การควบคุมเชิงเลขด้วยคอมพิวเตอร์

COMPUTERIZED NUMERICAL CONTROL

โดย

นาย จักรพรรดิ์	คล่องพยาบาล	34162108
นาย ธนิต	สกุลแก้ว	34162115
นาย มงคล	จันทรชูเกียรติ	34162123
นาย สมชาย	วรรณวิมลศรี	34162128

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

สาขา เทคโนโลยีคอมพิวเตอร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

ปีการศึกษา 2535

ปริญญานิพนธ์ปีการศึกษา 2535

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมเชิงเลขด้วยคอมพิวเตอร์

COMPUTERIZED NUMERICAL CONTROL (SIMULATION CNC)

ผู้จัดทำ

นาย จักรพรรดิ์	คล่องพยาบาล	34162108
นาย ธนิต	สกุลแก้ว	34162115
นาย มงคล	จันทรชูเกียรติ	34162123
นาย สมชาย	วรรณวิมลศรี	34162128

.....อาจารย์ที่ปรึกษา

(อาจารย์ภากร หุตะสังกาศ)

บทนำ

ในปัจจุบันนี้ได้มีปัญหาทรัพยากรถูกทำลายลงไปมาก ดังนั้นจึงได้มีการตื่นตัวในการใช้ทรัพยากรมากยิ่งขึ้นว่าจะทำเช่นใดให้มีการสูญเสียทรัพยากรน้อยที่สุด

ปริญญานิพนธ์นี้ จึงได้มีความคิดที่จะเขียนโปรแกรมเพื่อที่จะทำการจำลองการทำงานของเครื่องพับเหล็ก ซึ่งจะนำไปเป็นประโยชน์ในโรงงานอุตสาหกรรมที่จะต้องมีการทำงานเกี่ยวกับด้านการพับเหล็ก ในการพับเหล็กในโรงงานโดยทั่วไปจะใช้เครื่อง CNC เป็นหลักใหญ่ในการทำงาน

ในการทำงานของเครื่อง CNC นั้นจะต้องมีการใส่แผ่นเหล็กเข้าไปแล้วทำการป้อนโปรแกรมให้แก่เครื่อง CNC เมื่อเป็นเช่นนั้นในการพับเหล็กชิ้นแรกๆ กว่าจะได้ มุม ระยะ ตามที่ต้องการอาจจะต้องทำการแก้ไขโปรแกรมหลายครั้งดังนั้นจึงทำให้เกิดการสูญเสียแก่ชิ้นงาน ซึ่งเป็นการสูญเสียทรัพยากรโดยใช่เหตุ ดังนั้นถ้ามีการจำลองการทำงานของเครื่องก่อนที่จะมีการปฏิบัติกับแผ่นเหล็กจริง โดยทำการป้อน มุม ระยะ ต่างๆ ในการพับงานชิ้นนั้นลงไปยังโปรแกรมในเครื่อง COMPUTER ก่อน เพื่อตรวจสอบดูว่าโปรแกรมที่ทำการป้อนเข้าไปนั้นจะทำให้เครื่องพับออกมาแล้วเป็นดังจุดวัตถุประสงค์ดังต้องการหรือไม่ ถ้าไม่ใช่ก็สามารถทำการแก้ไขโปรแกรมเสียใหม่ แล้วตรวจสอบดู ซึ่งสามารถที่จะทำการแก้ไขโปรแกรมได้จนกว่าตรงตามวัตถุประสงค์ เมื่อโปรแกรมที่เราทำการป้อนที่โปรแกรมจำลองเป็นผลที่ออกมาตรงตามวัตถุประสงค์แล้วก็สามารถที่จะทำการป้อนโปรแกรมนั้นให้แก่เครื่อง CNC ได้เลย ซึ่งโปรแกรมที่เราได้ป้อนให้แก่เครื่อง CNC หลังจากการผ่านขั้นตอนการป้อนค่าต่างๆ ลงโปรแกรมจำลองการทำงานของเครื่องแล้วจะเป็นโปรแกรมที่ถูกต้องตามวัตถุประสงค์แล้ว ดังนั้นเมื่อนำไปปฏิบัติงานจริงจึงไม่มีการสูญเสียแผ่นเหล็กเกิดขึ้นอีก ซึ่งถือได้ว่าเป็นการประหยัดทรัพยากรที่มีค่าต่างๆ ที่จะต้องนำมาผลิตเป็นแผ่นเหล็กได้อีกทางหนึ่ง

บทคัดย่อ

ปัจจุบันการใช้งาน Computer มีการใช้งานกันแพร่หลาย Computer Graphics ก็เป็นสาขาหนึ่งของการใช้งาน การเขียนโปรแกรม Graphics จำเป็นต้องมีความรู้เรื่องเมตริกซ์ ระบบ โคออร์ดิเนท คณิตศาสตร์ สำหรับ Computer 2 มิติ และ 3 มิติ การสร้างภาพ 3 มิติ ซึ่งรายงานฉบับนี้ได้รวบรวมพื้นฐานต่างๆไว้ในตอนต้นของรายงาน ในตอนท้ายเป็นการประยุกต์ ใช้งาน Computer Graphics โดยเขียนโปรแกรมจำลองการทำงานเครื่องพับเหล็ก ควบคุมด้วย Computer ในการจำลองการทำงานนั้นจะต้องมีความรู้เกี่ยวกับการทำงานของเครื่องจักรนั้นๆ

ABSTRACT

The computer is becomes very popular today, and the computer graphics is the branch of work. Graphics programming needs to know about matrics, coordinate system, mathematical elements 2-D and 3-D computer graphics, and building 3-D model. This report is collect the basic of that inthe top, and the bottom is apply of computer graphics by computer numerical control. In this simulation need knowledge about machine of that.

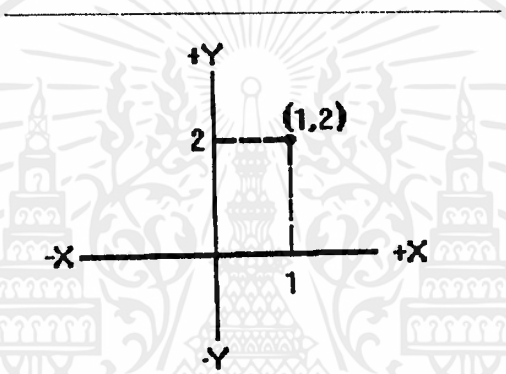
สารบัญ

บทที่ 1

Basic of Computer Graphics

1.1 ระบบโคออร์ดิเนท

ระบบโคออร์ดิเนทที่รู้จักทั่ว ๆ ไปก็คือระบบโคออร์ดิเนท XY ซึ่งใช้คู่ลำดับในการกำหนดจุดต่าง ๆ ในระบบ เช่นจุด (1,2) หมายถึงจุดที่อยู่ห่างจากจุดเริ่มต้นไปทางแกน +X 1 หน่วย และแกน +Y 2 หน่วย ดังรูปแสดงในรูป 1.1

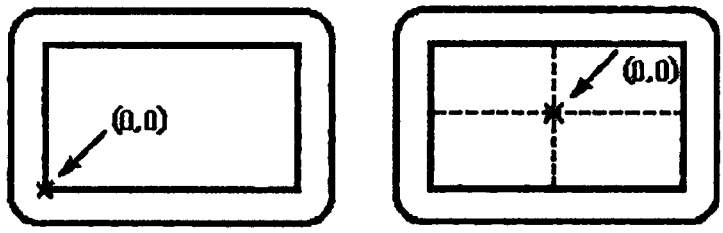


รูป 1.1 แสดงตำแหน่งของจุด (1,2)

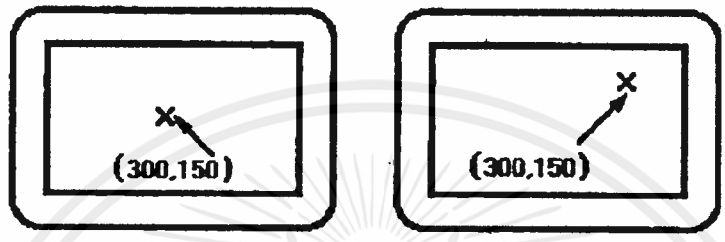
ภาพในระบบกราฟิคจะถูกวาดขึ้นบนจอภาพ ดังนั้นการอ้างถึงจุดหรือตำแหน่งใด ๆ บนจอภาพ เราจึงต้องใช้โคออร์ดิเนทด้วยเช่นกัน แต่เป็นโคออร์ดิเนทของจอภาพแทนที่จะเป็นโคออร์ดิเนท XY อย่างไรก็ตามเรามักจะพบปัญหาที่เกิดขึ้นจากความแตกต่างกันทางด้านฮาร์ดแวร์ของจอภาพ ซึ่งมักจะเกิดจาก

- จุดกำเนิด (0,0) ของโคออร์ดิเนทของจอภาพอยู่คนละตำแหน่ง จอบางชนิดอยู่มุมล่างซ้าย บางชนิดอยู่ตรงกลางจอภาพ ดังแสดงในรูป 1.2

- สเกลที่ใช้ต่างกันอันเนื่องมาจากความละเอียดของจอภาพ ตัวอย่างเช่น จอภาพที่มีจุดเริ่มต้นตรงกลางเหมือนกัน แบบแรกมีความละเอียด 600x300 แบบที่สองมีความละเอียด 1200x1000 ดังนั้นถ้าอ้างถึงจุด (300,150) จอภาพแบบแรกจะได้จุดอยู่บนตำแหน่งมุมบนขวา แต่สำหรับจอภาพแบบที่สองจะได้ตำแหน่งที่ใกล้จุดกึ่งกลางจอภาพเข้ามาอีก (ดูรูป 1.3)

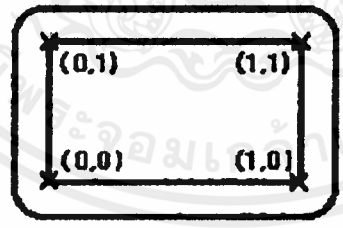


รูป 1.2 แสดงตำแหน่งของจุดเริ่มต้นบนจอภาพบางชนิด



รูป 1.3 แสดงตำแหน่งของจุด (300,150) บนจอภาพที่มีความละเอียดต่างกัน

เพื่อตัดปัญหาความยุ่งยากของความแตกต่างกันของจอภาพคนละชนิด ในระบบคอมพิวเตอร์กราฟิค จึงได้กำหนดโคออร์ดิเนทบนจอภาพเอาไว้ให้ใช้ได้กับจอภาพทุกชนิด เป็นมาตรฐานเดียวกันหมดเรียกว่าโคออร์ดิเนทของอุปกรณ์มาตราฐาน (Normal Device Coordinate) การวางตำแหน่งของจุดต่าง ๆ บนจอภาพไม่ว่าจะมีขนาดใหญ่หรือเล็กหรือมีความละเอียดเท่าใดก็จะมีลักษณะเหมือนกันหมด คือมีความกว้างและสูง 1 หน่วย จุดกำเนิดอยู่ที่มุมล่างซ้าย ดังในรูป 1.4



รูป 1.4 แสดงลักษณะของโคออร์ดิเนทของอุปกรณ์มาตรฐาน

ค่าพารามิเตอร์ของคำสั่งต่าง ๆ ที่เกี่ยวข้องกับโคออร์ดิเนทจะต้องใช้ค่าของโคออร์ดิเนทของอุปกรณ์มาตราฐานเท่านั้น ในลักษณะนี้ทำให้ผู้ใช้ไม่ต้องพะวงว่ากำลังใช้งานจอภาพชนิดใด มีจุดกำเนิดของโคออร์ดิเนทอยู่ที่ใด และมีความละเอียดของจอภาพเท่าไร ขอเพียงให้ทราบว่าโคออร์ดิเนทของอุปกรณ์มาตราฐานเป็นอย่างไรเท่านั้นก็พอ การทำงานของคำสั่งต่าง ๆ จะแปลงให้เป็นตำแหน่งโคออร์ดิเนทที่แท้จริงของจอภาพให้เอง

1.2 เส้นตรง

เส้นต่าง ๆ เป็นส่วนประกอบที่สำคัญและพื้นฐานที่สุดในการวาดภาพ เส้นมีเพียง 2 ชนิด เท่านั้นคือ เส้นตรงและเส้นโค้ง ในที่นี้จะขอกล่าวถึงเส้นตรงเท่านั้นเพราะเข้าใจได้ง่ายในระบบโคออร์ดิเนท เส้นตรงทุกเส้นจะมีสมการเส้นตรงของมัน ดังนั้นการกล่าวอ้างถึงหรือชี้เฉพาะเจาะจงเส้นตรงหนึ่ง ๆ จะใช้สมการเส้นตรงระบุ สมการเส้นตรงมีหลายรูปแบบ เช่น

$$y = mx + b \tag{1.1}$$

โดยที่ m คือ ความชันของเส้นตรง

b คือ จุดตัดแกน Y ของเส้นตรง

หรือมีรูปแบบเป็น

$$\frac{y-y_1}{x-x_1} = \frac{y_2-y_1}{x_2-x_1} \tag{1.2}$$

โดยที่ $(x_1,y_1),(x_2,y_2)$ คือจุด 2 จุดบนเส้นตรง

จากสมการ (1.2)

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

$$y-y_1 = \left[\frac{y_2-y_1}{x_2-x_1}\right](x-x_1)$$

$$y = \left[\frac{y_2-y_1}{x_2-x_1}\right] x - \left[\frac{y_2-y_1}{x_2-x_1}\right] x1_1 + y_1 \qquad (1.3)$$

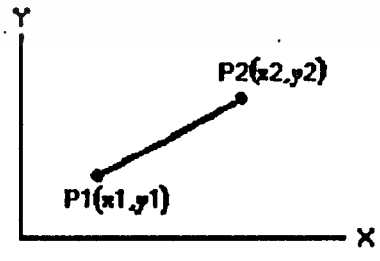
เปรียบเทียบสมการ (1.2) กับ (1.3) จะเห็นว่า ความชัน

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

และ

$$b = \left[\frac{y_2-y_1}{x_2-x_1}\right] x1_1 + y_1 \qquad (1.4)$$

นั้นคือถ้าทราบจุด 2 จุดบนเส้นตรง เราสามารถหาความชันของเส้นตรงนั้น ๆ ได้โดยใช้สมการ (1.4) สมการเส้นตรงนั้นใช้แทนเส้นตรงที่มีความยาวไม่จำกัด แต่การวาดเส้นตรงให้เกิดขึ้นบนจอ เราต้องการเพียง "ส่วนของเส้นตรง" (Line Segment) เท่านั้น การกำหนดส่วนของเส้นตรง กำหนดด้วยตำแหน่งของจุดปลายทั้งสองของส่วนของเส้นตรง ดังตัวอย่างในรูป 1.5 ซึ่งเราจะได้เส้นตรงระหว่างจุด P_1 และ P_2



รูป 1.5 ตัวอย่างการกำหนดส่วนของเส้นตรง

1.3 การสร้างเส้นตรงบนจอภาพ

บนจอภาพแบบราสเตอร์ พิกเซลแต่ละพิกเซลมีตำแหน่งที่แน่นอนตายตัว ซึ่งตำแหน่งของพิกเซลกำหนดได้ด้วยโคออร์ดิเนทของจอภาพ ตัวอย่างเช่น จอภาพโมโนโครมที่ใช้กับเครื่องไมโครคอมพิวเตอร์มีความละเอียด 720x348 พิกเซลตำแหน่งที่ (0,0) คือจุดที่อยู่มุมบนซ้าย พิกเซลตำแหน่งที่ (0,347) คือจุดที่อยู่มุมล่างซ้าย พิกเซลตำแหน่งที่ (719,347) คือพิกเซลที่อยู่ตำแหน่งมุมล่างขวา จะเห็นว่าการที่จะกำหนดหรืออ้างถึงพิกเซลใด ๆ เราต้องกำหนดเป็นโคออร์ดิเนทของพิกเซลนั้น ๆ ให้ถูกต้อง และที่สำคัญคือเป็นโคออร์ดิเนทของเลขจำนวนเต็มเท่านั้น

ในการสร้างเส้นตรงบนจอภาพแบบราสเตอร์ เราจะอาศัยหลักการที่ว่า เส้นตรงเกิดจากจุดเล็ก ๆ เรียงติดต่อกัน ดังนั้นเราจะใช้พิกเซลของจอภาพแทนจุดทำให้เกิดเป็นเส้นตรงขึ้นมาบนจอภาพ การสร้างจุดให้เกิดขึ้นบนจอภาพทำได้โดยเปลี่ยนสี (แอตตริบิ้ว) ของพิกเซล ณ ตำแหน่งที่เราต้องการให้เกิดจุด ซึ่งระบุโดยใช้โคออร์ดิเนทของพิกเซลนั้น สีที่จะเปลี่ยนต้องต่างจากสีพื้นหรือสีแบคกราวนด์ เช่น สีแบรคกราวนด์ของจอภาพเป็นสีขาว การสร้างจุดก็คือการเปลี่ยนสีของพิกเซลเป็นดำ หรือแดงหรือสีอื่น ๆ ที่ไม่ใช่สีขาว

ความจริงแล้วสีแบคกราวนด์ของจอภาพ ก็คือสีของพิกเซลทุกพิกเซลก่อนที่จะมีการทำคำสั่งใด ๆ เพื่อให้เกิดภาพขึ้นบนจอ ซึ่งเป็นสีเดียวกันหมดทั่วทั้งจอภาพ คำสั่งในการเปลี่ยนสีหรือแอตตริบิ้วของพิกเซลบนจอภาพ มักจะมีรูปแบบดังนี้

```
Set-Attribute (X,Y,C)
```

โดยที่ (X,Y) คือโคออร์ดิเนทหรือตำแหน่งของพิกเซลที่ต้องการเปลี่ยนสีของมัน
และ C คือสีที่ต้องการให้พิกเซลนั้นเป็น

ต่อไปจะเป็นการสร้างเส้นตรงบนจอภาพแบบราสเตอร์ โดยอาศัยวิธีการสร้างจุดทีละจุดเพื่อให้เกิดเส้นตรงขึ้น จากสมการ (1.1)

$$y = mx + b \qquad (1.5)$$

ถ้าเรามีจุด $P_1(x_1,y_1)$ และ $P_2(x_2,y_2)$ อยู่บนเส้นตรงนี้ ดังนั้น

$$y_1 = mx_1 + b$$

$$y_2 = mx_2 + b$$

$$y_2-y_1 = m(x_2-x_1) \qquad (1.6)$$

ถ้า $$x_2 = x_1 + 1 \qquad (1.7)$$

แทนค่าสมการ (1.7) ลงในสมการ (1.6) จะได้

$$y_2-y_1 = m \qquad (1.8)$$

จากสมการ (1.6) (1.7) และ (1.8) อธิบายได้ว่า จุดต่าง ๆ ที่อยู่บนเส้นตรงเดียวกัน ถ้า X เพิ่มค่าขึ้น 1 ค่าของ Y จะต้องเพิ่มขึ้น m ในทำนองเดียวกันเราสามารถพิสูจน์ได้ว่า ถ้าเราเพิ่มค่า Y ขึ้น 1 ค่าของ X ต้องเพิ่มขึ้น 1/m จากข้อสรุปที่กล่าวมานี้ทำให้เราสร้างอัลกอริทึม ที่ใช้ในการวาดเส้นตรงบนจอภาพแบบราสเตอร์ได้ ซึ่งต้องการเพียงตำแหน่งของจุดปลายทั้ง 2 ของเส้นตรงคือ (x_1,y_1) และ (x_2,y_2) เท่านั้น การทำงานของอัลกอริทึมเป็นดังนี้

1. คำนวณหาความชัน m
2. ให้ $x = x_1$ และ $y = y_1$
3. เปลี่ยนสีของพิกเซลที่มีโคออร์ดิเนทใกล้เคียงกับจุด (x,y) มากที่สุด
4. เพิ่มค่า x ขึ้น 1 และเพิ่มค่า y ขึ้น m
5. ทำซ้ำข้อ 2 3 4 จนกระทั่ง x มากกว่าหรือเท่ากับ x_2

<u>หมายเหตุ</u> x_1 ต้องน้อยกว่าหรือเท่ากับ x_2

อัลกอริทึมข้างบนนี้จะใช้ได้ดีกับเส้นตรงที่มีความชันน้อยกว่าหรือเท่ากับ 1 (มีมุมเอียงไม่เกิน 45 องศา) ถ้าเส้นตรงมีความชันมากกว่า 1 จะเกิดอลิแอสซิ่งและอาจทำให้เส้นที่ไม่ต่อเนื่อง ดังนั้นในกรณีที่ค่าความชันมากกว่า 1 เราจะแก้ไขวิธีการสร้างเส้นตรงเล็กน้อยดังนี้

1. คำนวณค่าส่วนกลับของ m : $m' = 1/m = (x_2-x_1)/(y_2-y_1)$
2. ให้ $x = x_1$ และ $y = y_1$
3. เปลี่ยนสีของพิกเซลที่มีโคออร์ดิเนทใกล้เคียงกับจุด (x,y) มากที่สุด

4. เพิ่มค่า y ขึ้น 1 และเพิ่มค่า x ขึ้น m'

5. ทำซ้ำข้อ 2 3 4 จนกระทั่ง y มากกว่าหรือเท่ากับ y_2

<u>หมายเหตุ</u> y_1 ต้องน้อยกว่าหรือเท่ากับ y_2

ดังนั้นการสร้างเส้นตรงจากจุด ต้องแยกพิจารณากรณีของความชัน m มากกว่า 1 และน้อยกว่าหรือเท่ากับ 1 เพื่อให้เส้นตรงมีความเงียบมากที่สุด เป็นการลดอลิแอสซิ่ง และทำให้เส้นตรงมีความต่อเนื่องกันตลอด อาศัยอัลกอริทึมที่กล่าวมานี้ เราสามารถสร้างคำสั่งในการลากเส้นซึ่งรับพารามิเตอร์เป็นโคออร์ดิเนทของจุดปลายทั้ง 2 ของเส้นตรง คำสั่งที่ทำหน้าที่ว่าเส้นตรงมักมีรูปแบบดังนี้

$$Line(X_1,Y_1,X_2,Y_2)$$

หมายความว่าเป็นการลากเส้นตรงจากจุด (X_1,Y_1) ไปยังจุด (X_2,Y_2) คำสั่งวาดเส้นตรงที่สร้างจากอัลกอริทึมที่กล่าวมาแล้วนั้น ต้องใช้คำสั่งเปลี่ยนแอตตริบิ้วของพิกเซลในการทำงาน นั่นคือคำสั่ง Set-Attribute ก็เป็นคำสั่งพื้นฐานของการสร้างคำสั่ง Line

1.4 คำสั่งอื่น ๆ ที่เกี่ยวข้อง

ในระบบคอมพิวเตอร์กราฟิคบางระบบ จะมีตัวแปรตัวหนึ่งชื่อ CP (Current Pointer) ตัวแปรนี้คล้าย ๆ กับเคอร์เซอร์ในระบบคอมพิวเตอร์ทั่วไป ทุกครั้งที่มีการพิมพ์ข้อความออกสู่จอภาพตำแหน่งของเคอร์เซอร์จะไปอยู่ในตำแหน่งท้ายสุดของข้อความที่ถูกพิมพ์ออกมา CP ก็เช่นเดียวกันแต่มีอยู่ในระบบการทำงานแบบกราฟิค และ CP นี้จะมองไม่เห็นบนจอภาพ ทุกครั้งที่มีการวาดรูปบนจอภาพ CP จะอยู่ที่ตำแหน่งสุดท้ายของการวาดครั้งนั้น ตัวอย่างเช่น เราสั่งให้วาดเส้นตรงจากจุด (X_1,Y_1) ไปยังจุด (X_2,Y_2) หลังจากทำคำสั่งนี้แล้ว CP จะไปอยู่ที่ตำแหน่ง (X_2,Y_2)

