;
    e=e+(deltay/deltax);
}
```

ข้อเสียของวิธีการนี้ คือ การหาร ซึ่งใช้ในการคำนวณค่าเริ่มต้น และการเพิ่มขึ้นของค่า e อาจให้ค่าที่ผิดไป แต่สามารถที่จะแก้ไขได้ โดยไม่ให้มีผลกระทบ คือ คูณค่า e ด้วยค่าคงที่ (ทดสอบเฉพาะเครื่องหมายของ e) ค่าคงที่ที่นำมาใช้คูณ คือ $2\Delta X$ โปรแกรมที่บรรยายวิธีการนี้คือ

```
int  x,y,deltax,deltay,i,e;
e=2*deltay-deltax;
for (i=0;i<deltax;i++)
{   plot(x,y);
    if (e<0)
    {  y=y+1;
       e=e+(2*deltay-2*deltax);
    }
    else
       e=e+2*deltax;
    x=x+1;
}
```

ชุดคำสั่งนี้จะใช้กับกรณี $0 \leq \Delta Y \leq \Delta X$ เท่านั้น

ประวัติผู้เขียน

นางสาวศุปกิจ จีระมงคลพาณิชย์ เกิดเมื่อวันที่ 26 มกราคม พ.ศ. 2512 ที่กรุงเทพมหานคร สำเร็จการศึกษา วิศวกรรมศาสตรบัณฑิต จากคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ปีการศึกษา 2534 ปัจจุบันอยู่บ้านเลขที่ 111/55 หมู่ 14 แขวงบางหว้า เขตภาษีเจริญ กรุงเทพมหานคร ประกอบวิชาชีพ รับราชการ ที่กองรังวัดและทำแผนที่ กรมที่ดิน