เราอาจจะเปรียบ CP หัวปากกาของพล็อตเตอร์ก็ได้ เราใช้ปากกานี้ในการเขียนเส้นบนกระดาษ การลากเส้นจากจุด (X_1,Y_1) ไปยังจุด (X_2,Y_2) เราต้องยกปากกาไปไว้ที่จุด (X_1,Y_1) และลากเส้นตรงไปยังจุด (X_2,Y_2) เมื่อลากเส้นตรงเสร็จแล้ว ถ้าไม่มีคำสั่งอื่นต้องทำต่อ หัวปากกาก็จะหยุดค้างอยู่ที่ตำแหน่ง (X_2,Y_2) นั่นคือ CP อยู่ที่ตำแหน่ง (X_2,Y_2) ถ้ามีการลากเส้นครั้งต่อไป จึงค่อยเลื่อนหัวปากกา (หรือ CP) ไปยังตำแหน่งที่ต้องการได้

จะสังเกตุได้ว่าการเคลื่อนที่ของหัวปากกาหรือ CP นี้มี 2 แบบคือ การเคลื่อนที่ไปยังตำแหน่งใหม่แล้วเกิดเส้นตรงขึ้นมา (หัวปากกาลากเส้นขณะเคลื่อนที่) ลักษณะเช่นนี้เหมือนกับไปกระทำคำสั่งลากเส้น แต่ต่างกันตรงที่เป็นการลากเส้นจากจุดที่หัวปากกาหรือ CP อยู่ในขณะนั้นไปยังจุดอื่น ลักษณะเช่นนี้เรามักจะมีคำสั่งอีกคำสั่งหนึ่ง คือ

Line-to (x,y) หรือ Line-Abs (x,y)

คำสั่งนี้เป็นการลากเส้นตรงจากจุดที่ CP อยู่ในขณะนั้นไปยังจุด (x,y) ซึ่งหลังจากทำคำสั่งนี้แล้วตำแหน่งของ CP จะย้ายไปอยู่ที่ (x,y) ด้วย (คำสั่ง Line-Abs ย่อมาจาก Line-Absolute)

การเคลื่อนที่หัวปากกาอีกลักษณะหนึ่งคือ เคลื่อนที่ไปยังตำแหน่งอื่นโดยไม่มีเส้นเกิดขึ้น (ยกหัวปากกาไป) ลักษณะนี้เป็นการเปลี่ยนตำแหน่งของ CP เท่านั้น รูปแบบของคำสั่งมักจะเป็น

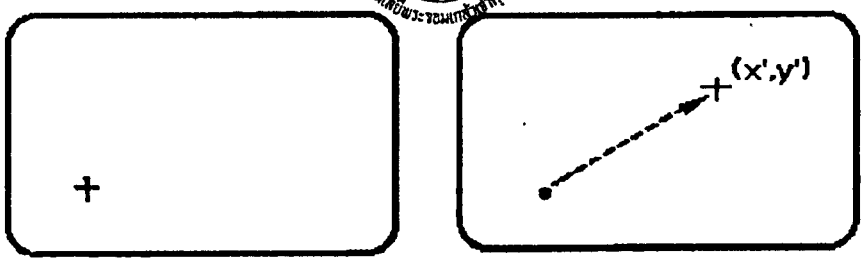
Move-to (x,y) หรือ Move-Abs (x,y)

คำสั่งนี้เป็นการเปลี่ยนตำแหน่งของ CP ให้ไปอยู่ที่จุด (x,y) โดยไม่สนใจว่าก่อนที่ทำคำสั่งนี้ CP อยู่ที่ตำแหน่งใด (คำสั่ง Move-Abs ย่อมาจาก Move-Absolute) ในรูป 1.6 และ 1.7 เป็นตัวอย่างของผลลัพธ์ที่ได้จากการใช้คำสั่ง Line-to และ Move-to เครื่องหมายกากบาทเป็นเครื่องหมายที่แสดงให้เห็นถึงตำแหน่งของ CP เท่านั้น

ก) ก่อนทำคำสั่ง ข) หลังทำคำสั่ง

รูป 1.6 ผลจากการใช้คำสั่ง Line-to (x',y')

ยังมีคำสั่งเกี่ยวกับการเคลื่อนที่ของหัวปากกาหรือ CP อีก 2 คำสั่ง ซึ่งมีลักษณะคล้ายกับคำสั่ง Line-to และ Move-to คำสั่งแรกคือคำสั่ง

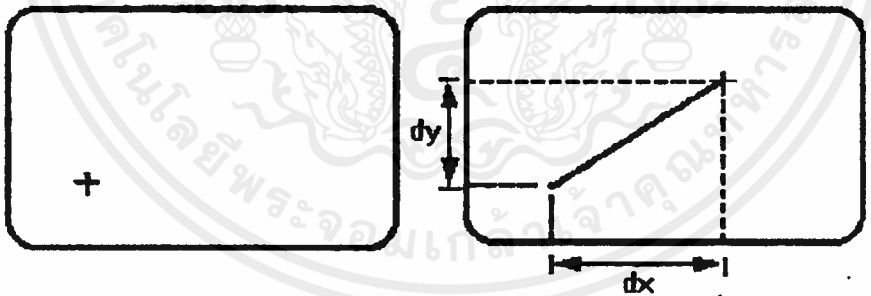


ก) ก่อนทำคำสั่ง ข) หลังทำคำสั่ง

รูป 1.7 ผลจากการใช้คำสั่ง Move-to (x',y')

Line-Rel(dx,dy)

(ย่อมาจาก Line-Relative) คือการสั่งให้ลากเส้นตรงจากจุดที่ CP อยู่ในขณะนั้นไปยังจุดที่ห่างไปทางแกน X เป็นจำนวน dx และห่างไปทางแกน Y เป็นจำนวน dy ทั้ง dx และ dy นี้อาจเป็นได้ทั้งจำนวนบวกหรือจำนวนลบก็ได้ สมมุติว่าเดิม CP อยู่ที่ตำแหน่ง (x_1,y_1) คำสั่ง Line-Rel(dx,dy) จะลากเส้นตรงจากจุด (x_1,y_1) ไปยังจุด (x_1+dx,y_1+dy) จะเห็นว่าจุดสุดท้ายของเส้นตรงที่เกิดขึ้น ขึ้นอยู่กับตำแหน่งของ CP ก่อนทำคำสั่ง Line-Rel ในรูป 1.8 เป็นตัวอย่างการใช้คำสั่ง Line-Rel



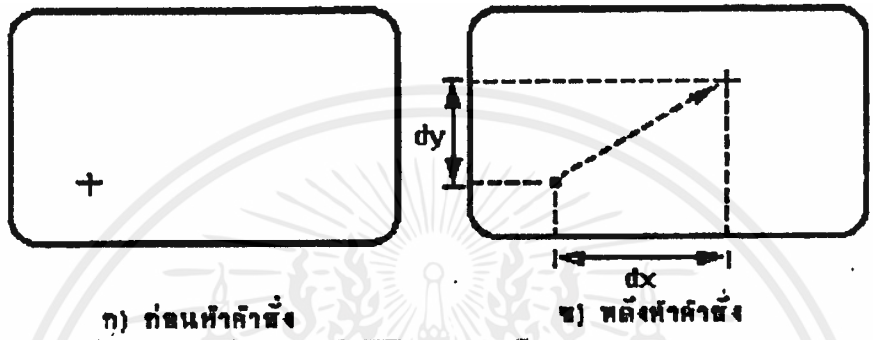
ก) ก่อนทำคำสั่ง ข) หลังทำคำสั่ง

รูป 1.8 ผลจากการใช้คำสั่ง Line-Rel (dx,dy)

ส่วนคำสั่งที่สองคือ คำสั่ง

Move-Rel (dx, dy)

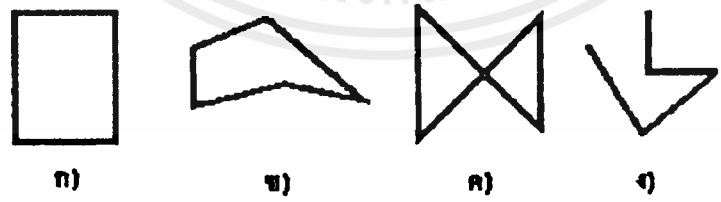
(ย่อมาจาก Move-Relative) ซึ่งเป็นการสั่งให้ย้าย CP จากตำแหน่งที่อยู่ในขณะนั้นไปยังตำแหน่งใหม่ ซึ่งห่างออกไปทางแกน X เป็นจำนวน dx และห่างออกไปทางแกน Y เป็นจำนวน dy และเช่นกันทั้ง dx และ dy เป็นได้ทั้งจำนวนบวกหรือจำนวนลบก็ได้ ตัวอย่างเช่น เดิม CP อยู่ที่ตำแหน่ง (x_1,y_1) คำสั่ง Move-Rel(dx,dy) จะทำให้ CP ย้ายไปยังตำแหน่ง (x_1+dy,y_1+dy) นั้นคือตำแหน่งใหม่ของ CP ขึ้นอยู่กับตำแหน่งเดิมของมันก่อนทำคำสั่ง Move-Rel รูป 1.9 เป็นตัวอย่างของการใช้คำสั่ง Move-Rel



รูป 1.9 ผลจากการใช้คำสั่ง Move-Rel(dx,dy)

1.5 รูปหลายเหลี่ยม

รูปหลายเหลี่ยม (Polygon) เกิดจากการนำเอาส่วนของเส้นตรง 3 เส้นหรือมากกว่ามาต่อกันไปเรื่อย ๆ จนกระทั่งมาบรรจบกับกันเป็นรูปปิด และไม่มีการตัดกันของส่วนของเส้นตรงเหล่านั้น ดังเช่นแสดงไว้ในรูป 1.10 ก) และ ข) ส่วนรูป ค) และ ง) ไม่เป็นรูปหลายเหลี่ยมเพราะมีการตัดกันของเส้น และไม่เป็นรูปปิดตามลำดับ



รูป 1.10 ตัวอย่างของรูปแบบต่าง ๆ

เราสามารถแบ่งประเภทของรูปหลายเหลี่ยมได้เป็น 2 ประเภทคือ **รูปหลายเหลี่ยมเว้าเข้า** (Concape Polygon) และ**รูปหลายเหลี่ยมเว้าออก** (Convex Polygon) รูปหลาย

เหลี่ยมใด ๆ ที่สามารถลากเส้นตรงภายในรูปหลายเหลี่ยมเชื่อมจุด 2 จุดบนขอบของเส้นรอบรูปนั้น แล้วไม่มีส่วนหนึ่งส่วนใดของเส้นตรงอยู่ภายนอกขอบของรูปหลายเหลี่ยม เราเรียกรูปหลายเหลี่ยมประเภทนี้ว่า รูปหลายเหลี่ยมเว้าออก

จากคำนิยามนี้จะเห็นว่า รูปสามเหลี่ยมเป็นรูปหลายเหลี่ยมเว้าออกเสมอ ในรูป 1.11 ก) เป็นตัวอย่างของรูปหลายเหลี่ยมเว้าออก และรูป 1.11 ข) เป็นตัวอย่างของรูปหลายเหลี่ยมเว้าเข้า

ก) รูปหลายเหลี่ยมเว้าออก

ข) รูปหลายเหลี่ยมเว้าเข้า

รูป 1.11 ตัวอย่างของรูปหลายเหลี่ยม

เนื่องจากรูปหลายเหลี่ยมเกิดจาก การนำเอาส่วนของเส้นตรงมาต่อกันจนเกิดเป็นรูปปิดขึ้น การสร้างคำสั่งสร้างรูปหลายเหลี่ยมจึงเป็นสิ่งที่ไม่ยากเลย เพียงแค่กำหนดจุดมุมต่าง ๆ ของรูปหลายเหลี่ยมที่เราต้องการสร้าง จากนั้นลากเส้นเชื่อมโยงจากจุดสองจุดที่เป็นจุดมุมที่ติดกันไปเรื่อย ๆ จนครบทุกจุด ก็จะได้รูปหลายเหลี่ยมขึ้นมา คำสั่งในการสร้างรูปหลายเหลี่ยมมักจะมีรูปแบบดังนี้

```
Polygon (AX,AY,N)
```

โดยที่ N คือจำนวนของจุดมุมของรูปหลายเหลี่ยม AX และ AY คือตัวแปรชนิดที่เป็นอาเรย์ (array) ขนาด 1 x N มิติ ซึ่งเก็บค่าโคออร์ดิเนท X และ Y ของจุดมุม N จุดของรูปหลายเหลี่ยม คำสั่ง Polygon จะเริ่มลากเส้นตรงจากจุดแรกไปยังจุดที่สอง และจากจุดที่สองไปยังจุดที่สาม และต่อ ๆ ไป จนกระทั่งถึงจุดที่ N จากนั้นลากเส้นตรงจากจุดที่ N กลับไปยังจุดแรก

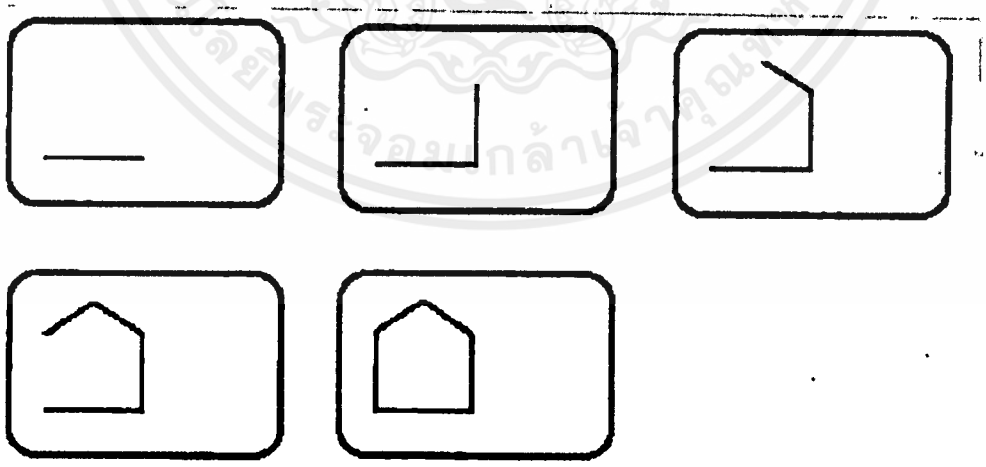
ตัวอย่างเช่นเราต้องการสร้างรูป 5 เหลี่ยมรูปหนึ่ง ค่าที่เก็บไว้ในตัวแปร AX และ AY มีดังนี้

AX	AY	
0.1	0.1	จุดที่ 1 (0.1,0.1)
0.5	0.1	จุดที่ 2 (0.5,0.1)
0.5	0.5	จุดที่ 3 (0.5,0.5)
0.3	0.8	จุดที่ 4 (0.3,0.8)
0.1	0.5	จุดที่ 5 (0.1,0.5)

เราใช้คำสั่ง Polygon (Ax,Ay,5) เพื่อสร้างรูป 5 เหลี่ยมขึ้น การทำงานของคำสั่ง Polygon จะเริ่มลากเส้นตรงจากจุดที่ 1 (0.1,0.1) ไปยังจุดที่ 2 (0.5,0.1) ลากต่อไปยังจุดที่ 3 (0.5,0.5) ลากต่อไปยังจุดที่ 4 (0.3,0.8) ลากต่อไปยังจุดที่ 5 (0.1,0.5) และลากเส้นตรงกลับไปหาจุดที่ 1 (0.1,0.1) อีกครั้ง ดังแสดงในรูป 1.12 เราสามารถแทนการทำงานของคำสั่ง Polygon (Ax,Ay,5) ด้วยคำสั่งที่กล่าวมาในหัวข้อที่แล้วได้ดังนี้

```
Move-to (0.1,0.1)          /* เลื่อน CP ไปยังจุดแรก */
Line-to (0.5,0.1)
Line-to (0.5,0.5)
Line-to (0.3,0.8)
Line-to (0.1,0.5)
Line-to (0.1,0.1)
```

เราสร้างคำสั่ง Polygon นี้ได้จากคำสั่ง Move-to และ Line-to



รูป 1.12 ตัวอย่างขั้นตอนการวาดรูป 5 เหลี่ยม

บทที่ 2

การแปลง (Transformation)

สิ่งที่ทำให้คอมพิวเตอร์กราฟิค มีประโยชน์อย่างมากในการสร้างภาพก็คือ ความง่ายดาย สะดวกสบายในการเปลี่ยนแปลงภาพที่เราวาดขึ้นมาแล้ว เช่น เปลี่ยนสเกลของกราฟเปลี่ยนมุมมองของแบบติกที่ออกแบบไว้ หรือการเปลี่ยนขนาดภาพของแผนที่ เป็นต้น ทั้งหมดนี้สามารถทำได้ง่าย รวดเร็ว โดยใช้คอมพิวเตอร์กราฟิค เพราะข้อมูลของรูปภาพต่าง ๆ หรือข้อมูลทางกราฟิคได้ถูกป้อนเก็บไว้ในคอมพิวเตอร์ เพียงแค่เปลี่ยนแปลงวิธีที่จะแปลงข้อมูลเหล่านี้ออกมาเป็นภาพเท่านั้น เราก็จะได้ภาพในลักษณะที่ต้องการ การเปลี่ยนแปลงนี้อาศัยการคำนวณทางคณิตศาสตร์โดยเปลี่ยนค่าพารามิเตอร์ต่าง ๆ ที่เกี่ยวข้องกับการสร้างภาพเท่านั้น เราเรียกวิธีการนี้ว่า **การแปลง** (Transformation)

2.1 เมตริกซ์

โดยพื้นฐานแล้วภาพของระบบคอมพิวเตอร์กราฟิค ถูกสร้างขึ้นด้วยส่วนของเส้นตรงหรือเวคเตอร์หลาย ๆ เวคเตอร์มาประกอบกัน ซึ่งเราสามารถกำหนดเส้นตรงหรือเวคเตอร์เหล่านี้ได้ด้วยจุดปลายทั้งสองของมัน การที่จะเปลี่ยนแปลงลักษณะภาพที่วาด จำเป็นจะต้องใช้การคำนวณทางคณิตศาสตร์กระทำกับจุดต่าง ๆ เหล่านี้ วิธีหนึ่งที่จะช่วยให้เข้าใจได้ง่ายก็คือ การใช้เมตริกซ์เข้าช่วย แต่สำหรับผู้ที่ไม่ถนัดการใช้เมตริกซ์ก็อาจจะจำสูตรไปใช้งานได้เช่นกัน

เราใช้เมตริกซ์ 2 มิติเพื่อช่วยทำการแปลง โดยแทนจุดต่าง ๆ ด้วยเมตริกซ์ขนาด 1x2 เช่นจุด (0.5,1) แทนด้วยเมตริกซ์ [0.5 1] และจุด (-5,0.3) แทนด้วย [-5 0.3] เป็นต้น เมื่อต้องทำการแปลงใด ๆ ก็เพียงแต่หาเมตริกซ์ที่เหมาะสมมาคูณกับเมตริกซ์ของจุด ผลลัพธ์ที่ได้คือเมตริกซ์ใหม่ ซึ่งก็คือจุดหรือตำแหน่งใหม่ของจุดปลายของเส้นตรงต่าง ๆ ทำให้เส้นหรือเวคเตอร์เหล่านี้มีลักษณะที่เปลี่ยนไป เมตริกซ์ที่นำมาคูณเพื่อเปลี่ยนตำแหน่งของจุดนี้เราเรียกว่า **เมตริกซ์การแปลง** (Transformation matrix)

2.3 การแปลงแบบสเกล

สมมติว่าเรามีจุด $P_1 = [x_1 \quad y_1]$ ซึ่งเป็นเมตริกซ์ขนาด 1x2 ถ้าเราคูณเมตริกซ์นี้ด้วยเมตริกซ์ T ที่มีขนาด 2x2 เราจะได้เมตริกซ์ขนาด 1x2 เมตริกซ์ใหม่ (P_2) กลับว่าโดยที่

$$P_2 = [x_2 \quad y_2] = P_1T$$

การคูณเมตริกซ์นี้จะเป็นการคูณทางซ้าย และ T เป็นเมตริกซ์การแปลงที่จะแปลงจุด P_1 ไปเป็นจุด P_2 ถ้าเราน่าเมตริกซ์การแปลง T นี้คูณเข้ากับจุดทุก ๆ จุด แล้วอะไรจะเกิดขึ้น คำตอบก็คือเราจะได้ภาพที่มีลักษณะเปลี่ยนแปลงไป แต่จะเปลี่ยนอย่างไรนั้นขึ้นอยู่กับค่าต่าง ๆ ในเมตริกซ์ T เช่นถ้าเราคูณด้วย เมตริกซ์เอกลักษณ์ (Identity matrix)

$$T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad (2.1)$$

$$P_2 = P_1T = [x_1 \quad y_1] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$= [x_1 \quad y_1] = P_1 \qquad (2.2)$$

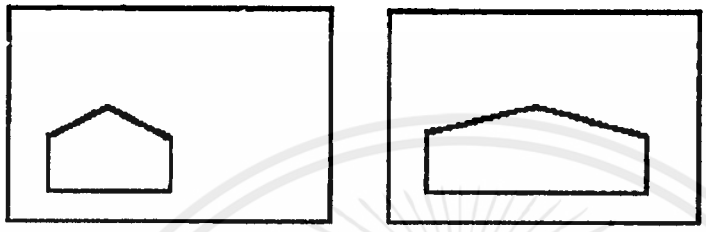
จุด P_2 กับ P_1 จะเป็นจุดเดียวกัน คือไม่มีการเปลี่ยนแปลงใด ๆ แต่ถ้าเราเลือกเมตริกซ์ T_1 เป็น

$$T_1 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \qquad (2.3)$$

$$P_2 = P_1T_1 = [x_1 \quad y_1] \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

$$= [2x_1 \quad y_1] \qquad (2.4)$$

ผลลัพธ์ที่ได้คือโคออร์ดิเนท X ของทุก ๆ จุด จะมีค่ามากขึ้นเป็น 2 เท่าของค่าเดิม เส้นตรงในแนวนอนก็จะมีความยาวเป็น 2 เท่าของภาพเดิม ภาพใหม่ที่เปลี่ยนไปจะมีความสูงเท่าเดิม แต่ขยายออกทางแนวนอนออกจากจุดกำเนิด (0,0) เป็น 2 เท่าจากของเดิม ดังตัวอย่างในรูป 2.1



รูป 2.1 แสดงผลของการแปลงแบบสเกลโดยขยายออกทางแนวนอนเป็น 2 เท่า

ในทำนองเดียวกัน ถ้าเมตริกซ์การแปลง T_2 มีค่าเป็น

$$T_2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 1 \end{bmatrix} \qquad (2.5)$$

โคออร์ดิเนท X ของทุก ๆ จุด ก็จะลดลงเป็นครึ่งหนึ่งของค่าเดิม ภาพใหม่ที่ได้จะมีความสูงเท่าเดิม แต่จะถูกบีบให้แคบทางแนวนอนเป็นครึ่งหนึ่ง

คราวนี้เราลองขยายภาพออกทางแนวนอนเป็น 2 เท่า แล้วบีบภาพใหม่นี้ให้แคบลงเป็นครึ่งหนึ่ง แน่นอนเราต้องได้ภาพที่มีขนาดเท่าเดิมกลับมา

$$P_1 = (P_1T_1)T_2 = P_1(T_1T_2) \qquad (2.6)$$

เราสามารถตรวจสอบคำตอบของเราได้โดย นำเมตริกซ์ T_1 และ T_2 มาคูณกัน

$$T_1T_2 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 1 \\ 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} (2*0.5)+(0*0) & (2*0)+(0*1) \\ (0*0.5)+(1*0) & (0*0)+(1*1) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad (2.7)$$

ผลลัพธ์ที่ได้คือเมตริกซ์เอกลักษณ์ นั่นคือเราจะได้ภาพเดิมก่อนเกิดการแปลง

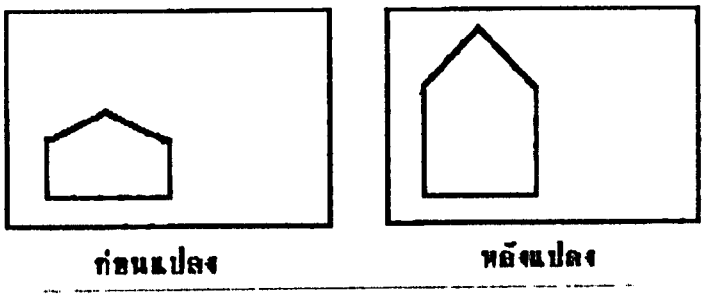
เราสามารถที่จะขยายภาพให้สูงขึ้น หรือกดภาพให้เตี้ยลงในแนวตั้งได้เช่นกัน โดยการหาเมตริกซ์การแปลงที่จะทำให้ค่าโคออร์ดิเนท Y เปลี่ยนไปโดยที่ค่า X ไม่เปลี่ยน ตัวอย่างเช่น เมตริกซ์การแปลง T_3

$$T_3 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \qquad (2.8)$$

$$P_2 = P_1T_3 = [x_1 \quad y_1] \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

$$= [x_1 \quad 2y_1] \qquad (2.9)$$

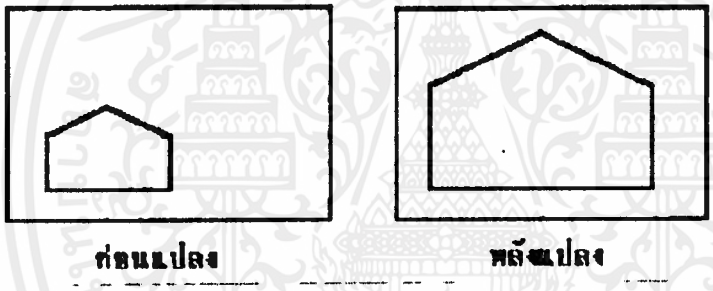
จะได้ภาพที่มีความสูงเพิ่มขึ้นจากเดิม 2 เท่า ดังในภาพ 2.2



รูป 2.2 แสดงผลของการแปลงแบบสเกลโดยเพิ่มความสูงขึ้นเป็น 2 เท่า

ถ้าเรานำเมตริกซ์ T_1 และ T_3 มาคูณกับ P_1 เราก็จะได้ภาพที่มีการขยายออกทั้งแนวนอนและแนวตั้งเป็น 2 เท่าของภาพเดิม หรือกล่าวได้ว่า จะได้ภาพที่มีความโตเป็น 2 เท่า ดังในรูป 2.3

$$P_2 = P_1T_1T_2 = P_1(T_1T_3) \qquad (2.10)$$



รูป 2.3 การแปลงภาพแบบสเกลเพื่อให้ได้ขนาดโตขึ้นเป็น 2 เท่าของภาพเดิม

เรานำเมตริกซ์ T_1 และ T_3 มาคูณกันเพื่อให้ได้เมตริกซ์ใหม่ T_4 เมตริกซ์เดียว

$$T_4 = T_1T_3 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

$$T_4 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \qquad (2.11)$$

เราอาจเขียนเมตริกซ์การแปลงแบบนี้ในรูปทั่ว ๆ ไป คือ

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \qquad (2.12)$$

S_x เรียกว่า **สเกลแฟคเตอร์** (Scale Factor) สำหรับโคออร์ดิเนท X ซึ่งจะมีผลต่อขนาดของภาพในแนวนอนหรือแนวแกน X S_y เรียกว่า **สเกลแฟคเตอร์สำหรับโคออร์ดิเนท Y** ซึ่งจะมีผลต่อขนาดของภาพในแนวตั้งหรือแนวแกน Y เมตริกซ์ S เป็นเมตริกซ์การแปลงซึ่งจะมีผลต่อขนาดและสัดส่วนของภาพ เราเรียกว่า **เมตริกซ์การแปลงแบบสเกล** (Scaling Transformation matrix) การแปลงโดยการคูณด้วยเมตริกซ์ S เราเรียกว่าเป็น **การแปลงแบบสเกล** (Scaling Transformation)

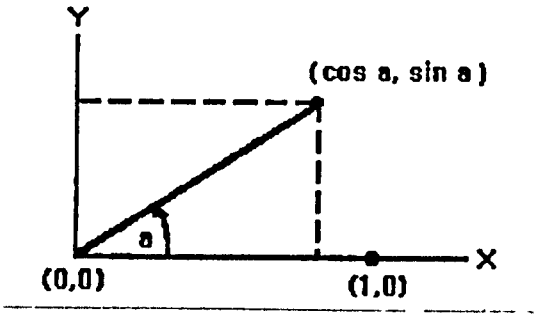
สังเกตว่าการสเกลภาพจะทำให้ทุก ๆ จุดมีการเปลี่ยนแปลง ยกเว้นเพียงจุดเดียวคือ จุดกำเนิด (0,0) ดังนั้นนอกจากขนาดของภาพจะเปลี่ยนไปแล้ว ตำแหน่งของจุดต่าง ๆ ก็เปลี่ยนไปด้วย ถ้า S_x มีค่ามากกว่า 1 ก็จะทำให้ภาพเลื่อนไปทางขวา (สำหรับภาพที่อยู่ทางขวาของแกน Y) และมีความกว้างมากขึ้นด้วย ถ้า S_x มีค่าน้อยกว่า 1 ภาพก็จะเลื่อนไปทางซ้ายและมีขนาดแคบลง ในทำนองเดียวกันถ้า S_y มีค่ามากกว่า 1 ก็จะทำให้ภาพขยายห่างออกจากแกน X และมีความสูงเพิ่มขึ้น และถ้า S_y มีค่าน้อยกว่า 1 ก็จะทำให้ภาพหดเข้าหาแกน X และเตี้ยลง

2.3 การแปลงแบบหมุน

การแปลงแบบหมุน (Rotation Transformation) เป็นวิธีการเปลี่ยนภาพโดยการหมุนจุด (หรือภาพ) ในทิศทางทวนเข็มหรือตามเข็มนาฬิกา โดยมีจุดศูนย์กลางของการหมุนอยู่ที่จุดกำเนิด สิ่งที่เราต้องการทราบก็คือ เมตริกซ์ซึ่งมาคูณจุดที่เราต้องการหมุนเพื่อไปอยู่ที่ตำแหน่งใหม่ เราเรียกเมตริกซ์นี้ว่า **เมตริกซ์การแปลงแบบหมุน** (Rotation Transformation Matrix)

สมมติว่าเราทำการหมุนจุด (1,0) ไปในทิศทางบวก (ทวนเข็มนาฬิกา) เป็นมุม a องศา ตำแหน่งใหม่จะเป็นจุด (cos(a),sin(a)) (ดูรูป 2.4) ถ้าเมตริกซ์การแปลงแบบหมุนคือ

$$S = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad (2.13)$$



รูป 2.4 การหมุนทวนเข็ม

ดังนั้นจะได้ว่า

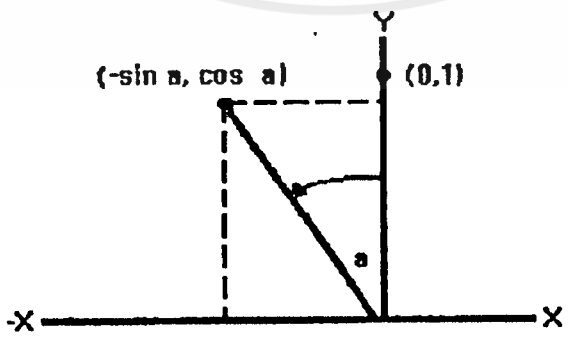
$$[\cos(a) \quad \sin(a)] = [1 \quad 0] \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$= [a \quad b] \qquad (2.14)$$

ถ้าเราหมุนจุด (0,1) ในทิศทางบวกเป็นมุม a เช่นกัน นั้นคือใช้เมตริกซ์การแปลงตัวเดิม ตำแหน่งใหม่จะเป็น (-sin(a),cos(a)) (ดูรูป 2.5) ดังนั้น

$$[-\sin(a) \quad \cos(a)] = [0 \quad 1] \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$= [c \quad d] \qquad (2.15)$$



รูป 2.5 การหมุนทวนเข็ม

จากสมการ (2.13), (2.14) จะได้ว่า

$$a = \cos(a)$$
$$b = \sin(a)$$
$$c = -\sin(a)$$
$$d = \cos(a) \qquad (2.16)$$

ดังนั้นเมตริกซ์การแปลงการหมุน (ทวนเข็ม) คือ

$$R = \begin{bmatrix} \cos(a) & \sin(a) \\ -\sin(a) & \cos(a) \end{bmatrix} \qquad (2.17)$$

ในสมการ (2.17) เป็นเมตริกซ์การแปลงที่ใช้กับการหมุนในทิศทางทวนเข็มนาฬิกา สำหรับการหมุนในทิศทางตามเข็มนาฬิกาคือ

$$R = \begin{bmatrix} \cos(a) & -\sin(a) \\ \sin(a) & \cos(a) \end{bmatrix} \qquad (2.18)$$

ตัวอย่างเช่น ต้องการหมุนจุด $P_1(3,2)$ ในทิศทางทวนเข็มเป็นมุม 30 องศา เมตริกซ์การแปลงจะเป็น

$$\begin{bmatrix} \cos(30) & \sin(30) \\ -\sin(30) & \cos(30) \end{bmatrix} = \begin{bmatrix} 0.866 & 0.5 \\ -0.5 & 0.866 \end{bmatrix}$$

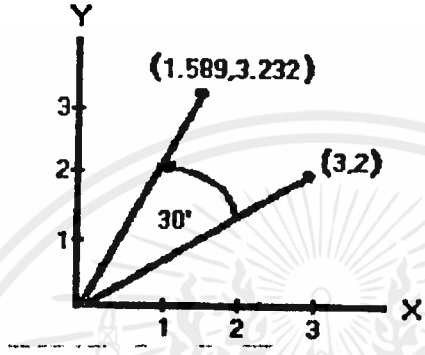
และจุดใหม่หลังการหมุน P_2 คือ

$$P_2 = [3 \quad 2] \begin{bmatrix} 0.866 & 0.5 \\ -0.5 & 0.866 \end{bmatrix}$$

$$= [1.598 \quad 3.232]$$

2.4 การแปลงแบบย้ายและโฮโมจีเนียสโคออร์ดิเนท

การย้าย (Translation) เป็นการเลื่อนตำแหน่งของภาพทั้งภาพไปยังตำแหน่งอื่น ๆ เป็นระยะทางเท่ากันทั้งหมด โดยขนาดของภาพไม่เปลี่ยนแปลงและไม่ทำให้ภาพเอียงไปจากแนวเดิม การย้ายนี้ทำได้โดยการบวกจุดทุก ๆ จุดของภาพด้วยระยะทางที่ต้องการให้ภาพเลื่อนไป ดังนั้น



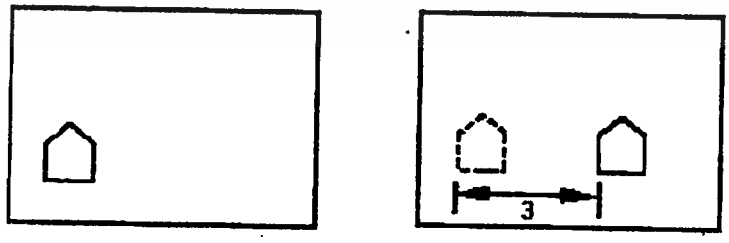
รูป 2.6 การหมุนจุด (2,3) ในทิศทางทวนเข็มเป็นมุม 30 องศา

$$x_2 = x_1 + t_x$$

และ

$$y_2 = y_1 + t_y \qquad (2.19)$$

นั่นคือการย้ายตำแหน่งของภาพไปในแนวแกน X เป็นระยะ t_x และแนวแกน Y เป็นระยะทาง t_y หน่วย โดยที่ t_x และ t_y เป็นระยะทางที่ต้องการให้ภาพเลื่อนไปในแนวแกน X และแกน Y ตามลำดับ ตัวอย่างเช่น ต้องการย้ายภาพให้เลื่อนไปอยู่ทางขวา 3 หน่วย เราก็บวก 3 เข้ากับค่า X ของทุก ๆ จุดของภาพ ดังในภาพ 2.7



ก) ก่อนย้าย ข) หลังย้าย

รูป 2.7 การย้ายไปทางขวา

ในโฮโมจีเนียสโคออร์ดิเนท เมตริกซ์การแปลงแบบสเกลจะถูกเปลี่ยนจากเดิม

$$\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

เป็น

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (2.20)$$

เราลองทดสอบโดยนำเมตริกซ์การแปลงของสมการ (2.20) มาคูณกับจุด $P_1(x,y)$ ซึ่งต้องแทนด้วยเมตริกซ์ [xw yw w]

$$P_2 = P_1 S$$

$$= [xw \quad yw \quad w] \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P_2 = [S_x xw \quad S_y yw \quad w] \qquad (2.21)$$

เมื่อนำโคออร์ดิเนท W มาหารสองโคออร์ดิเนทแรก จะได้จุด P_2 เป็น $(S_x x, S_y y)$ ซึ่งกับการแปลงแบบสเกลที่กล่าวมาแล้ว

เมตริกซ์การแปลงแบบหมุน ในทิศทางทวนเข็มนาฬิกาจากเดิม

$$\begin{bmatrix} \cos(a) & \sin(a) \\ -\sin(a) & \cos(a) \end{bmatrix}$$

จะเปลี่ยนเป็น

$$R = \begin{bmatrix} \cos(a) & \sin(a) & 0 \\ -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (2.22)$$

นำเมตริกซ์ในสมการ (2.22) คูณกับจุด $P_1(xw,yw,w)$ เราจะได้

$$P_2 = [xw \quad yw \quad w] \begin{bmatrix} \cos(a) & \sin(a) & 0 \\ -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (2.22)$$

$$= [xw\cos(a)-yw\sin(a) \quad xw\cos(a)+yw\cos(a) \quad w] \qquad (2.23)$$

ซึ่งจะได้จุด P_2 ที่ได้จากการหมุนจุด P_1 คือ $(x\cos(a)-y\sin(a),\ x\sin(a)+y\cos(a))$ ตรงกับการแปลงแบบหมุนที่กล่าวมาแล้ว

สำหรับการแปลงแบบย้าย เมื่อต้องการเลื่อนภาพหรือจุดไปทางแนวนอน t_x และไปทางแนวตั้ง t_y จะได้เมตริกซ์การแปลงแบบย้ายคือ

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \qquad (2.24)$$

เราสามารถพิสูจน์ได้ว่าเมตริกซ์ในสมการ (2.24) ใช้งานได้โดยนำไปคูณกับจุด $P_1(xw,yw,w)$

$$P_2 = P_1T$$

$$= [xw \quad yw \quad w] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

$$P_2 = [xw+t_x w \quad yw+t_y w \quad w] \qquad (2.25)$$

เราจะได้จุด P_2 ในโฮโมจีเนียสโคออร์ดิเนทเป็น P_2 $(xw+t_x w, yw+t_y w, w)$ ดังนั้นจุด P_2 ก็คือ $(x+t_x, y+t_y)$

2.5 การแปลงโคออร์ดิเนท

ที่กล่าวมาแล้วเราใช้การแปลงกับจุดต่าง ๆ แล้วได้ผลลัพธ์เป็นจุดใหม่ แต่เราอาจใช้การแปลงกับการแปลงระบบโคออร์ดิเนทได้ด้วยเช่นกัน ตัวอย่างเช่น ระยะทางที่ใช้วัดเดิมมีหน่วยเป็นนิ้ว ก็สามารถแปลงเป็นหน่วยเซนติเมตร โดยมีระยะทางเท่าเดิมได้ด้วยการแปลงแบบสเกล การแปลงยังคงทำได้เหมือนเดิม เพียงแต่มีความหมายเปลี่ยนไปเท่านั้น นั่นคือจุด ๆ หนึ่งในโคออร์ดิเนทหนึ่ง เมื่อถูกคูณด้วยเมตริกซ์การแปลงก็ยังคงหมายถึงจุด ๆ เดิมที่อยู่ (หรือถูกวัด) ในโคออร์ดิเนทอื่น

การแปลงแบบย้ายก็สามารถใช้ในการแปลงโคออร์ดิเนทได้เช่นกัน คือในกรณีที่จุดกำเนิดไม่ตรงกัน ตัวอย่างเช่น เราคิดว่ามุมล่างซ้ายของจอภาพมีโคออร์ดิเนท (0,0) แต่สำหรับจอภาพบางรุ่น ตำแหน่งนี้อาจหมายถึงพิกเซลที่มีโคออร์ดิเนท (100,319) ก็ได้ หรือในจอภาพบางแบบจะกำหนดจุด (0,0) ไว้ที่มุมบนซ้ายของจอภาพและค่าโคออร์ดิเนท Y ของแต่ละพิกเซลที่อยู่เหนือกว่า ที่เป็นเช่นนี้เพราะสาเหตุจากการสแกนของจอภาพที่เริ่มสแกนจากบนลงข้างล่าง และในการทำงานของเครื่องพิมพ์ปริ้นเตอร์ก็เช่นกัน เริ่มพิมพ์จากส่วนบนของจอลงมา วิธีที่จะแปลงระบบโคออร์ดิเนทนี้ กับระบบโคออร์ดิเนทที่เราใช้กันธรรมดาทำได้โดยการสเกลค่า X ด้วย 1 และค่า Y ด้วย -1 เพื่อเปลี่ยนลำดับของเส้นสแกน และทำการแปลงแบบย้ายเฉพาะโคออร์ดิเนท Y ด้วยขนาดจอภาพในแนวตั้ง เพื่อให้จุดกำเนิดมาอยู่ในตำแหน่งที่ถูกต้อง

การแปลงแบบหมุนก็อาจใช้สำหรับการแปลงโคออร์ดิเนทได้เช่นกัน แต่ส่วนมากแล้วจะเป็นการหมุนเป็นมุม 90 องศา ตัวอย่างของการแปลงแบบหมุน เพื่อเปลี่ยนโคออร์ดิเนท เช่น การใช้งานปริ้นเตอร์แบบดอตเมตริกซ์ ซึ่งใช้กระดาษขนาด 8.5x11 นิ้ว จะมีแกน Y ไปตามแนวยาวของกระดาษ และมีแกน X ตามแนวขวางของกระดาษ กรณีที่ปริ้นเตอร์ทำงานในลักษณะนี้เราเรียกว่าเป็นการทำงานในโหมด **พรอตเทรต** (Portait Mode) ในบางครั้งเราอาจต้องการจัดให้แกน Y อยู่ในแนวขวางของกระดาษ และแกน X อยู่ตามแนวยาวของกระดาษ ในกรณีนี้เป็นการทำงานในโหมดที่เรียกว่าโหมด **แลนด์สเคป** (Landscape Mode) ซึ่งการใช้งานเหมาะกับภาพที่มีความยาวตามแนวนอนยาวกว่าแนวตั้ง เราใช้การแปลงแบบหมุนด้วยมุม 90 และใช้การ

แปลงแบบย้าย เพื่อเปลี่ยนจุดกำเนิดให้เหมาะสม เพื่อให้ได้ภาพตามโหมดที่เราต้องการ (โหมดพรอตเทรตหรือแลนด์สเคป)

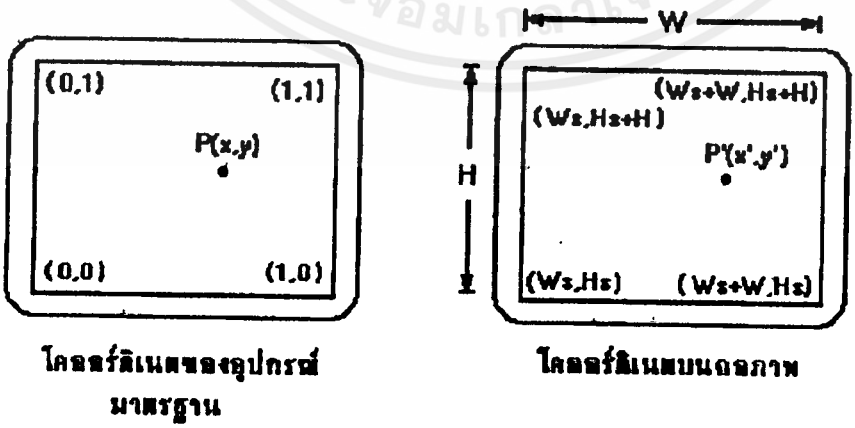
ตัวอย่างการแปลงโคออร์ดิเนทที่กล่าวมา ยังไม่ได้ยกตัวอย่างที่ใช้เมตริกซ์การแปลง ตัวอย่างต่อไปนี้ เป็นการแปลงจากโคออร์ดิเนทของอุปกรณ์มาตรฐาน ไปยังโคออร์ดิเนทของจอภาพจริง ๆ โดยใช้เมตริกซ์การแปลง จุด $P(x,y)$ บนโคออร์ดิเนทของอุปกรณ์มาตรฐานจะถูกเปลี่ยนเป็นจุด $P'(x',y')$ บนจอภาพ ซึ่งมีขนาดความกว้างตามแนวนอน W และสูง H โคออร์ดิเนทเริ่มต้นที่มุมล่างซ้ายเป็น (W_s,H_s) ดังนั้นจะได้ว่า

$$x' = xW + W_s \qquad (2.26)$$

$$y' = yH + H_s \qquad (2.27)$$

หมายความว่า เราทำการสเกลโคออร์ดิเนท X ด้วยความกว้างของจอภาพ และสเกลโคออร์ดิเนท Y ด้วยความสูงของจอภาพ และทำการย้ายด้วยจุดเริ่มต้นของจอภาพ (ดูรูป 2.9) จากสมการ (2.26) และสมการ (2.27) เราจะได้เมตริกซ์การแปลงโคออร์ดิเนทของจอภาพเป็น

$$D = \begin{bmatrix} W & 0 & 0 \\ 0 & H & 0 \\ W_s & H_s & 1 \end{bmatrix} \qquad (2.28)$$



รูป 2.9 การแปลงโคออร์ดิเนทของจอภาพ

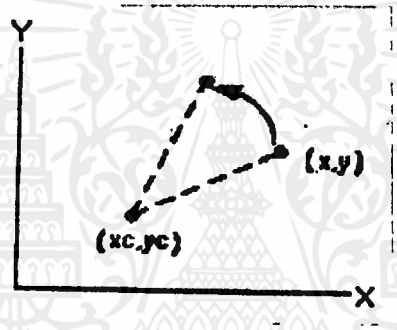
2.6 การหมุนรอบจุดใด ๆ

การแปลงแบบหมุนที่กล่าวมาเป็นการหมุนรอบจุดกำเนิด (0,0) เท่านั้น แต่ในหัวข้อนี้เราจะหาเมตริกซ์การแปลงแบบหมุนที่ใช้หมุนรอบจุดใด ๆ (x_c,y_c) (ดูรูป 2.10)

วิธีที่จะทำการหมุนแบบนี้มีขั้นตอนอยู่ 3 ขั้นตามลำดับ คือ

1. ทำการย้ายภาพเพื่อให้จุดศูนย์กลางของการหมุน (x_c,y_c) ไปอยู่ที่จุดกำเนิด
2. ทำการหมุนรอบจุดกำเนิด
3. ย้ายภาพเพื่อให้จุดศูนย์กลางของการหมุนกลับไปอยู่ในตำแหน่งเดิม

ในรูป 2.11 แสดงขั้นตอนการหมุนรอบจุด (x_c,y_c)

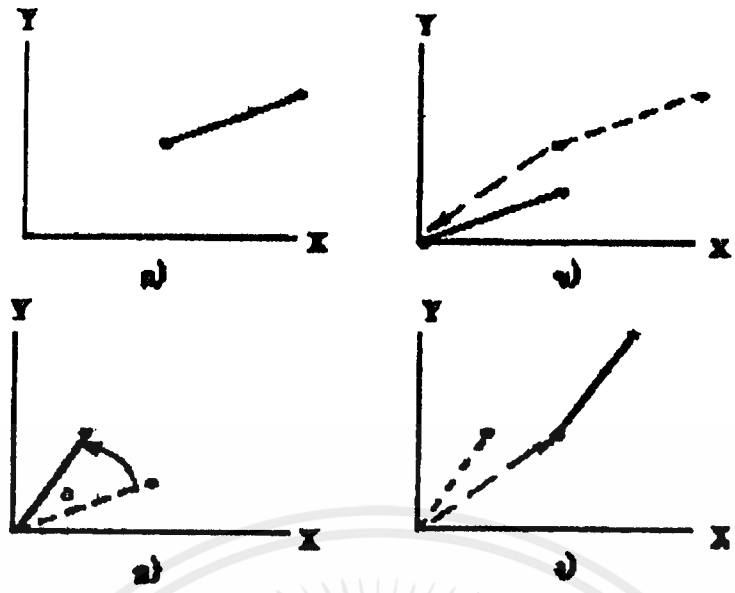


รูป 2.10 แสดงการหมุนรอบจุดใด ๆ

ขั้นตอนทั้ง 3 ขั้นนี้จะต้องทำทีละขั้นตอน เรียงลำดับให้ถูกต้อง เพราะในการทำแต่ละขั้นตอน ต้องใช้เมตริกซ์การแปลงมาคูณเข้าไป และการคูณเมตริกซ์ไม่มีคุณสมบัติการสลับที่ ดังนั้นถ้าคูณเมตริกซ์การแปลงไม่เรียงลำดับ จะได้ผลลัพธ์ที่ไม่ถูกต้อง

เมตริกซ์ที่จะย้ายจุด (x_c,y_c) ไปยังจุดกำเนิดก็คือ

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 1 \end{bmatrix}$$



รูป 2.11 การหมุนรอบจุดใด ๆ

เมตริกซ์สำหรับการหมุนคือ

$$R = \begin{bmatrix} \cos(a) & \sin(a) & 0 \\ -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

และเมตริกซ์ที่จะย้ายกลับไปยังจุดเดิมคือ

$$T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_c & y_c & 1 \end{bmatrix}$$

นำเมตริกซ์ทั้ง 3 มาคูณเข้ากับจุดที่เราจะหมุน P(x,y) ได้จุดใหม่เป็น $P_1(x_1,y_1)$

$$\begin{aligned} P_1 &= [(PT_1)R]T_2 \\ &= [P(T_1R)]T_2 \\ &= P(T_1RT_2) \end{aligned} \qquad (2.29)$$

ดังนั้นเมตริกซ์การแปลงแบบหมุนรอบจุด (x_c,y_c) คือ

$$T_1RT_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 1 \end{bmatrix}\begin{bmatrix} \cos(a) & \sin(a) & 0 \\ -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_c & y_c & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 1 \end{bmatrix}\begin{bmatrix} \cos(a) & \sin(a) & 0 \\ -\sin(a) & \cos(a) & 0 \\ x_c & y_c & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(a) & \sin(a) & 0 \\ -\sin(a) & \cos(a) & 0 \\ -x_c\cos(a)+y_c\sin(a)+x_c & -x_c\sin(a)-y_c\sin(a)+y_c & 1 \end{bmatrix} \qquad (2.30)$$

สมการ (2.30) คือ เมตริกซ์การแปลงที่หมุนภาพในทิศทางทวนเข็มนาฬิกาเป็นมุม a องศารอบจุด (x_c,y_c) เพื่อความสะดวกเราอาจเขียนเป็นสูตรสำหรับการหมุนจุด P(x,y) รอบจุด (x_c,y_c) เป็นมุม a องศาได้เป็นจุดใหม่ P'(x',y') ดังนี้

$$x' = (x-x_c)\cos(a) - (y-y_c)\sin(a) + xc \qquad (2.31)$$

$$y' = (x-x_c)\sin(a) + (y-y_c)\cos(a) + yc \qquad (2.32)$$

2.7 การแปลงแบบอื่น ๆ

การแปลงแบบสเกล การแปลงแบบหมุน และการแปลงแบบย้าย เป็นการแปลงที่มีประโยชน์และมีการใช้งานมากที่สุด แต่นอกเหนือจากการแปลงทั้ง 3 แล้ว ยังมีการแปลงแบบอื่น ๆ ที่น่าสนใจอีก เนื่องจากเมตริกซ์การแปลงขนาด 2x2 ใด ๆ

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

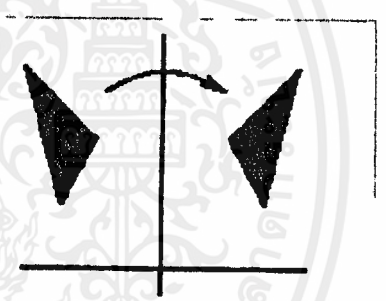
สามารถเปลี่ยนเป็นเมตริกซ์การแปลงขนาด 3x3 ในโฮโมจีเนียสโคออร์ดิเนทได้ดังนี้

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ดังนั้นเมตริกซ์การแปลงต่อไปนี้ จะเขียนอยู่ในรูปของเมตริกซ์ขนาด 2x2 เท่านั้น ดังต่อไปนี้

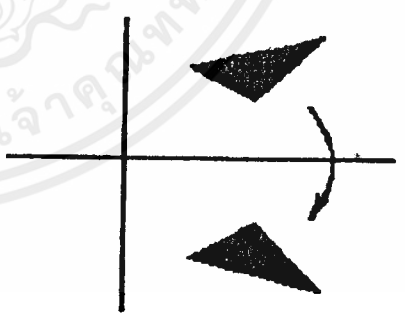
- *การสะท้อนกับแกน Y*

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



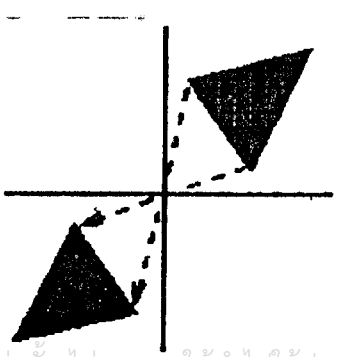
- *การสะท้อนกับแกน X*

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



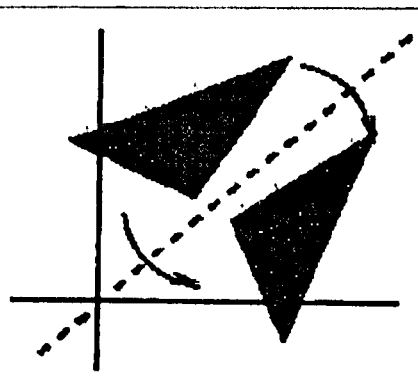
- *การสะท้อนกับจุดกำเนิด*

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

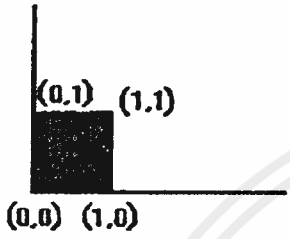


- *การสะท้อนกับเส้นตรง Y = X*

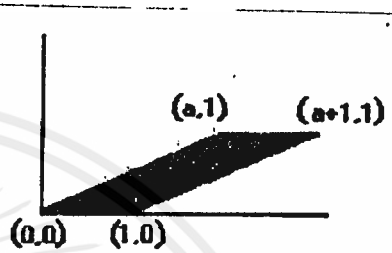
$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$



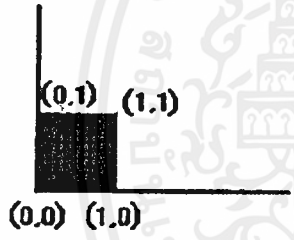
- *X shear*



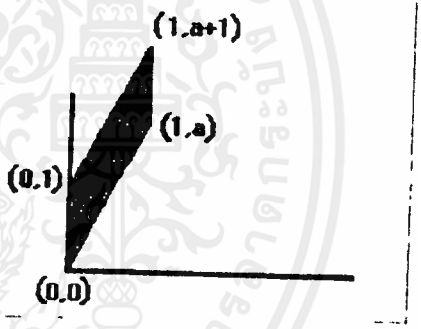
$$\begin{bmatrix} 1 & 0 \\ a & 1 \end{bmatrix}$$



- *Y shear*



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



2.8 การแปลงกลับ

เราได้เรียนรู้การแปลงแบบต่าง ๆ ที่ทำการแปลงจุด P(x,y) ไปเป็นจุด P'(x',y') มาแล้ว ในบางครั้งเราต้องการยกเลิกคำสั่งการแปลงนั้น ๆ หรือต้องการทราบว่าจุด P'(x',y') นั้นถูกแปลงมาจากจุดใด ๆ ในกรณีเช่นนี้เราต้องทำการแปลงแบบเดียวกันกับการแปลงจุด P ไปเป็นจุด P' แต่ใช้เมตริกซ์การแปลงเมตริกซ์ใหม่ การแปลงจากจุด P' กลับไปเป็นจุด P เรียกว่า **การแปลงกลับ (Inverse Transformation)** เมตริกซ์ที่นำมาใช้ในการแปลงกลับก็คือ **อินเวอร์สเมตริกซ์ของเมตริกซ์การแปลงจากจุด P ไปเป็นจุด P'**

สมมติว่า เราใช้เมตริกซ์การแปลง T แปลงจากจุด P ไปเป็นจุด P' ดังนั้น

$$P' = PT \qquad (2.33)$$

อินเวอร์สเมตริกซ์ของ T เขียนแทนด้วย T^{-1} จากคุณสมบัติของอินเวอร์สเมตริกซ์ จะได้ว่า

$$TT^{-1} = T^{-1}T = I \qquad (2.34)$$

โดยที่ I คือ เมตริกซ์เอกลักษณ์

เราทำการแปลงจุด P' ด้วยอินเวอร์สเมตริกซ์ของเมตริกซ์การแปลง T ได้จุดใหม่เป็น P"

$$\begin{aligned} P'' &= P'T^{-1} \\ &= (P\ T)\ T^{-1} \\ &= PTT^{-1} \\ &= PI \\ &= P \end{aligned} \qquad (2.35)$$

จากสมการ (2.35) จะเห็นว่าถ้าเราใช้เมตริกซ์ T^{-1} แปลงจุด P' เราจะได้จุด P กลับคืนมา

อินเวอร์สเมตริกซ์ของเมตริกซ์

$$T = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad (2.36)$$

คือ

$$T^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$T^{-1} = \begin{bmatrix} \frac{d}{ad-bc} & \frac{-b}{ad-bc} \\ \frac{c}{ad-bc} & \frac{a}{ad-bc} \end{bmatrix} \qquad (2.37)$$

ในกรณีของเมตริกซ์ขนาด 3x3

$$T = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix} \qquad (2.38)$$

จะได้อินเวอร์สเมตริกซ์ของ T คือ

$$T^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b & 0 \\ -c & a & 0 \\ cf-de & eb-af & ad-bc \end{bmatrix} \qquad (2.39)$$

สำหรับการแปลงแบบสเกลที่มีเมตริกซ์เป็น

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \qquad (2.40)$$

จะมีอินเวอร์สเมตริกซ์เพื่อแปลงกลับ คือ

$$S^{-1} = \begin{bmatrix} 1/S_x & 0 \\ 0 & 1/S_y \end{bmatrix} \qquad (2.41)$$

สำหรับการแปลงแบบหมุนที่มีเมตริกซ์เป็น

$$R = \begin{bmatrix} \cos(a) & \sin(a) \\ -\sin(a) & \cos(a) \end{bmatrix} \qquad (2.42)$$

จะมีอินเวอร์สเมตริกซ์ คือ

$$R^{-1} = \begin{bmatrix} \cos(a) & -\sin(a) \\ \sin(a) & \cos(a) \end{bmatrix} \qquad (2.43)$$

สำหรับการแปลงแบบย้ายที่มีเมตริกซ์เป็น

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \qquad (2.44)$$

จะมีอินเวอร์สเมตริกซ์คือ

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{bmatrix} \qquad (2.45)$$

บทที่ 3

คณิตศาสตร์สำหรับคอมพิวเตอร์กราฟิกในระนาบ 2 มิติ

3.1 TRANSFORMATION OF POINTS

เมื่อเราแทนจุด P ในระนาบ 2 มิติบนพิกัด (X,Y) แล้วแปลงจุด P เป็นจุด P' พิกัด (X',Y') ซึ่งเกิดจากการคูณจุด P กับเมตริกซ์ T ขนาด 2x2

$$T = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$P' = PT = (x,y) \begin{bmatrix} A & B \\ C & D \end{bmatrix} = [(Ax + Cy),(Bx + Dy)] \quad (3.1)$$

3.1.1 IDENTITY

ถ้า B = C = 0 และ A = D = 1 แล้วเมตริซ์ T จะเป็นเมตริกซ์เอกลักษณ์

$$(x',y') = (x,y) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = (x,y) \quad (3.2)$$

จะเห็นได้ว่าถ้าเป็น identity transformation แล้วพิกัด P จะไม่เปลี่ยนแปลง

3.1.2 SCALING

กรณี D = 1 และ B = C = 0 จะเป็นการเปลี่ยน scale ในทิศทางของแกน X

$$(x',y') = (x,y)\begin{bmatrix} A & 0 \\ 0 & 1 \end{bmatrix} = (Ax,y) \qquad (3.3)$$

แต่ถ้า A = 1 และ B = C = 0 จะทำให้เปลี่ยน scale ตามแนวแกน y

$$(x',y') = (x,y)\begin{bmatrix} 1 & 0 \\ 0 & D \end{bmatrix} = (x,Dy) \qquad (3.4)$$

ถ้าเพียงแต่ B = C = 0 จะเกิดการเปลี่ยน scale ทั้งในทิศทางแกน X และ Y

$$(x',y') = (x,y)\begin{bmatrix} A & 0 \\ 0 & D \end{bmatrix} = (Ax,Dy) \qquad (3.5)$$

3.1.3 REFLECTION

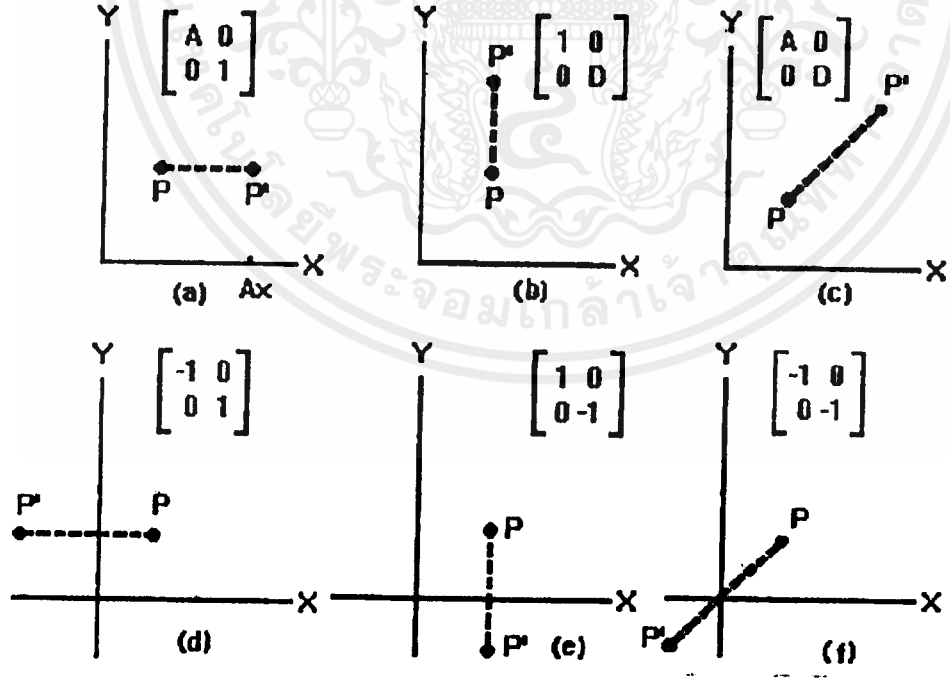
จะเกิดการ Reflection ก็ต่อเมื่อ A หรือ D หรือทั้งคู่ มีเครื่องหมายตรงข้าม ในเมตริกซ์ T ถ้าเกิด Reflection ของจุดกับแกน X หรือแกน Y จะเกิดภาพสะท้อนของจุดบนฝั่งตรงกันข้ามของแกน

$$(x',y') = (x,y)\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = (-x,y) \qquad (3.6a)$$

กรณีนี้จะเกิดการสะท้อนในแนวแกน X

$$(x',y') = (x,y)\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = (x,-y) \qquad (3.6b)$$

จะเกิดการสะท้อนในแนวแกน Y แต่ถ้า A = D = -1 และ B = C = 0 แล้วจะเกิดการสะท้อนกับจุด origin (0,0)



รูป 3.1 Transformation of points

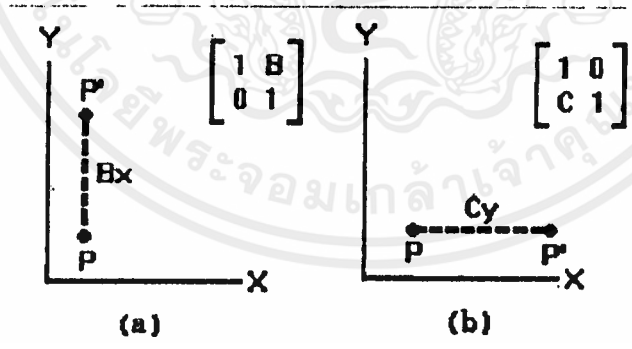
3.1.4 SHEAR

เมตริกซ์ T ที่มี A = D = 1 และ C = 0

$$(x',y') = (x,y)\begin{bmatrix} 1 & B \\ 0 & 1 \end{bmatrix} = (x, Bx + y) \qquad (3.7)$$

เป็นการแปลงโดยที่พิกัดทางแกน X ไม่เปลี่ยนแปลง ในขณะที่พิกัดทางแกน Y มีความสัมพันธ์เชิงเส้นกับ X และ Y เรียกว่า Shear ในทิศทางแกน Y ถ้า Shear ในทิศทางแกน X แสดงดังนี้

$$(x',y') = (x,y)\begin{bmatrix} 1 & 0 \\ C & 1 \end{bmatrix} = (x + Cy, y) \qquad (3.8)$$



รูป 3.2 ผลที่เกิดจาก shearing

3.1.5 ROTATION

จุดสามารถหมุนเป็นมุม • กับจุด origin สามารถแสดงในรูปเมตริกซ์ ดังนี้

$$(x',y') = (x,y)\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \qquad (3.9)$$

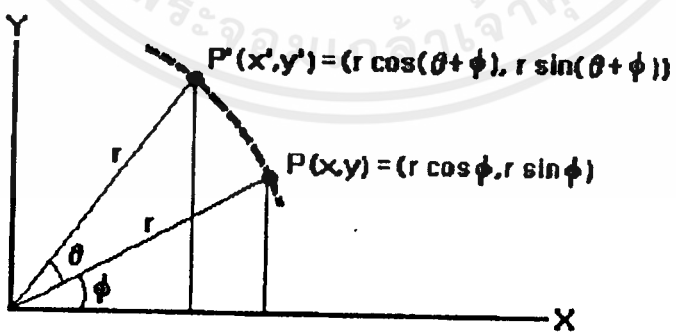
$$= (x\cos\theta - y\sin\theta\ ,\ x\sin\theta + y\cos\theta)$$

ถ้ามุม θ เป็นบวกจะหมุนทวนเข็มนาฬิกา (counterclockwise) จากแกน X ไปยังแกน Y ถ้าต้องการหมุนตามเข็มนาฬิกา (clockwise) ใส่ $-\theta$ ลงไป

$$(x',y') = (x,y)\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \qquad (3.10)$$

$$= (x\cos\theta + y\sin\theta\ ,\ -x\sin\theta + y\cos\theta)$$

ได้จาก $\cos(-\theta) = \cos\theta$ และ $\sin(-\theta) = -\sin(\theta)$



รูป 3.3 การหมุนของจุดในระนาบ 2 มิติ

จากรูปจุด P หมุนเป็นมุมบวก เกิดจุดใหม่คือ P' โดยระยะจากจุด origin (0,0) คงที่

คือระยะ r เพราะเป็นการหมุนเทียบกับจุด origin ที่จุด P แสดงได้ดังนี้

$$x = r \cos \phi$$

$$y = r \sin \phi$$

และ $$x' = r \cos (\theta + \phi) = r \cos \phi \cos \theta - r \sin \phi \sin \theta$$

$$= x \cos \theta - y \sin \theta$$

$$y' = r \sin (\theta + \phi) = r \cos \phi \sin \theta + r \sin \phi \cos \theta$$

$$= x \sin \theta + y \cos \theta$$

ซึ่งจะเหมือนกับสมการที่แสดงไว้ข้างบน

3.2 TRANSFORMATION OF LINES AND OBJECT

เส้นตรงเกิดจากการกำหนดจุดตำแหน่งของ vector ที่ปลายทั้งสอง การทำ line transformation สามารถกระทำที่จุดปลายทั้งสองของ vector และลากเส้นตรงขึ้นใหม่ระหว่างจุดที่เกิดขึ้นใหม่

object ก็ถูกสร้างขึ้นจากเส้นตรงหลายเส้น เราสามารถทำ transformation object โดยการ transform vertices แต่ละจุดบน object ถ้า object ประกอบขึ้นจาก n vertices รูปแบบการแปลงเป็นดังนี้

Transformaed object coordinate		Original object coordinate		Transformation matrix

$$\begin{bmatrix} P_1' \\ P_2' \\ . \\ . \\ . \\ P_n' \end{bmatrix} = \begin{bmatrix} x_1' & y_1' \\ x_2' & y_2' \\ . & . \\ . & . \\ . & . \\ x_n' & y_n' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ . & . \\ . & . \\ . & . \\ x_n & y_n \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

3.2.1 SCALING

$$\begin{bmatrix} P_1' \\ P_2' \\ P_3' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & D \end{bmatrix} = \begin{bmatrix} x_1' & y_1' \\ x_2' & y_2' \\ x_3' & y_3' \end{bmatrix}$$

3.2.2 ROTATION

$$\begin{bmatrix} P_1' \\ . \\ . \\ . \\ P_n' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ . & . \\ . & . \\ . & . \\ x_n & y_n \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \qquad (3.11)$$

3.2.3 TRANSLATION

วัตถุ (Object) สามารถ translated คือการ shift บน plane โดยการบวก magnitude กับแต่ละ vertex ของ Object ซึ่งมีรูปแบบ

Translated coordinates = Original coordinate + Translation magnitude

$$\begin{bmatrix} P_1' \\ . \\ . \\ . \\ P_n' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ . & . \\ . & . \\ . & . \\ x_n & y_n \end{bmatrix} + \begin{bmatrix} T_x & T_y \\ . & . \\ . & . \\ . & . \\ T_x & T_y \end{bmatrix} \qquad (3.12)$$

3.3 HOMOGENEOUS COORDINATE SYSTEM

ใน homogeneous coordinate system จุด (x,y) แทนโดย (x,y,H) เมื่อ H เป็น scale ที่ไม่เท่ากับ 0 normalized homogeneous coordinate แทนจุด (x,y) ด้วย (x/H,y/H,1) เมื่อ H เป็นค่าคงที่ไม่เท่ากับศูนย์ ปกติเรามักเลือก H = 1 จึงเขียนในรูป (x,y,1)

3.3.1 REVISED TRANSFORMATION MATRIX

ในระบบ homogeneous coordinate transformation matrix 2x2 จะต้องขยายเป็นเมตริกซ์ 3x3 ดังนี้

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \longrightarrow \begin{bmatrix} A & B & 0 \\ C & D & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3.13)$$

ต้องการเพิ่มคุณสมบัติ translation เข้าไปในเมตริกซ์ 3x3 โดยการเปลี่ยนแถวที่ 3 จาก (0 0 1) เป็น (L M 1) ซึ่งจะมีผลกับการเคลื่อนที่เชิงเส้นของจุดทางแนวนอน L หน่วย และแนวตั้ง M หน่วย เมตริกซ์ที่เปลี่ยนแปลงจะอยู่ในรูป

$$T = \begin{bmatrix} A & B & 0 \\ C & D & 0 \\ L & M & 1 \end{bmatrix} \qquad (3.14)$$

ผลของการแปลงจุด (x,y) แสดงโดย

$$(x',y',1) = (x,y,1) \cdot T$$

หรือ

$$x' = Ax + Cy + L$$

$$y' = Bx + Dy + M$$

3.3.2 TRANSLATION

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ L_1 & M_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ L_2 & M_2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ L_1+L_2 & M_1+M_2 & 1 \end{bmatrix} \qquad (3.15)$$

3.3.3 SCALING

$$\begin{bmatrix} A_1 & 0 & 0 \\ 0 & D_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_2 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} A_1A_2 & 0 & 0 \\ 0 & D_1D_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3.16)$$

3.3.4 ROTATION

(3.17)

$$\begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & \sin\theta_2 & 0 \\ -\sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_1+\theta_2) & \sin(\theta_1+\theta_2) & 0 \\ -\sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.3.5 INVERSES OF TRANSFORMATION

$$T_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ -L & -M & 1 \end{bmatrix} \quad , \quad T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ L & M & 1 \end{bmatrix}$$

เมตริกซ์ T_1 และ T_2 จะให้ผลที่ตรงกันข้าม T_1 ทำให้เกิดการ translation(-L,-M) ขณะที่ T_2 ทำให้เกิด translation (L,M) ถ้าคูณ T_1 กับ T_2 จะเกิด identity transformation ซึ่งตำแหน่งของจุดไม่มีการเปลี่ยนแปลง

$$T_1T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -L & -M & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ L & M & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.18)$$

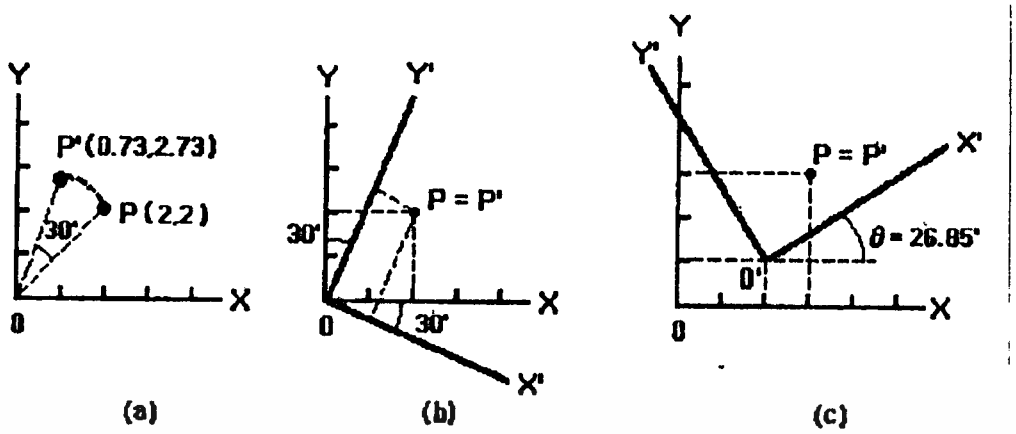
เช่นเดียวกับการหมุนที่มุมเท่ากัน แต่ตรงกันข้ามก็จะได้ Identity Matrix

เช่น $R(30) \cdot R(-30) = I$

3.3.6 TRANSFORMAING A COORDINATE SYSTEM

จากที่กล่าวมาเป็นการแปลง set of point ไปเป็น set of point ซึ่งอยู่ในระบบ coordinate เดียวกัน ซึ่งคล้ายกับการแปลง set of point โดยการเปลี่ยนระบบ coordinate

ถ้าจุด P หมุนไปทวนเข็มนาฬิกาเป็นมุม θ เป็นจุด P' ซึ่งพิกัดกับ P' บน coordinate ที่หมุนตามเข็มนาฬิกาเป็นมุม θ จะเป็นจุดเดียวกับ P พอดี ดังรูป



รูป 3.4 ความสัมพันธ์ระหว่าง (a) transforming a point,
(b) transforming a coordinate system
(c) จุด P กับ coordinate ทั้งสองระบบ

จากรูปที่ 3.4 (c) จุด P (3,3) บน coordinate XY จะมีตำแหน่งเดียวกัน แต่พิกัด (3.5,2.5) บน coordinate X'Y' การ translation coordinate จาก XY ไปยัง X'Y' เริ่มจาก translate origin ไป O' เปลี่ยน scale ทั้ง 2 แกน และสุดท้ายหมุน coordinate X'Y' ตามเข็มนาฬิกาเป็นมุม ϕ

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2\cos\phi & -2\sin\phi & 0 \\ 2\sin\phi & 2\cos\phi & 0 \\ -4\cos\phi - 2\sin\phi & 4\sin\phi - 2\cos\phi & 1 \end{bmatrix}$$

$$P' = (3,3,1) \cdot T = (2\cos\phi + 4\sin\phi, -2\sin\phi + 4\cos\phi, 1)$$

ถ้าเราทราบมุม θ ก็สามารถทราบตำแหน่ง P' จากตัวอย่าง P' (3.5,2.5)

$$2\cos\theta + 4\sin\theta = 3.5$$

$$-2\sin\theta + 4\cos\theta = 2.5$$

ปกติแล้ว ถ้าเรากำหนด T คือผลคูณของ n เมตริกซ์

$$T = T_1 . T_2 ..... T_{n-1} . T_n$$

$$T^{-1} = T_n^{-1} . T_{n-1}^{-1} ..... T_2^{-1} . T_1^{-1}$$

จากตัวอย่างจะได้

$$T^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/2\cos\theta & 1/2\sin\theta & 0 \\ -1/2\sin\theta & 1/2\cos\theta & 0 \\ 2 & 1 & 1 \end{bmatrix}$$

$$P = P'.T^{-1} = (1.75\cos\theta - 1.25\sin\theta + 2, 1.75\sin\theta + 1.25\cos\theta + 1, 1)$$

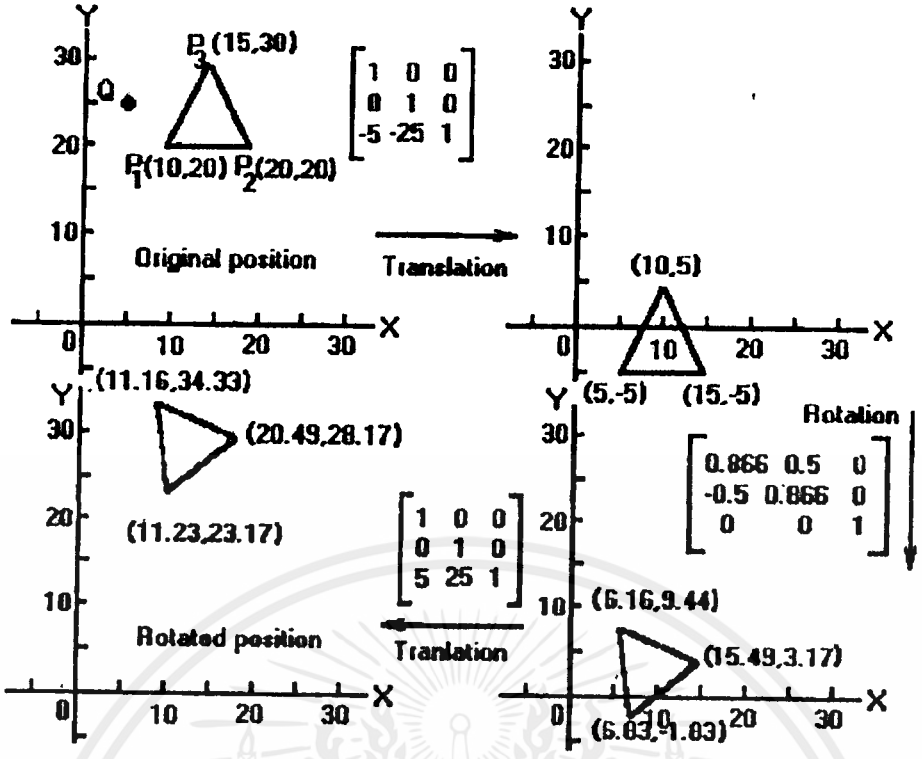
$$= (3,3,1)$$

3.4 SEQUENTIAL 2-D TRANSFORMATION

4.4.1 หมุนรอบจุด

ขั้นที่ 1 ย้ายจุด center ของการหมุนไปที่ origin

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -q_1 & -q_2 & 1 \end{bmatrix}$$



รูป 3.5 การหมุนรอบจุดในระนาบ 2 มิติ

ขั้นที่ 2 หมุนรอบ origin

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

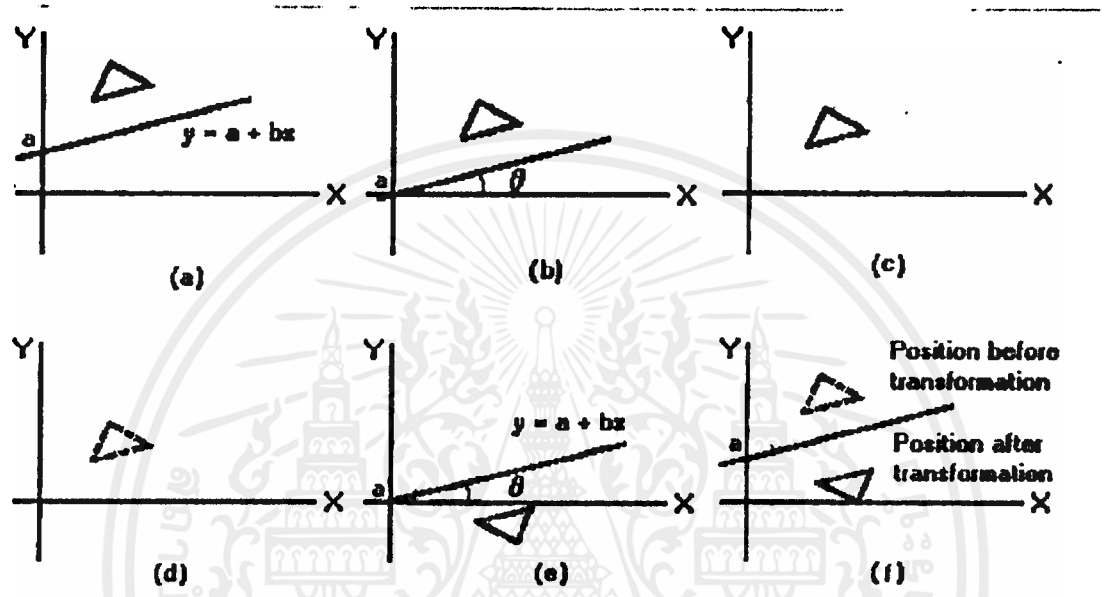
ขั้นที่ 3 ย้ายจุด Q ไปยังที่เดิม

$$T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ q_1 & q_2 & 1 \end{bmatrix}$$

จะได้ $T = T_1 . R(\theta) . T_2$

$$= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ q_1(1-\cos\theta)+q_2\sin\theta & q_2(1-\cos\theta)+q_1\sin\theta & 1 \end{bmatrix}$$

3.4.2 REFLECTION กับแกนใด ๆ



รูป 3.6 ภาพสะท้อนกับแกน (a) orginal figure, (b) translation, (c) rotation, (d) reflection, (e) rotation, (f) translation

แกนซึ่งมีสมการ $y = a + bx$

ขั้นที่ 1 ย้าย origin ไปยังพิกัด (0,a)

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -a & 1 \end{bmatrix}$$

ขั้นที่ 2 หมุนรอบแกนตามเข็มนาฬิกาเป็นมุม θ จากสมการ $y = a + bx$ มุม θ สามารถคำนวณจาก slop จากแกนคือ b โดย $\cos\theta = 1/r$, $\sin\theta = b/r$ โดยที่ $r = (b^2 + 1)^{1/2}$

$$R(-\theta) = \begin{bmatrix} 1/r & -b/r & 0 \\ b/r & 1/r & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ขั้นที่ 3 Reflect กับแกน X

$$T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ขั้นที่ 4 ทำตรงข้ามกับขั้นที่ 2

$$R(\theta) = \begin{bmatrix} 1/r & b/r & 0 \\ -b/r & 1/r & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ขั้นที่ 5 ทำตรงข้ามกับขั้นที่ 1

$$T_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & a & 1 \end{bmatrix}$$

จะได้ transformation matrix ที่สมบูรณ์

$$T = T_1 R(-\theta)\ T_2 R(\theta)\ T_1^{-1}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -a & 1 \end{bmatrix} \begin{bmatrix} \cos 2\theta & \sin 2\theta & 0 \\ \sin 2\theta & -\cos 2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & a & 1 \end{bmatrix}$$

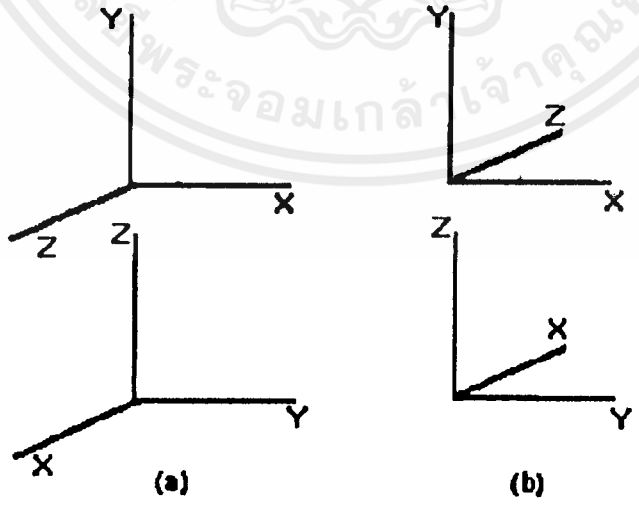
บทที่ 4

คณิตศาสตร์สำหรับคอมพิวเตอร์กราฟิกในระนาบ 3 มิติ

ในบทนี้จะกล่าวถึงเทคนิคที่ใช้ในกราฟิก 3 มิติ เช่น การวาด wire frame หรือ รูปเงา การกระทำกับภาพ 3 มิตินั้นจะต้องใช้เทคนิคที่ยุ่งยาก โชคร้ายที่จอภาพของคอมพิวเตอร์แสดงได้เพียง 2 มิติ ดังนั้นจะต้องทำการแปลงภาพวัตถุ 3 มิติให้อยู่ในรูป 2 มิติ ซึ่งจะทำได้โดยการแนะนำเรื่อง perspective projection และการแปลงวัตถุ 3 มิติให้อยู่ในระนาบ projection 2 มิติ ดังนั้นในบทนี้กล่าวถึงตั้งแต่เริ่มต้นถึง projection และการแปลงรูป 3 มิติ และจะอธิบายว่าจะต้องใช้คณิตศาสตร์ช่วยอย่างไร

4.1 ระบบโคออร์ดิเนท

ในโลกของ 3 มิติ เราจะต้องมีระบบโคออร์ดิเนท 3 มิติ โดยที่ระบบโคออร์ดิเนท 3 มิติสามารถแสดงให้อยู่ในระบบโคออร์ดิเนท 2 มิติได้ (ระนาบ XY) ซึ่งจะเพิ่มแกน Z ขึ้นมา ทำให้มีระนาบเพิ่มขึ้นอีก 2 ระนาบ คือ ระนาบ XZ และระนาบ YZ คำแนะนำสองข้อที่เป็นไปได้เมื่อเราเลือกที่จะเพิ่มแกน Z ขึ้นมา คือ ถ้าจุดในแกน Z พุ่งเข้าหาตา ระบบโคออร์ดิเนทที่เกี่ยวข้องที่ถูกอ้างถึงคือ กฎมือขวา (rigth-hand system) แต่ถ้าระบบโคออร์ดิเนทที่แกน Z พุ่งออกไปจะใช้กฎมือซ้าย (left-hand system) ซึ่ง 2 กฎนี้จะแสดงให้เห็นได้ดังรูปที่ 4.1 ซึ่งโดยทั่วไปแล้วจะใช้กฎมือขวามากกว่า



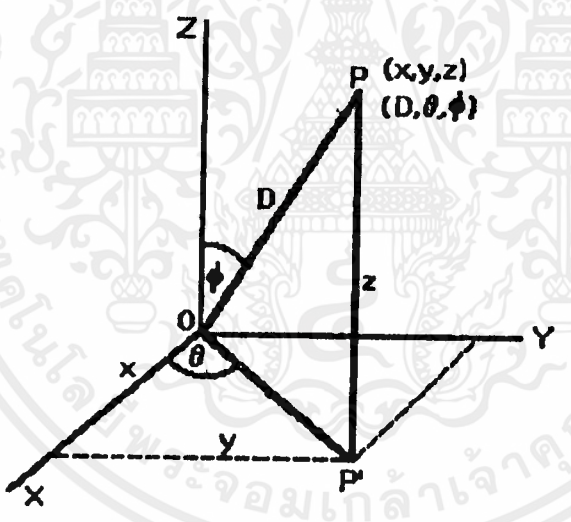
รูป 4.1 ระบบ coordinate (a) right-hand system (b) left-hand system

ระบบโคออร์ดิเนทที่แทนจุดในระบบ 3 มิตินั้นมีอยู่ 2 ระบบคือ ระบบโคออร์ดิเนทแบบตั้งฉาก (rectangular coordinate system) และระบบโคออร์ดิเนทแบบทรงกลม (spherical coordinate system)

Rectangular coordinate system จุดจะถูกกำหนดโดยตัวเลข 3 ตัวซึ่งเป็นตัวเลขในแกน X, Y และ Z โดยที่จะเป็นการบอกระยะทางของจุดในแต่ละแกน (ดูรูปที่ 4.2)

Spherical coordinate system ในระบบนี้ แต่ละจุดจะถูกแทนด้วยตัวเลข 3 ตัวคือ (D, θ, ϕ) ระยะทางจากจุด P ถึง origin คือ D มุมระหว่างเส้น OP และแกน Z ที่มีค่าเป็นบวกคือ ϕ มุมระหว่างแกน X ที่มีค่าเป็นบวกและเส้น OP' (projection ของ OP บนระนาบ XY) คือ θ มุมนี้จะถูกวัดในทิศทางทวนเข็มนาฬิกาจากแกน X

4.1.2 ความสัมพันธ์ระหว่างระบบโคออร์ดิเนตแบบ Rectangular และ Spherical



รูป 4.2 ความสัมพันธ์ระหว่างระบบโคออร์ดิเนตแบบ Rectangular และ Spherical

เพราะระบบโคออร์ดิเนททั้ง 2 ถูกใช้ในระบบคอมพิวเตอร์กราฟิก ดังนั้นจำเป็นที่จะต้องแปลงไปมาระหว่าง 2 ระบบ ซึ่งเราสามารถสร้างตามความสัมพันธ์ระหว่างระบบโคออร์ดิเนท 2 ระบบดังนี้

$$x = D \cdot \sin\phi \cos\theta \ ,$$

$$y = D \cdot \sin\phi \sin\theta \ ,$$

$$z = D \cdot \cos\phi$$

และ (4.1)

$$D = (x^2 + y^2 + z^2)^{1/2} ,$$

$$\theta = \tan^{-1} (y / x) ,$$

$$\phi = \cos^{-1} (z/D)$$

สำหรับตัวอย่าง , จุด (4,2,3) ใน rectangular coordinate system จะเท่ากับจุด (5.4,27,56) ใน spherical coordinate system

4.2 TRANSFORMATION MATRIX

เราจะนำระบบ homogeneous coordinate มาใช้สำหรับการแปลงรูป 3 มิติ เราจะใช้ rectangular coordinate system แทนตำแหน่งเวกเตอร์ (x,y,z) transformation matrix โดยทั่วไปจะใช้ 4x4 เมตริกซ์ สำหรับจุดใน 3 มิติ homogeneous coordinate system คือ

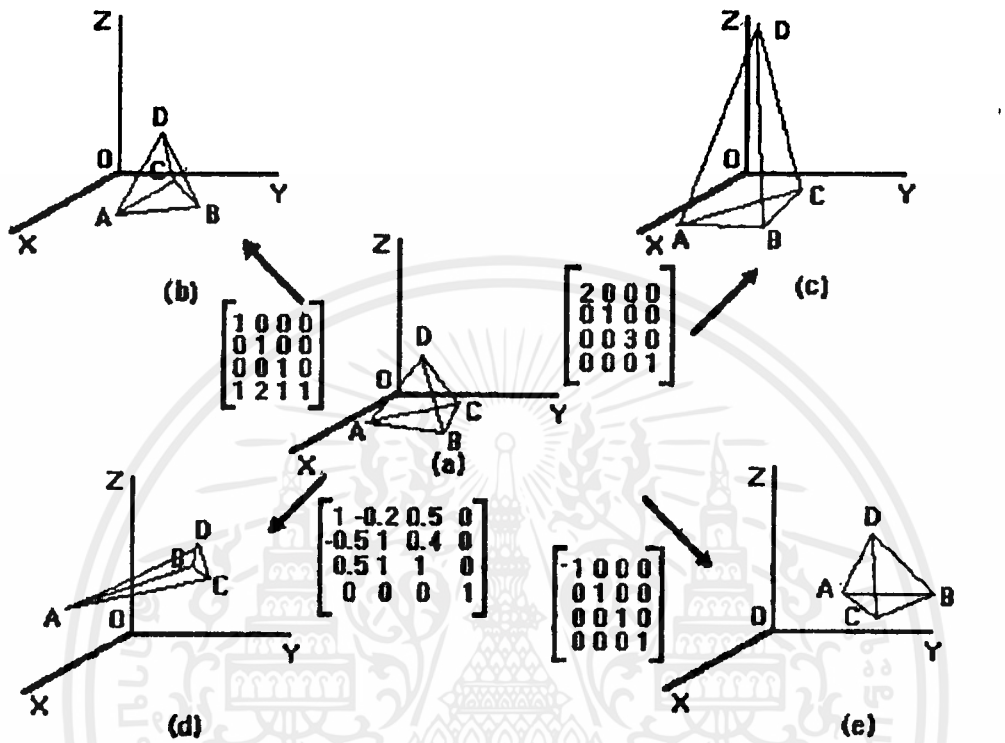
$$\begin{bmatrix} A & B & C & 0 \\ D & E & F & 0 \\ G & H & I & 0 \\ L & M & N & 1 \end{bmatrix} \qquad (4.2)$$

และสามารถแบ่งเป็น 3 ส่วนของเมตริกซ์ย่อย

$$\begin{bmatrix} \text{I} \\ (3 \times 3) \end{bmatrix} \quad \begin{bmatrix} \text{III} \\ 4 \\ x \end{bmatrix} \qquad (4.3)$$

$$\begin{bmatrix} \text{II} \\ (1 \times 3) \end{bmatrix} \quad \begin{bmatrix} 1 \end{bmatrix}$$

เมตริกซ์ย่อย I จะใช้กระทำ Scale , Shearing , Reflection และการหมุนของวัตถุ 3 มิติ เมตริกซ์ย่อย II จะทำการเคลื่อนย้ายเส้นตรง และเมตริกซ์ย่อย III อนุญาตให้ทำการเคลื่อนย้าย และการแปลงในแบบการคูณ เราจะพิจารณาคุณสมบัติของเมตริกซ์ย่อยเหล่านี้ในส่วนนี้ และแสดงให้เห็นในรูปประมิด ซึ่งจะเห็นดังรูป 4.3



รูป 4.3 transformation ในระบบ 3 มิติ

4.2.1 3-D TRANSLATION

การแปลงการเคลื่อนย้ายจุด (x,y,z) ไปเป็นจุดใหม่ (x',y',z') จะใช้ (L,M,N) ซึ่งถูกแสดงโดย

$$[x',y',z',1] = [x,y,z,1] \begin{bmatrix} 1 & 0 & 0 & : & 0 \\ 0 & 1 & 0 & : & 0 \\ 0 & 0 & 1 & : & 0 \\ \multicolumn{5}{c}{\text{----------+---}} \\ L & M & N & : & 1 \end{bmatrix} \qquad (4.4)$$

โดยที่ L,M และ N เป็นระยะทางที่เลื่อนจาก x,y และ z ตามลำดับ รูป 4.3(b) จะแสดงผลการเคลื่อนย้ายรูปประมิด 3 มิติ ถึง 1,2 และ 1 หน่วยในแกน x,y และ z ตามลำดับ

4.2.2 3-D SCALING

เทอมของเส้นทแยงมุมของ transformation matrix 4x4 โดยทั่วไปจะจัดอยู่ในรูป scale การแปลงดังนี้

$$[x',y',z',1] = [x,y,z,1] \begin{bmatrix} A & 0 & 0 & 0 \\ 0 & E & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.5)$$

จากสมการ (4.5) ทำให้เราเห็นว่า เราสามารถควบคุม scale ของโคออร์ดิเนทแต่ละแกนได้ การตั้ง scale แบบง่าย ๆ ทำได้โดยเซตค่า A = E = I ถ้าต้องการให้วัตถุใหญ่กว่าเดิมเป็นเฟกเตอร์ของ 2 จะทำได้โดยเซตค่า A = E = I = 2 ซึ่งผลของการตั้ง scale แสดงดังรูปที่ 4.3(c)

4.2.3 3-D SHEARING

3-D Shearing จะทำได้โดยการ ปิดเทอมของเส้นทแยงมุมในเมตริกซ์ 3x3 บนซ้ายของสมการ (4.2) ผลลัพธ์ของการ shearing เป็นดังนี้

$$[x',y',z',1] = [x,y,z,1] \begin{bmatrix} 1 & B & C & 0 \\ D & 1 & F & 0 \\ G & H & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.6)$$

3-D Shearing บนปิระมิดจะแสดงได้ดังรูป 4.3(d)

4.2.4 3-D REFLECTION

วัตถุจะสะท้อนไปยังระนาบ ๆ หนึ่ง ทำได้โดยการย้ายตัวเลขในเส้นทแยงมุมของ 3-D transformation matrix พิจารณาจากบทที่ 1 วัตถุจะสะท้อนถึงระนาบ ๆ หนึ่งโดยจะสร้างภาพสะท้อน (ภาพเงา) ของจุดทั้งหมดของวัตถุบนระนาบตรงกันข้าม สำหรับตัวอย่าง,ในการสะท้อนถึงระนาบ XY เพียงค่าในโคออร์ดิเนท Z ของตำแหน่งของวัตถุเท่านั้น ที่จะมีผลที่ถูกเปลี่ยนแปลง ซึ่งจะต้องทำการเปลี่ยนเครื่องหมายค่าในโคออร์ดิเนท Z นั้น ดังนั้น transformation matrix สำหรับการสะท้อนนี้ คือ

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.7)$$

สำหรับการสะท้อนถึงระนาบ YZ จะได้เมตริกซ์ดังนี้

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.8)$$

และสำหรับการสะท้อนถึงระนาบ XZ จะได้เมตริกซ์ ดังนี้

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.9)$$

การสะท้อนของปิระมิดถึงระนาบ YZ จะแสดงได้ดังรูป 4.3(e)

4.2.5 การหมุนรูป 3 มิติ

ก่อนที่จะพิจารณากรณีทั่วไปของการหมุนรูป 3 มิติรอบแกนจำลอง เราจะกล่าวถึงการหมุนรอบแกนโคออร์ดิเนท X,Y,Z ก่อน อันดับแรกจะพิจารณาการหมุนรอบแกน Z ในกรณีนี้มิติในแกน Z จะไม่เปลี่ยน ซึ่งเมตริกซ์การแปลงจะมีค่าเป็น 0 ในแถวที่ 3 ทั้งแถวและคอลัมน์ที่ 3 ทั้งคอลัมน์ ยกเว้นในเส้นทแยงมุมหลักจะเป็น 1 สำหรับเทอมอื่น ๆ จะถูกกำหนดโดยการพิจารณาได้จากการหมุนของวัตถุ 2 มิติ แสดงในสมการ (3.9) ตัวอย่างเช่นการหมุนรอบแกน Z ใน 3 มิติ transformation matrix สำหรับการหมุนของมุม θ รอบแกน Z คือ

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.10)$$

เราจะวัดมุมของการหมุน θ ซึ่งจะวัดทวนเข็มนาฬิกาโดยจุด origin เมื่อดูที่จุด origin จากจุดบนแกน X ที่เป็นบวก

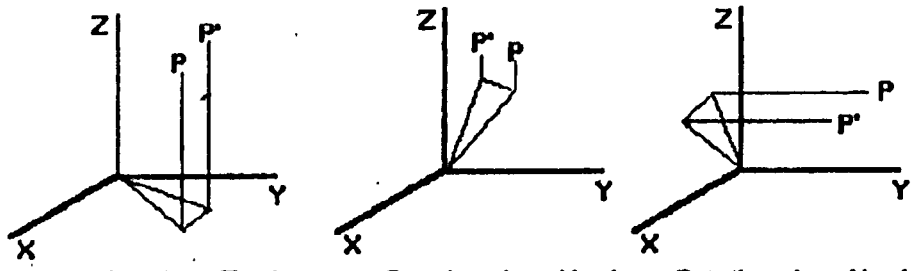
transformation matrix ที่มีผลต่อการหมุนเป็นมุม θ รอบแกน X และแกน Y มีดังนี้

$$R(\theta)_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.11)$$

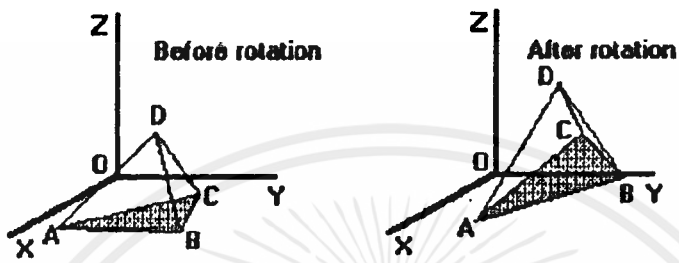
และ

$$R(\theta)_y = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.12)$$

การหมุนรูป 3 มิติรอบแต่ละแกน จะแสดงได้ดังรูป 4.4



(a)



(b)

รูป 4.4 การหมุนในระบบ 3 มิติ (a) การหมุนของจุด, (b) การหมุนของวัตถุ

ตัวอย่างของการหมุนรูป 3 มิติ , พิจารณาปิระมิดที่ให้ในรูป 4.4(a) ถ้าปิระมิดถูกหมุนไป 30 องศา (ทวนเข็มนาฬิกา) รอบแกน Z ใช้สมการ (4.10) เราจะได้โคออร์ดิเนทใหม่ดังนี้

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} 3 & 1 & 0 & 1 \\ 3 & 5 & 0 & 1 \\ 1 & 4 & 0 & 1 \\ 2 & 3 & 3 & 1 \end{bmatrix} \begin{bmatrix} 0.866 & 0.5 & 0 & 0 \\ -0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2.1 & 2.37 & 0 & 1 \\ 0.1 & 5.83 & 0 & 1 \\ 1.13 & 3.96 & 0 & 1 \\ 0.232 & 3.60 & 3 & 1 \end{bmatrix}$$

ผลของการแปลงจะแสดงในรูป 4.4(b)

สมมติประมิดเดิม แต่หมุนไป -30 องศา (ตามเข็มนาฬิกา) รอบแกน Z ด้วย $\theta = -30$ และใช้สมการ (4.10) เราจะได้

$$R(-30) = \begin{bmatrix} 0.866 & -0.5 & 0 & 0 \\ 0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

นี่คือ อินเวอร์เมตริกซ์ของ I :

$$R(30).R(-30) = \begin{bmatrix} 0.866 & 0.5 & 0 & 0 \\ -0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.866 & -0.5 & 0 & 0 \\ 0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = R(30).R(30)^{-1} = I$$

ที่กล่าวมาข้างต้นนี้ เมตริกซ์การหมุนมุมตามเข็มนาฬิการอบแกน จะเหมือนกับเมื่ออินเวอร์ $[R(\theta)^{-1}]$ ของเมตริกซ์การหมุนตามมุมทวนเข็มนาฬิการอบแกนเดียวกัน $[R(\theta)]$ โดยที่จะแทนเป็นกฎทั่วไป ซึ่งเราจะพบว่าเป็นประโยชน์ในคอมพิวเตอร์กราฟิกมาก โดยที่กฎนี้มีดังนี้

$$R(\theta)_z^{-1} = R(-\theta)_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.13)$$

$$R(\theta)_x^{-1} = R(-\theta)_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.14)$$

และ

$$R(\theta)_y^{-1} = R(-\theta)_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.15)$$

หมายเหตุที่สำคัญ, โดยทั่วไป order ของการคูณจะมีผลซึ่งเป็นผลลัพธ์สุดท้าย ทั้งนี้เพราะการหมุนรูป 3 มิติ บางทีเป็นแบบ noncommulative เช่นอาจจะหมุนรอบแกน Z เป็นมุม θ_1 ตามด้วยหมุนรอบแกน X เป็นมุม θ_2 :

$$R(\theta_1)_z.R(\theta_2)_x = \begin{bmatrix} \cos\theta_1 & \sin\theta_1\cos\theta_2 & \sin\theta_1\sin\theta_2 & 0 \\ -\sin\theta_1 & \cos\theta_1\cos\theta_2 & \cos\theta_1\sin\theta_2 & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.16)$$

ในทางกลับกัน ถ้าหมุนรอบแกน X ด้วยมุม θ_2 ตามด้วยการหมุนรอบแกน Z ด้วยมุม θ_1 จะได้

$$R(\theta_2)_x.R(\theta_1)_z = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 & 0 \\ -\cos\theta_2\sin\theta_1 & \cos\theta_2\cos\theta_1 & \sin\theta_2 & 0 \\ \sin\theta_2\sin\theta_1 & -\sin\theta_2\cos\theta_1 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.17)$$

เมื่อเปรียบเทียบผลลัพธ์ของทั้ง 2 จะไม่เท่ากัน ดังนั้นถ้าการหมุนรูป 3 มิติ เป็นแบบ noncommutative คือหมุนหลายแกน (มีการคูณ transformation matrix) จะใช้วิธีนี้ไม่ได้

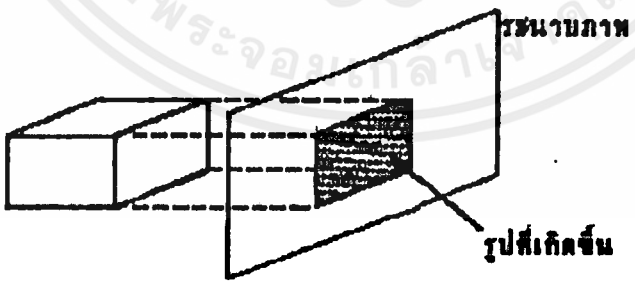
บทที่ 5

การสร้างภาพ 3 มิติ

ในบทที่แล้วเราได้ศึกษาเกี่ยวกับระบบภาพ 3 มิติ ซึ่งเป็นการกล่าวถึงโคออร์ดิเนทในระบบ 3 มิติทั้งหมด สำหรับการแปลงเป็นการแปลงวัตถุในระบบ 3 มิติเท่านั้น อย่างไรก็ตามถึงแม้ว่าเราจะมีวัตถุ 3 มิติ ซึ่งต้องใช้โคออร์ดิเนทถึง 3 แกน ภาพของวัตถุต่าง ๆ เหล่านี้ จะต้องถูกวาดลงบนระนาบราบเรียบ 2 มิติเพื่อให้ผู้ใช้เห็นภาพของวัตถุนั้น เช่น แสดงภาพวัตถุออกทางจอภาพ หรือเครื่องพิมพ์ภาพ ดังนั้นจึงต้องมีวิธีการสร้างภาพ 2 มิติของวัตถุที่ถูกกำหนดไว้ในโคออร์ดิเนท 3 มิติ

5.1 PROJECTION (โปรเจ็กชั่น)

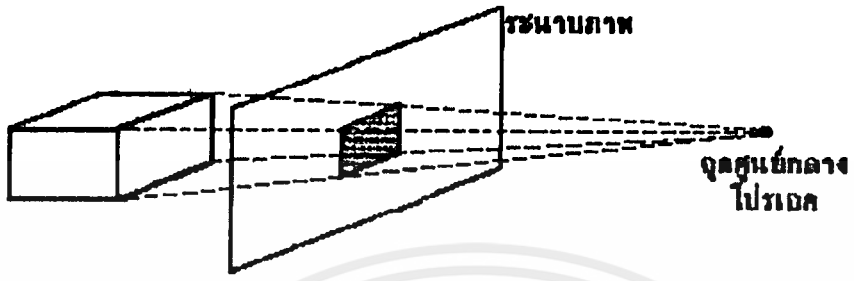
โปรเจ็กชั่น (Projection) เป็นวิธีการหนึ่งที่สามารถสร้างภาพ 2 มิติได้ โดยการฉายเงา (Project) ของวัตถุให้ไปตกลงบนระนาบหนึ่ง ที่ระนาบก็จะเกิดเป็นเงาหรือภาพ 2 มิติของวัตถุนั้น ระนาบนี้เราเรียกว่า **ระนาบภาพ** (View Plane) ดังแสดงในรูป 5.1 จะสังเกตได้ว่าภาพที่เกิดขึ้นบนระนาบนั้น ขึ้นอยู่กับการวางตัวของระนาบภาพ ทิศทางของแสงที่ฉายเงาไปยังระนาบ และองค์ประกอบอื่น ๆ อีก



รูป 5.1 โปรเจ็กชั่น

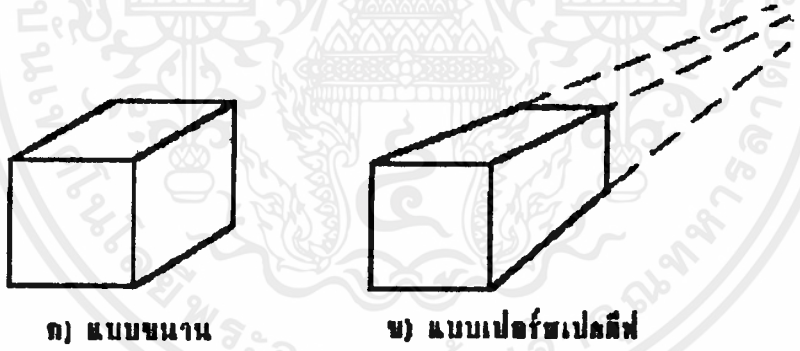
การทำโปรเจ็กชั่นนั้นมีหลายประเภท แต่ที่นิยมใช้กันมีเพียง 2 ประเภทคือ **โปรเจ็กชั่นแบบขนาน** (Parallel Projection) และ **โปรเจ็กชั่นแบบเปอร์สเป็คตีฟ** (Perspective

Projection) สำหรับโปรเจ็กชั่นแบบขนานนั้น มีลักษณะเหมือนกับในรูป 5.1 กล่าวคือ แสงของเงาที่ฉายจะพุ่งขนานกับไปหมดตรงสู่จอภาพ ส่วนโปรเจ็กชั่นแบบเปอร์สเป็คตีฟนั้น แสงของเงาทั้งหมดจะพุ่งสู่จุด ๆ หนึ่ง ไม่ขนานกัน จุดที่แสงของเงาพุ่งไปนี้เรียกว่า จุดศูนย์กลางโปรเจ็กชั่น (Center of Projection) ดังแสดงในรูป 5.2



รูป 5.2 โปรเจ็กชั่นแบบเปอร์สเป็คตีฟ

ลักษณะของภาพ ที่เกิดขึ้นจากการทำโปรเจ็กชั่นต่างประเภทกัน จะมีลักษณะต่างกันดังแสดงในรูป 5.3 เป็นการเปรียบเทียบภาพที่ได้จากการทำโปรเจ็กชั่นแบบขนาน และโปรเจ็กชั่นแบบเปอร์สเป็คตีฟ



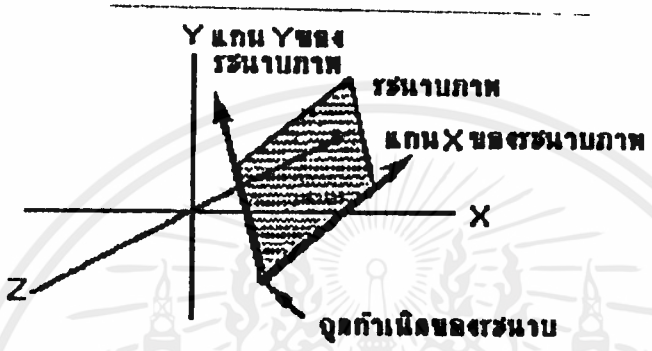
รูป 5.3 ภาพที่ได้จากการทำโปรเจ็กชั่น

5.2 พารามิเตอร์ภาพ

ดังที่กล่าวไว้แล้วว่า ภาพที่เกิดขึ้นบนระนาบภาพของวัตถุเดียวกัน อาจได้ภาพที่ต่างกันก็ได้ขึ้นอยู่กับองค์ประกอบต่าง ๆ ซึ่งมีผลต่อการสร้างภาพ เราเรียกว่าองค์ประกอบต่าง ๆ นี้ พารามิเตอร์ภาพ (Viewing Parameters)

ก่อนที่จะกล่าวถึงพารามิเตอร์ต่าง ๆ เราควรรู้จักระนาบภาพให้ละเอียดยิ่งขึ้นเสียก่อน

ระนาบภาพคือระนาบที่เราฉายเงาของวัตถุลงไป ภาพของเงาที่เกิดขึ้นบนระนาบจะเป็นภาพ 2 มิติของวัตถุ และเป็นภาพที่จะถูกวาดออกทางจอภาพหรืออุปกรณ์พิมพ์ภาพ ระนาบภาพสามารถเป็นระนาบใด ๆ ก็ได้ที่ตั้งอยู่ในโคออร์ดิเนท 3 มิติ ระบบโคออร์ดิเนท 3 มิติที่วัตถุอยู่นี้เรียกว่า **ออปเจ็กโคออร์ดิเนท** (Object Coordinates) เราจะมองระนาบภาพนี้เสมือนกับเป็นระนาบ XY ธรรมดาที่วางตัวอยู่ในระบบ มีแกน X แกน Y และจุดกำเนิดของระนาบ ดังแสดงในรูป 5.4 ระบบโคออร์ดิเนท XY ของระนาบภาพนี้เรียกว่า **โคออร์ดิเนทของระนาบภาพ** (View Plane Coordinates)

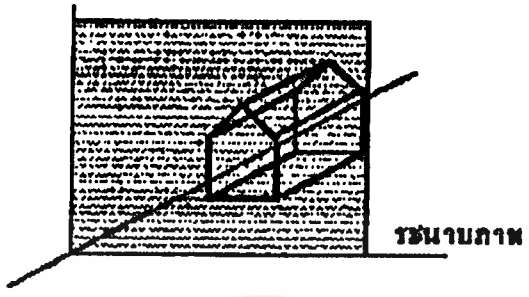


รูป 5.4 ระนาบภาพในระบบโคออร์ดิเนท 3 มิติ

เราอาจเปรียบเทียบระนาบภาพได้กับ ฟิล์มในกล้องที่เราใช้ถ่ายภาพทั่วไป วัตถุที่เราต้องการถ่ายคือวัตถุในระบบโคออร์ดิเนท ซึ่งเงาของวัตถุนั้นจะตกไปอยู่บนแผ่นฟิล์มในกล้องเกิดเป็นภาพบนฟิล์ม (ภาพบนระนาบภาพ) ขณะเดียวกัน ตำแหน่งของฟิล์มก็คือตำแหน่งที่เรายืนมองวัตถุด้วย เราสามารถย้ายตำแหน่งหรือเปลี่ยนมุมของกล้องถ่ายภาพ เพื่อให้ได้ภาพถ่ายที่ออกมามีลักษณะต่างออกไป ระนาบภาพก็เช่นกัน เราสามารถโยกย้ายและวางระนาบภาพไว้ในลักษณะใดก็ได้ที่เราต้องการ ตำแหน่งและการวางตัวของระนาบภาพต้องใช้พารามิเตอร์ 4 ตัวในการกำหนด

พารามิเตอร์ตัวแรกคือ **จุดอ้างอิงภาพ** (View Referance Point) $P_r(x_r, y_r, z_r)$ จุดนี้เป็นจุดศูนย์กลางของความสนใจในการมองภาพของเรา พารามิเตอร์ตัวที่สอง คือ **เวคเตอร์ตั้งฉากของระนาบภาพ** (View Plane Normal Vector) $N = [x_n\ y_n\ z_n]$ เป็นเวคเตอร์ที่ตั้งฉากกับระนาบภาพ ถ้าเราลากเส้นตรงจากจุดอ้างอิงภาพ ให้ขนานกับเวคเตอร์ N ไปตั้งฉากกับระนาบภาพ จุดที่เส้นตรงนี้ตัดกับระนาบภาพคือจุดกำเนิดของโคออร์ดิเนทของระนาบภาพนั้นเอง พารามิเตอร์ตัวต่อไปคือ **ระยะภาพ** (View Distance) หมายถึง ระยะห่างระหว่างจุดอ้างอิงภาพและระนาบภาพ โดยปกติ เรามักกำหนดระยะภาพให้เป็นศูนย์ ทำให้จุดอ้างอิงภาพและจุดกำเนิดของระนาบภาพเป็นจุดเดียวกัน และพารามิเตอร์ตัวสุดท้ายคือ **วิวอัพเวค**

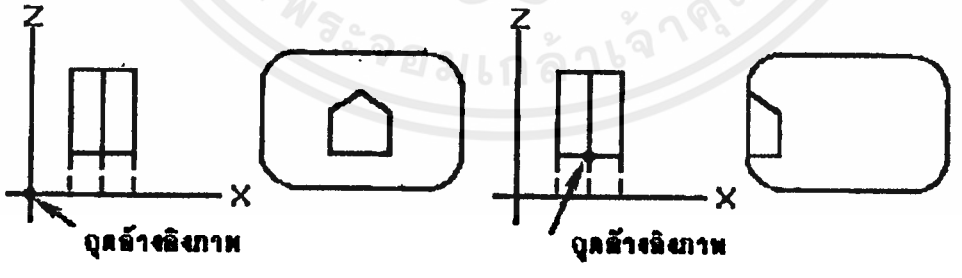
เดอร์ (View-up Vector) $V = [x_v\ y_v\ z_v]$ เป็นเวคเตอร์ที่กำหนดแนวทิศทางตั้งขึ้นของโคออร์ดิเนทของระนาบภาพ นั่นคือ เวคเตอร์ V เป็นเวคเตอร์ที่ขนานและมีทิศทางเดียวกับแกน +Y ของโคออร์ดิเนทของระนาบภาพ



รูป 5.5 ตัวอย่างวัตถุในระบบออปเจ็กโคออร์ดิเนท

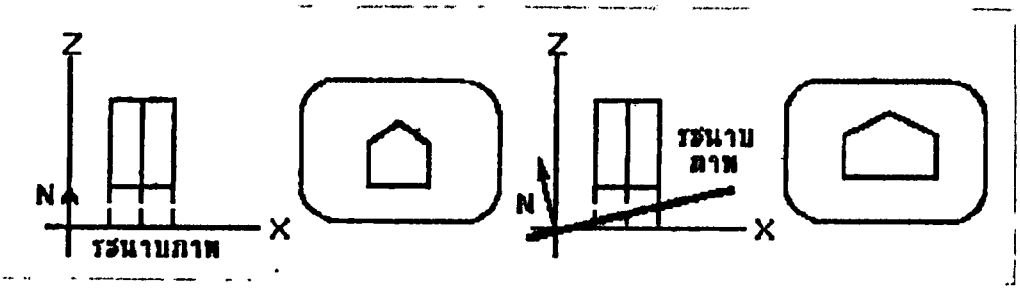
ในระบบคอมพิวเตอร์กราฟิค ผู้ใช้ต้องกำหนดพารามิเตอร์เหล่านี้ เพื่อให้ได้ลักษณะภาพที่ต้องการ การกำหนดค่าของพารามิเตอร์ที่ต่างกัน ย่อมได้ภาพที่ต่างกันออกไป สมมติว่ามีวัตถุชิ้นหนึ่งในออปเจ็กโคออร์ดิเนท เราต้องการให้เกิดภาพบนระนาบ XY นั่นคือระนาบ XY เป็นระนาบภาพ ในรูป 5.5 แสดงวัตถุและระนาบภาพดังกล่าว

การเปลี่ยนจุดอ้างอิงภาพจะทำให้ได้ภาพที่เกิดขึ้น มีการย้ายตำแหน่ง ส่วนของวัตถุที่ปรากฏที่จุดกำเนิดบนโคออร์ดิเนทของระนาบภาพจะเปลี่ยนไป ในรูป 5.6 แสดงตัวอย่างผลของการเปลี่ยนตำแหน่งของจุดอ้างอิงภาพ ภาพทางซ้ายเป็นมุมมองด้านบนของวัตถุในรูป 5.5 คือมองตรงลงมาจากแกน Y และภาพทางขวาคือ ภาพที่เกิดขึ้นบนระนาบภาพ



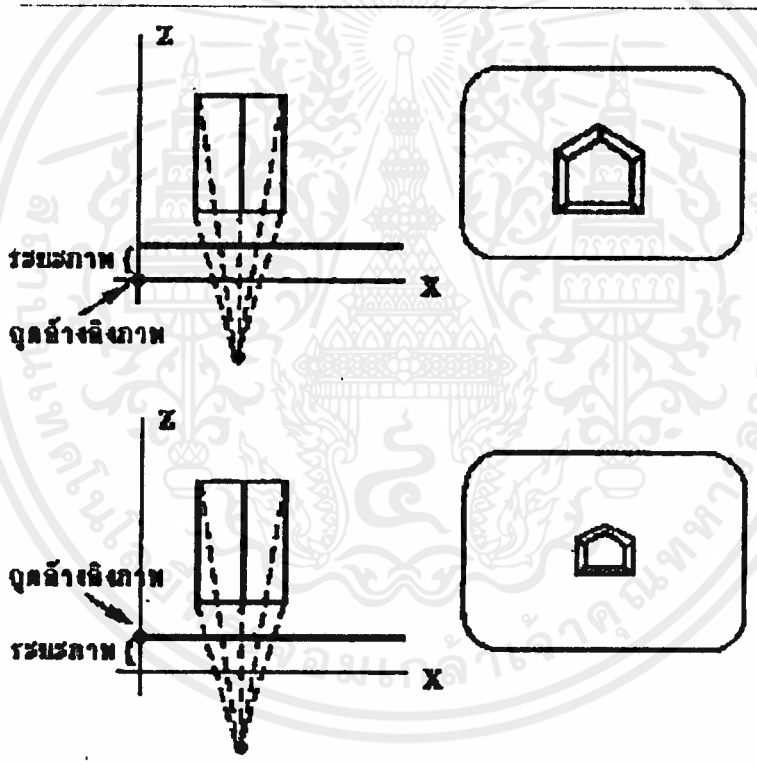
รูป 5.6 การเปลี่ยนจุดมองภาพ

ถ้าเปลี่ยนเวคเตอร์ตั้งฉากของระนาบภาพ การจัดวางระนาบในระบบจะเปลี่ยนไปรูป 5.7 แสดงตัวอย่างผลของการเปลี่ยนทิศของเวคเตอร์ตั้งฉากของระนาบภาพ



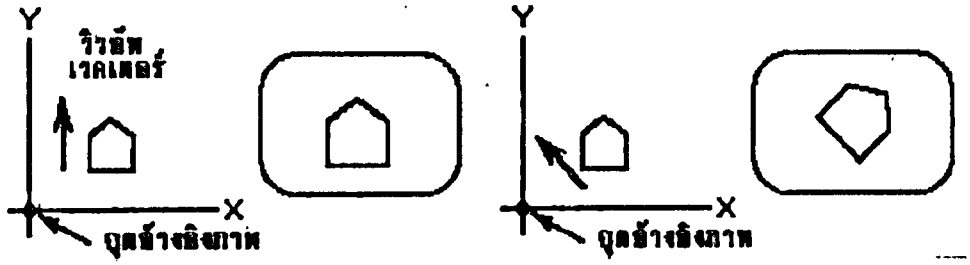
รูป 5.7 การเปลี่ยนเวคเตอร์ตั้งฉากของระนาบภาพ

ถ้าเปลี่ยนระยะภาพ จะทำให้ได้ภาพที่มีขนาดใหญ่เล็กต่างกัน ดังแสดงในรูป 5.8 จะสังเกตได้ว่า รูป 5.8 นั้นเป็นภาพจากการทำโปรเจ็กชั่นแบบเปอร์สเป็คตีฟ ถ้าเป็นการทำโปรเจ็กชั่นแบบขนาน ขนาดของภาพจะไม่ขึ้นอยู่กับระยะภาพ



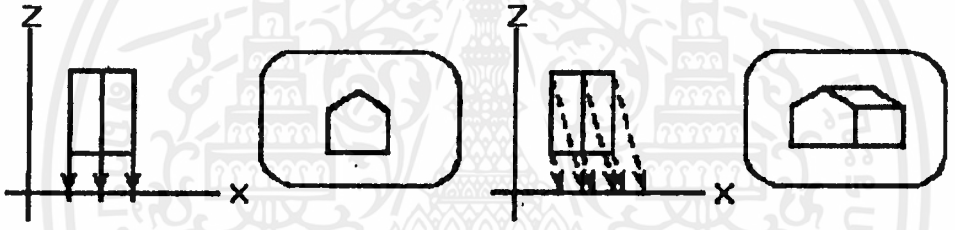
รูป 5.8 การเปลี่ยนระยะภาพสำหรับโปรเจ็กชั่นแบบเปอร์สเป็คตีฟ

การเปลี่ยนทิศทางของวิวอัฟเวคเตอร์ เปรียบได้กับการที่เราเอียงกล้องถ่ายภาพไปมานั่นเอง ทำให้ภาพที่ได้มีการเอนเอียงต่างกัน ดังแสดงในรูป 5.9



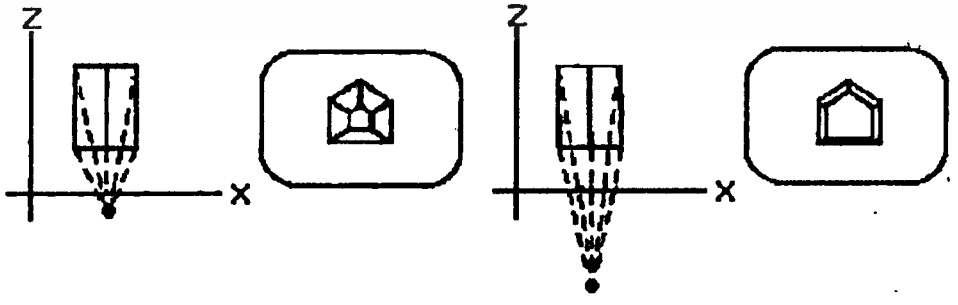
รูป 5.9 การเปลี่ยนวิวอัพเวคเตอร์

พารามิเตอร์ที่กล่าวมาเป็นพารามิเตอร์ที่เกี่ยวข้องกับการมองวัตถุ เสมือนว่าเราจะมองวัตถุในแง่มุมไหน ลักษณะใด นอกจากพารามิเตอร์ทั้ง 4 ที่กล่าวมา แล้วยังมีพารามิเตอร์ที่เกี่ยวกับโปรเจ็กชั่นของระบบอีกด้วย สำหรับโปรเจ็กชั่นแบบขนาน ต้องกำหนดทิศทางของแสงที่ฉาย หรือ ทิศทางโปรเจ็กชั่น (Projection Direction) $D_p = [x_p\ y_p\ z_p]$ ผลของการเปลี่ยนทิศทางโปรเจ็กชั่น แสดงไว้ในรูป 5.10



รูป 5.10 การเปลี่ยนทิศทางโปรเจ็กชั่น

สำหรับโปรเจ็กชั่นแบบเปอร์สเป็คตีฟ ต้องกำหนดจุดศูนย์กลางโปรเจ็กชั่นแทนทิศทางของแสง รูป 5.11 แสดงผลของการเปลี่ยนตำแหน่งของจุดศูนย์กลางโปรเจ็กชั่น นอกจากพารามิเตอร์ทั้ง 2 แล้ว ระบบคอมพิวเตอร์กราฟิคควรมีพารามิเตอร์หรือตัวแปรอีกตัวหนึ่ง ซึ่งบ่งบอกประเภทของการทำโปรเจ็กชั่นไว้ด้วยว่า เป็นแบบขนานหรือ แบบเปอร์สเป็คตีฟ



รูป 5.11 การเปลี่ยนจุดศูนย์กลางโปรเจ็กชั่น

5.3 ตัวอย่างการทำโปรเจ็กชั่น

โดยทั่วไปแล้วสำหรับการทำโปรเจ็กชั่นแบบขนาน เรามักใช้ระนาบ XY เป็นระนาบภาพ และทิศทางโปรเจ็กชั่นมีทิศทางขนานกับแนวแกน Z เงาของจุดต่าง ๆ ที่เกิดขึ้นบนระนาบภาพจะมีค่าโคออร์ดิเนท X และ Y ที่ไม่เปลี่ยนแปลง เพราะทิศทางโปรเจ็กชั่นลากตรงจากจุดต่าง ๆ ขนานแกน Z ลงมาตัดกับระนาบ XY ทำให้การทำโปรเจ็กชั่นนั้นง่ายมาก คือเพียงแค่ตัดค่าโคออร์ดิเนท Z ของจุดเหล่านั้นออก เราก็จะได้ตำแหน่งเงาของจุดนั้นบนระนาบภาพ เช่น จุด $P(x,y,z)$ จะทำให้เกิดจุด $P'(x,y)$ บนระนาบภาพ

ในบางครั้งถ้าทิศทางโปรเจ็กชั่นเปลี่ยนไปในทิศทางอื่น วิธีดังกล่าวก็ใช้งานไม่ได้ ถ้าทิศทางของโปรเจ็กชั่นเป็นไปตามทิศทางของเวคเตอร์ $[x_p\ y_p\ z_p]$ แต่ยังคงใช้ระนาบ XY เป็นระนาบภาพ จะได้ว่าจุด $P(x,y,z)$ จะมีเงาหรือทำให้เกิดจุด $P'(x',y')$ บนระนาบภาพ ซึ่ง

$$x' = x - z(x_p/z_p) \qquad (5.1)$$

$$y' = y - z(y_p/z_p) \qquad (5.2)$$

หรืออาจเขียนให้อยู่ในรูปของเมตริกซ์ได้คือ

$$[x'\ y'\ z'\ 1] = [x\ \ y\ \ z\ \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ x_p/z_p & -y_p/z_p & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5.3)$$

โดยที่ z' คือโคออร์ดิเนท Z ของจุด P' ซึ่งเราตัดทิ้ง และ z' นี้จะมีค่าเป็นศูนย์เสมอ เพราะเงาโปรเจ็กชั่นไปบนระนาบ XY (ระนาบ $Z = 0$)

สำหรับตัวอย่างของโปรเจ็กชั่นแบบเปอร์สเป็คตีฟ ถ้าเรายังคงใช้ระนาบ XY เป็นระนาบภาพเช่นเดิม และให้จุด (x_c,y_c,z_c) เป็นจุดศูนย์กลางเปอร์สเป็คตีฟ จุด $P(x,y,z)$ จะมีเงาหรือทำให้เกิดจุด $P'(x',y')$ บนระนาบภาพ ซึ่ง

$$x' = \frac{x_c z - x z_c}{z - z_c} \qquad (5.4)$$

$$y' = \frac{y_c z - y z_c}{z - z_c} \qquad (5.5)$$

หรืออาจเขียนให้อยู่ในรูปเมตริกซ์คือ

$$[x' \; y' \; z' \; 1] = [x \;\; y \;\; z \;\; 1] \begin{bmatrix} z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & 0 & 1 \\ 0 & 0 & 0 & -z_c \end{bmatrix} \qquad (5.6)$$

ในบางครั้งเราจะกำหนดจุดศูนย์กลางของโปรเจ็กชั่นอยู่ที่จุดกำเนิด (0,0,0) และให้ระนาบภาพอยู่ที่ระนาบ Z = d ในกรณีนี้จะต้องใช้เมตริกซ์ต่อไปนี้ แทนเมตริกซ์ในสมการ(5.6)

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5.7)$$

ถ้าเรานำเมตริกซ์ในสมการ (5.7) ไปแทนที่เมตริกซ์ในสมการ (5.8) จะได้ค่า 'z' เท่ากับ d เพราะเมตริกซ์ในสมการ (5.7) เป็นการทำโปรเจ็กชั่นลงบนระนาบ Z = d

บทที่ 6

การลบเส้นและพื้นผิวที่ถูกบัง

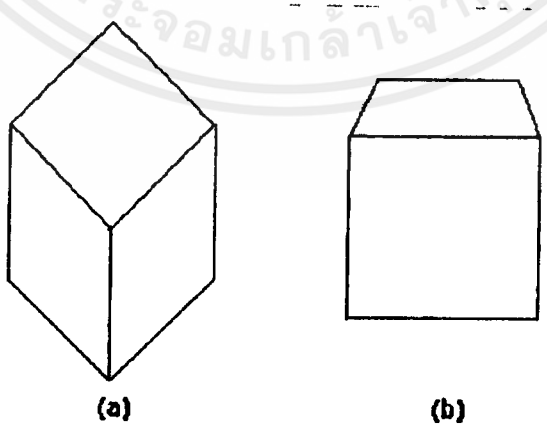
การสร้างรูป 3 มิติที่สมจริงจะต้องไม่ลากเส้นที่ถูกบัง หรือพื้นผิวที่มองไม่เห็น การกระทำเช่นนี้จะต้องพิจารณาว่าเส้นจะมองเห็นหรือมองไม่เห็น แล้วแสดงเฉพาะส่วนที่มองเห็น ส่วนที่ถูกบังหรือมองไม่เห็นไม่ต้องแสดง อัลกอริทึมในการทำจะต้องใช้เวลาในการคำนวณสูงจึงเสียเวลามากในการวาด

ก่อนที่จะพูดถึงขบวนการในการทำมี 2 สมมติฐานคือ หนึ่งวัตถุมีพื้นผิวภายนอกปิด เราใช้เส้นประกอบเป็นรูปทรงหลายเหลี่ยม (polygon) สองคือทุกพื้นผิวมีลักษณะแบนราบ หมายความว่าทุกจุดที่ล้อมรอบ polygon นั้นอยู่บนระนาบเดียวกัน

6.1 การมองวัตถุทรงเหลี่ยม 1 ก้อน

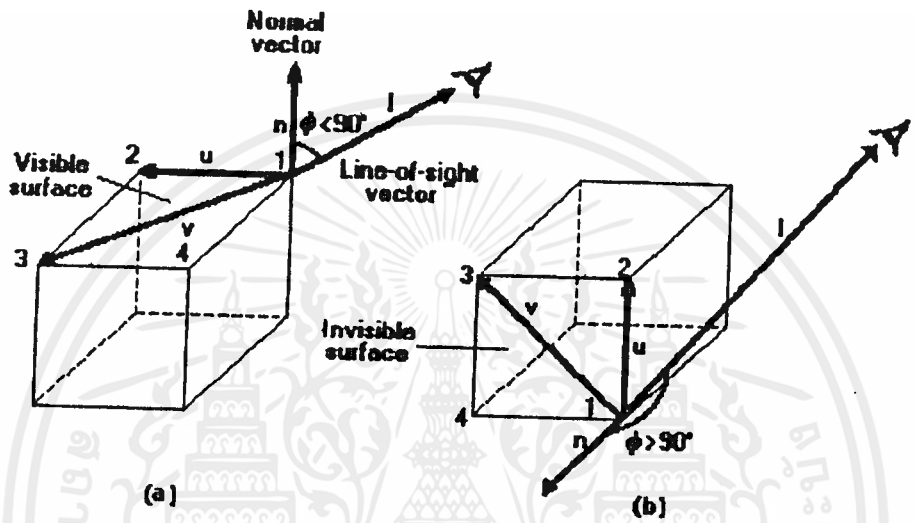
ถ้าเรามีวัตถุ 1 ก้อนลักษณะ convex (คือ วัตถุที่มีมุมแหลม วัตถุที่ เส้นต่อเชื่อมระหว่างจุดในวัตถุโดยทั้งหมดบรรจุในวัตถุ เช่น ลูกบาศก์) เราสามารถแก้ปัญหาการมองโดยคิดจากของ normals ของพื้นผิววัตถุ ซึ่งก็คือ face normal vector เป็นเวกเตอร์ที่ทำมุมฉากกับพื้นผิว

วัตถุทรงตันเช่นลูกลูกบาศก์ (ดังรูป 6.1) มี 8 vertex ประกอบด้วยเส้น 12 เส้นและ 6 พื้นผิว แต่ละพื้นผิวเรียบเป็นรูปสี่เหลี่ยม ถ้าเรามองลูกบาศก์จากจุด P เราจะมองเห็นด้านของลูกบาศก์เพียง 2 ถึง 3 ด้านเท่านั้น เพื่อเป็นการหลีกเลี่ยงการวาดพื้นผิวถูกบังจะต้องทำ visible test ในการทดสอบว่าพื้นผิวนั้นจะมองเห็นหรือไม่



รูป 6.1 จำนวนพื้นผิวของลูกบาศก์ที่สามารถมองเห็น

การทำ visible test จะใช้เวกเตอร์ 2 ตัว คือ surface-normal vector และ line-of-sight vector เราแทน surface-normal vector ด้วย n เป็นเวกเตอร์ที่ชี้ออกตั้งฉากกับพื้นผิว ส่วน line-of-sight vector แทนด้วย l เป็นเวกเตอร์ที่ลากจากจุดสายตา (viewpoint) ไปยังจุดเริ่มต้นของ surface-normal vector(ดังแสดงในรูป 6.2) เราจะคำนวณ มุม ϕ ระหว่าง n กับ l ถ้ามุมนั้นอยู่ระหว่าง 0 - 90 องศา แสดงว่าพื้นผิวนั้นมองเห็นและต้องวาดแสดงออกมา แต่ถ้ามุมมากกว่า 90 องศา พื้นผิวนั้นจะมองไม่เห็น และไม่ต้องแสดงออกมา



รูป 6.2 แสดง surface-normal และ line-og-sight vector

surface-normal vector การคำนวณ เวกเตอร์ n เราจะต้องทราบหมายเลขของแต่ละ vextor ของแต่ละพ์นผิวในทิศทางทวนเข็มนาฬิกา หลังจากนั้นกำหนดเวกเตอร์ u ซึ่งลากจากจุดที่ 1 ไปยังจุดที่ 2 และเวกเตอร์ v ลากจากจุดที่ 1 ไปยังจุดที่ 3 ผลจากการทำ cross produce u X v จะได้เวกเตอร์ n ซึ่งทิศทางชี้ออกจากพื้นผิว

$$n = u \times v$$
$$= (bf-ce,\ cd-af,\ ae-bd) \qquad (6.1)$$

ซึ่ง $u = (a,b,c)$ และ $v = (d,e,f)$

line-of-sight vector เราคำนวณจากเส้นที่ลากจากจุดสายตา P (D,θ,ϕ)ไปยังจุดใดๆ บนพื้ผิววัตถุ เพื่อความสะดวกเราเลือกจุดที่ตั้งต้นของ surface-normal vector (vertex 1)

$l = (D \sin\phi \cos\theta,\ D \sin\phi \sin\theta,\ D \cos\phi) - (x_1, y_1, z_1)$ (6.2)

เมื่อ (x_1, y_1, z_1) คือพิกัด vertex 1

visible test กำหนด dot produce ของเวกเตอร์ n และ l ดังนี้

$$n.l = (n_1, n_2, n_3).(l_1, l_2, l_3)$$
$$= (n_1 l_1 + n_2 l_2 + n_3 l_3) \quad (6.3)$$

$$n.l = |n| . |l| . \cos\phi \quad (6.4)$$

$$\phi = \arccos \left[\frac{n.l}{|n||l|} \right]$$

โดยที่ $|n|$ = ความยาวของเวกเตอร์ n

$|l|$ = ความยาวของเวกเตอร์ l

ϕ = มุมระหว่าง n กับ l

สำหรับพื้นผิวที่มองเห็น มุม ϕ จะต้องอยู่ระหว่าง 0-90 องศา นั่นคือ $n.l > 0$ แต่พื้นผิวที่ถูกบัง มุม ϕ จะอยู่ระหว่าง 90 องศา และ 180 องศา นั่นคือ $n.l < 0$

บทที่ 7

การใช้โปรแกรม SIMULATE เครื่องพับเหล็ก CNC

7.1 FUNCTION KEY

โปรแกรมนี้จะแบ่งออกเป็น 2 ส่วนคือ ส่วนที่แสดงผลออกเป็น 2 มิติ และ ส่วนที่เป็นแบบ 3 มิติ เมื่อโหลด(LOAD) PROGRAM เข้าเครื่อง เครื่องจะแสดงเมนู(MENU) ออกมาตรงส่วนหัวของจอภาพของเครื่องซึ่งเราสามารถเลือกการทำงานต่างๆได้จาก การกด KEYBORD ตาม MENU ที่เราต้องการ

ในการทำงานของโปรแกรมการจำลองการทำงานของเครื่อง CNC มี FUNCTION KEY ต่างๆ ที่สำคัญดังต่อไปนี้

F1	เป็นการแสดงลายละเอียดของ KEY ต่างๆ
F2	เป็นการ LOAD FILE (ซึ่งจะเป็น FILE ที่ทำการเก็บค่าตัวเลขในการกำหนดระยะทาง มุมต่างๆ) ที่ต้องการที่จะทำการจำลองการทำงาน ซึ่ง FILE นี้จะเป็นในลักษณะ TEXT FILE
F3	เป็นการแสดงภาพในลักษณะ 2 มิติ
F4	เป็นการแสดงภาพในลักษณะ 3 มิติ
PAGE UP	เป็นการขยายภาพให้ใหญ่ขึ้น
PAGE DOWN	เป็นการย่อภาพให้เล็กลง
UP	เลื่อนภาพขึ้น / หมุนวัตถุขึ้น
DOWN	เลื่อนภาพลง / หมุนวัตถุลง
LEFT	เลื่อนภาพทางซ้าย / หมุนวัตถุทางซ้าย
RIGHT	เลื่อนภาพทางขวา / หมุนวัตถุทางขวา
ESC	ออกจาก PROGRAM

7.2 ลำดับขั้นตอนการทำงานในการ SIMULATE โปรแกรม

เมื่อได้เข้าสู่โปรแกรมการจำลองการทำงานแล้ว ผู้ใช้จะต้องทำการกดปุ่ม F2 เพื่อจะทำการ LOAD FILE ที่จะทำ SIMULATE เมื่อกดปุ่ม F2 เครื่องจะแสดงข้อความดังนี้

ENTER FILE NAME :

เมื่อข้อความนี้ขึ้นที่หน้าจอให้ทำการพิมพ์ Driver Path แล้วต่อด้วยชื่อ File และกด Enter เช่น TEXT.FILE ที่จะทำการ SIMULATE มีชื่อว่า TEST.CNC อยู่ที่ Driver A: จะต้องพิมพ์ว่า

ENTER FILE NAME : A:TEST.CNC

หลังจาก Load File แล้วที่หน้าจอจะแสดงรูป แผ่นเหล็กหลังจากการพับเสร็จแล้วซึ่งจะแสดงเป็นในลักษณะภาพ 3 มิติ แต่ถ้ามีความต้องการที่จะให้แสดงเป็นภาพ 2 มิติ ก็ให้ทำการกด F3 ภาพที่แสดงอยู่ที่หน้าจอก็จะกลายเป็นภาพ 2 มิติ เพื่อที่จะทำให้เราเห็นภาพได้ชัดเจนยิ่งขึ้น ทั้งภาพที่เป็น 2 มิติ และ 3 มิติ นั้นสามารถที่จะทำการควบคุมภาพที่แสดงอยู่ที่หน้าจอให้เป็นในลักษณะมุมมองต่างๆ เช่น ย่อ ขยาย เลื่อนภาพ หมุนภาพ เป็นต้น ได้โดยการใช้ KEY ควบคุมต่างๆ ซึ่งจะทำให้สามารถที่จะทำการในการมองมุมมองต่างๆ ของแผ่นเหล็กได้ชัดเจนยิ่งขึ้น

เมื่อที่หน้าจอกำลังแสดงรูปอยู่ แล้วต้องการที่จะทำการ SIMULATE ใน File อื่นต่อก็สามารถทำการ Load File นั้นได้ทันที โดยทำการกดปุ่ม F2 แล้วป้อนชื่อ File ลงไปและกด Enter ภาพที่ได้ออกมาที่หน้าจอภาพก็จะเป็นการ SIMULATE ที่ได้จาก File ใหม่ ส่วนการยกเลิกในการใช้โปรแกรมสามารถที่จะกระทำได้โดยการกดปุ่ม ESC โดยในการออกของโปรแกรมนั้นจะออกมาที่ DOS PROMPT

7.3 การสร้าง TEXT FILE เพื่อเป็น SOURCE FILE ในการ SIMULATE

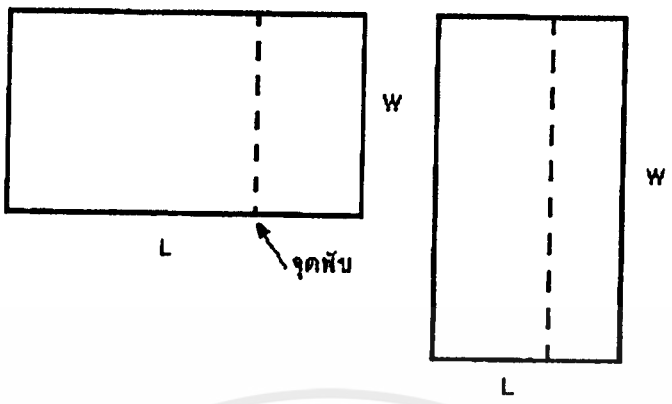
TEXT FILE สามารถที่จะทำการสร้างได้จากโปรแกรม Editor ทั่วๆไปโดยสามารถตั้งชื่อได้ไม่เกิน 8 ตัวอักษร และสามารถมีนานสกุลได้อีกไม่เกิน 3 ตัวอักษร ชื่อ และนามสกุลนี้จะใช้ในตอน Load File เพื่อที่จะทำการ SIMULATE

<u>ตัวอย่าง</u> ทำการสร้าง Source File จาก Editor แล้วทำการตั้ง ชื่อว่า TEST.CNC เมื่อต้องทำการ Load Source File นี้ ก็จะต้องใช้ชื่อ TEST.CNC

7.4 CODE ที่ใช้สำหรับการโปรแกรม SIMULATE

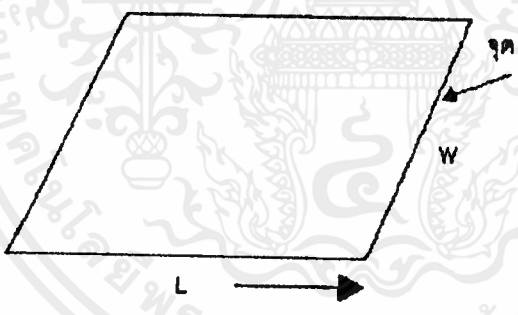
เป็น Code ที่จะต้องทำการเขียนเป็นโปรแกรมลงไปใน Source File ที่จะใช้ในการ SIMULATE เพื่อเป็นคำสั่งให้โปรแกรมจำลองการทำงานของเครื่องพับเหล็กทำงานตามคำสั่ง ซึ่ง CODE ต่างๆ ที่ใช้ในการเขียนโปรแกรมมีดังต่อไปนี้

CODE	ความหมาย
W	เป็นการบอกว่าโลหะนี้มีความกว้างเท่าไร
L	เป็นการบอกว่าโลหะนี้มีความยาวเท่าไร
X	เป็นการกำหนดตำแหน่งที่จะพับ ซึ่งจะกำหนดจากขอบนอกสุดของโลหะ
C	เป็นการกำหนดมุมที่จะทำการพับว่าเป็นมุมเท่าไร
R	เป็นการหมุนแผ่นโลหะไป 180 องศา
I	เป็นการกลับแผ่นโลหะไป 180 องศา



รูป 7.1

7.5 การกำหนดจุดตำแหน่งที่จะทำการพับ



รูป 7.2

เมื่อทำการนำแผ่นเหล็กใส่เข้าไปในเครื่องพับเหล็ก โดยการนำเอาด้านกว้างใส่เข้าหาเครื่อง ซึ่งในจุดแรกนั้นต้องกำหนดให้ตำแหน่งที่จุด O ซึ่งเป็นจุดที่อยู่ทางด้านมุมขวาสุด ของแผ่นเหล็กเมื่อทำการมองจากทางด้านข้างเป็นจุดที่จะใช้ในการอ้างอิง เมื่อมีความประสงค์ที่จะทำการพับเหล็กจากจุดที่ห่างจากตำแหน่งจุด O ไป 10 เซนติเมตร ก็จะใช้ Code X ในการ

กำหนดระยะที่ห่างจากจุดที่ทำการอ้างอิงดังนี้

X 10

แต่ถ้าต้องการกำหนดระยะที่ห่างจากจุดที่ทำการอ้างอิงเป็นระยะห่าง 25 เซนติเมตร ก็จะสามารถกำหนดระยะห่างจากจุดอ้างอิงได้ดังนี้

X 25

จะมีคำสั่งที่สามารถใช้ในการกำหนดระยะในการที่จะทำการพับเหล็กได้อีก 1 Code หรือ 1 คำสั่ง คือคำสั่ง U ซึ่งจะเป็นคำสั่งในรูปแบบของ Incremental เช่นถ้าต้องการทำการพับเหล็กจำนวน 2 จุด โดยจุดแรกกำหนดให้มีการห่างจากจุดอ้างอิงเป็นระยะ 10 เซนติเมตร และทำมุมเป็นมุม 90 องศา ส่วนจุดที่ 2 ให้มีการห่างจากจุดอ้างอิง เป็นระยะ 30 เซนติเมตร และทำมุมเป็นมุม 90 องศาเช่นเดียวกัน ซึ่งจากตัวอย่างนี้สามารถทำการกำหนดระยะทางได้ทั้ง 2 แบบ คือ กำหนดเป็นแบบ Absolute(X) หรือจะเป็นการกำหนดแบบ Incremental(U) ซึ่งสามารถทำการเขียนทั้ง 2 แบบได้ดังข้างล่างนี้

แบบ Absolute Program

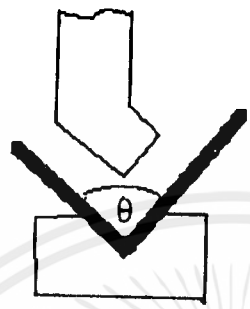
W 100	L 200
X 10	C 90
X 30	C 90

แบบ Incremental Program

W 100	L 200
X 10	C 90
U 20	C 90

จะเห็นได้ว่า Code U นั้นจะเป็นการเพิ่มค่าจากค่าของ Code X บรรทัดก่อนหน้า Code U เช่นจากในตัวอย่างข้างบน ค่าของ Code X ก่อนที่จะถึงบรรทัดของ Code U มีค่าเท่ากับ 10 เซนติเมตร แล้วให้ค่า Code U ในบรรทัดนั้นมีค่าเท่ากับ 20 เซนติเมตร ดังนั้นเมื่อทำการรวมค่าระยะเข้าด้วยกันแล้ว จะได้ค่าระยะที่ห่างจากจุดอ้างอิงมีระยะ เป็น 30 เซนติเมตร แต่ถ้าทำการใช้ Code X แทนในบรรทัดของ Code U ที่กล่าวมา จะต้องทำการให้ค่าที่ Code X มีค่าเท่ากับ 30 เซนติเมตรซึ่งจะเป็นระยะที่ทำการวัดจากจุดอ้างอิงเลย

ในการกำหนดมุมจากตัวอย่างข้างบน(หรือทุกๆ รูป)นั้นจะต้องใช้ Code C เสมอในการที่ต้องกำหนดมุม โดยทำการกำหนดว่าจุดที่จะต้องทำการพับนั้น จะทำการพับเป็นมุมกี่องศา ก็ทำการใส่เข้าไปใน Code C



รูป 7.3

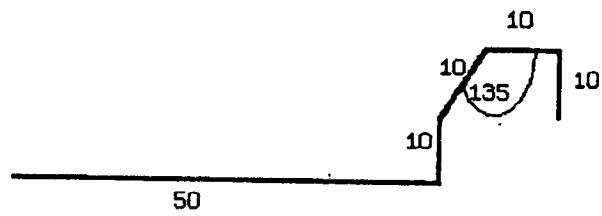
จากรูปข้างบนมุม θ คือมุมที่จะต้องทำการพับ เช่นถ้าต้องการพับเหล็กให้เป็นมุมเท่ากับ 135 องศาจะต้องทำการเขียนค่าของ Code C ดังนี้ C 135

ในการพับแผ่นเหล็กนั้น อาจจะไม่ได้มีการพับแผ่นเหล็กหน้าเดียวตลอด ดังนั้นจึงจำเป็นที่จะต้องมีการพลิกแผ่นเหล็กมาทำการพับอีกด้านหนึ่งบ้าง เพื่อที่จะให้ได้รูปแผ่นเหล็กตามต้องการ ซึ่งในการพลิกแผ่นนั้นจะต้องใช้ Code I เป็นรหัสคำสั่งของโปรแกรมที่ใช้ในการพลิกแผ่นเหล็ก สมมุติว่าต้องการที่จะทำการพลิกแผ่นไป 180 องศา(ซึ่งปกติแล้วจะต้องทำการพลิกแผ่นเหล็ก 180 องศาเสมอ) ก็จะต้องทำการเขียนค่าของ Code I ดังนี้

I 180

เมื่อทำการกำหนดการพลิกแล้วก็สามารถกำหนดระยะที่จะทำการพับต่อไป

ตัวอย่าง ในการพับแผ่นเหล็ก



รูป 7.4

เมื่อกำหนดให้แผ่นเหล็กมีขนาดกว้างเท่ากับ 50 เซนติเมตร และ ยาวเท่ากับ 90 เซนติเมตร

วิธีทำ

สามารถที่จะทำการแบ่งลำดับขั้นตอนในการพับแผ่นเหล็กได้ดังนี้

1. กำหนดขนาดความกว้าง และความยาวของแผ่นเหล็ก

W 50 L 90

2. ทำการพับแผ่นเหล็กโดยมีระยะห่างจากจุดอ้างอิงเป็นระยะ 10 เซนติเมตร โดยมีการทำมุมเป็นมุม 90 องศา

X 10 C 90

3. ทำการพับจุดที่สอง ซึ่งห่างจากการพับที่จุดแรกเป็นระยะ 10 เซนติเมตร โดยมีการทำมุมเป็นมุม 135 องศา

U 10 C 135

4. ทำการพับจุดที่สาม ซึ่งห่างจากการพับที่จุดที่สองเป็นระยะ 10 เซนติเมตร โดยมีการทำมุมเป็นมุม 135 องศา

U 10 C 135

5. ในการพับแผ่นเหล็กในจุดที่สี่นี้ จะต้องมีการทำการพลิกแผ่นเหล็ก เพื่อที่จะทำการพับแผ่นเหล็กอีกด้านหนึ่ง ซึ่งในการพลิกแผ่นเหล็กไปอีกด้านหนึ่งนั้นจะต้องทำ การพลิกเป็นมุม 180 องศา

I 180

6. ทำการพับจุดที่สี่ ซึ่งห่างจากการพับที่จุดที่สามเป็นระยะ 10 เซนติเมตร โดยมีการทำมุมเป็นมุม 90 องศา

U 10 C 90

เมื่อนำ Code จากลำดับขั้นตอนต่างๆ ของลำดับขั้นการทำงานมาทำการเขียนเป็นโปรแกรมจะได้โปรแกรมดังนี้

```
W 50      L 90
X 10      C 90
U 10      C 135
U 10      C 135
I 180
U 10      C 90
```

ในการพับแผ่นเหล็กนั้น จะมีการเริ่มทำการพับแผ่นเหล็กจากด้านนอกสุด เข้ามาทีละจุดโดยเริ่มทำการพับแผ่นเหล็กจากจุดที่ 1 ไปจนถึงจุดที่ 4 โดยไล่ตามลำดับขั้นตอนตามลำดับ

บทที่8

การทำงานของโปรแกรม

8.1 Algorithm of Program

เมื่อเริ่มต้น RUN Program โปรแกรม จะทำการ SET จอภาพให้เป็น Mode VGA 640 X 350 Pixel และ SET ค่าตัวแปรต่างๆที่ใช้ใน โปรแกรม จากนั้น จะมีการตรวจสอบ KEYBOARD อยู่ตลอดเวลา และถ้ามีการกด KEYBOARD โปรแกรม จะทำตามหน้าที่ของ KEY นั้นๆ ซึ่งมีการกำหนดไว้ในโปรแกรมแล้ว

8.2 การตรวจสอบการกด KEYBOARD

การตรวจสอบการกด KEYBOARD จะใช้วิธีอ่านค่าจาก MEMORY โดย เปรียบเทียบข้อมูลที่ ADDRESS 40:001A และ ADDRESS 40:001C ถ้าไม่เท่ากันแสดงว่ามีการกด KEY โดยค่าของ KEY ที่กดจะอยู่ที่ segment 40 offset จะถูกกำหนดที่ ADDRESS 40:001A เมื่อหลังจากอ่านข้อมูลแล้ว จะ SET ให้ค่าใน ADDRESS 40:001A มีค่าเท่ากับค่าใน ADDRESS 40:001C เพื่อให้โปรแกรมรู้ว่าอ่านข้อมูลแล้ว

8.3 การอ่านข้อมูลจาก FILE

การอ่านข้อมูลจาก FILE Source file ที่ใช้เป็น TEXT FILE ประกอบด้วย ตัวอักษร และ ตัวเลข การอ่าน FILE จะอ่านทีละ character character ที่ได้จะเปลี่ยนเป็น ตัวอักษรใหญ่ (UPPER CASE) เพื่อสะดวกในการแปลความหมาย จากนั้นจะนำ character ที่ได้ เก็บไว้ใน LINK LIST เพื่อสะดวกในการเรียกใช้ เมื่อเก็บจนหมดแล้วต่อมาจะทำการแยก ตัวอักษร และตัวเลขออกจากกัน โดย ตัวอักษร จะเก็บไว้ใน codelist[].command ส่วนตัวเลขเก็บไว้ใน codelist[].sdata โดยตัวเลขที่เก็บไว้ใน codelist[].sdata นั้นยังเป็น character ก็จะถูกเปลี่ยนให้เป็น REAL และจะเก็บไว้ในcodelist[].ndata หลังจากทำจนหมด FILE แล้ว จะ RETURN จำนวน ARRAY ที่ใช้ทั้งหมดออกมา

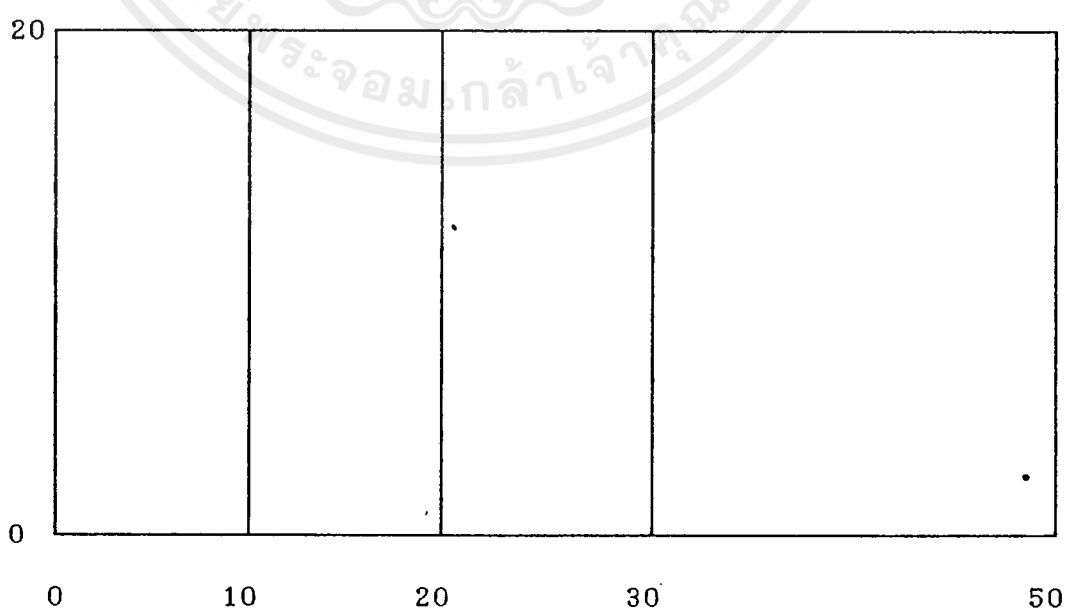
8.4 การสร้าง Object

หลังจากอ่านข้อมูลจาก FILE แล้วจากนั้นจะเปลี่ยนข้อมูลที่มีทั้ง Absolute และ Incremental ให้เป็น Absolute ทั้งหมด ต่อจากนั้น จะสร้าง Object แบบ 2 มิติ โดยมีพิกัด คือ x1,z1,x2,z2 จำนวน Object จะขึ้นอยู่กับจำนวนมุมที่จะพับ ถ้ามีจุดพับมาก ก็จะมีObject มาก หรือจำนวน Object = จำนวนจุดพับ + 1

การสร้าง Object สร้างจาก Code X,L,W Code L จะกำหนดความยาวทั้งหมดของ Object ส่วน W จะกำหนดความกว้าง Code X จะกำหนดความกว้างของ Object แต่ละ Object เช่น X 40 Object จะเริ่มต้นต่อจาก Object ก่อนหน้านี้และสิ้นสุดที่ 40 ถ้าเป็น Object แรกจะเริ่มต้นที่ 0 และสิ้นสุดที่ 40 และจะสร้างจนครบทุก Object เช่น Code ที่อ่านได้เป็น

```
L 50  w 20
X 10  C 90
X 20  C 90
X 30  C 90
```

จะได้ Object ดังนี้



8.5 การจำลองการพับ

หลังจากการสร้าง Object แล้ว เมื่อต้องการดูขั้นตอนการพับให้กด F5 โปรแกรมจะจำลองการพับทีละ Step โดยการอ่าน Code ที่ได้จากการอ่าน FILE โดำจะใช้ Code 3 Code ดังนี้คือ

X จะทำการเลื่อน Object เข้าไปยังจุดที่จะพับ การเลื่อนจะใช้วิธีการย้าย (Translation)

c จะทำการพับโดยมุมที่ได้จะกำหนดไว้ต่อจาก Code การพับจะเป็นการ Rotate ทางแกน Y โดยจะ Rotate Object ทางขวา ทวนเข็มนาฬิกา และ Rotate Object ทางซ้าย ตามเข็มนาฬิกา

I จะเป็นการหมุน Object ทุก Object ทางแกน X ไปเป็นมุม 180 องศา

8.6 การแสดงภาพ

การแสดงภาพแบบ 2 มิติ จะแสดง Object แบบ 2 มิติ โดยจะใช้คำสั่ง LINE (X1,Z1,X2,Z2) ทีละ Object จนครบทุก Object

การแสดงภาพแบบ 2 มิติ โดยจะใช้คำสั่ง MOVETO3D(X,Y,Z)ก่อนแล้วจึงใช้คำสั่ง LINETO3D(X1,Y1,Z1,X2,Y2,Z2) ทีละ Object จนครบทุก Object

กิตติกรรมประกาศ

โครงงานงานนี้สามารถสำเร็จลุล่วงได้ดีโดยมี อาจารย์ ภากร หุตะสังกาศ อาจารย์ประจำภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม เป็นอาจารย์ที่ปรึกษา โดยท่านได้เสียสละเวลาอันมีค่าให้การดูแลและคำแนะนำที่ดีมาโดยตลอด จึงขอขอบคุณมา ณ โอกาสนี้ด้วย

หนังสืออ้างอิง

Blinn, J.F.,and M.E Newell, Computer Graphics, August 1987

Brodlie, K.W., Mathematical Methode in Computer Graphics and Design, Academic Press, London(edited),1980

Chan S.Park, Interactive Microcomputer Graphics, Addison Wesley, September 1985

Edward Angel, Computer Graphics, Addison Wesley, June 1990

Harrington, S., Computer graphics : A Programming Approach, McGraw-Hill, New York, 1993

```
unit g3d;
interface
uses graph,crt,bgidriv,bgifont;

var   phi,theta,d,xp,yp,zp : real;

procedure putpixel3d(u,v,w,color:integer);
procedure line3d(u1,v1,w1,u2,v2,w2:integer);
procedure lineto3d(u,v,w:integer);
procedure moveto3d(u,v,w:integer);
procedure rotate_x(var x,y,z:real; deg:real);
procedure rotate_y(var x,y,z:real; deg:real);
procedure rotate_z(var x,y,z:real; deg:real);
procedure shift(var u,v,w:integer; tx,ty,tz:integer);
procedure trans(u,v,w:integer;var x,y:integer);

implementation

procedure trans(u,v,w:integer;var x,y:integer);
const
     vd  = 40;
     tx  = 320;
     ty  = 175;
     scf = 1;
     vx  = 100;
     vy  = 100;
     s   = 5;
var
   xe,ye,ze,xs,ys,zs,theta0,phi0,sn1,sn2,cn1,cn2 : real;
begin
     theta0 := theta*pi/180;
     phi0   := phi*pi/180;
     sn1 := sin(theta0);
     sn2 := sin(phi0);
     cn1 := cos(theta0);
     cn2 := cos(phi0);
     xe    := -u*sn1/s + v*cn1/s;
     ye    := -u*cn1*cn2/s - v*sn1*cn2/s + w*sn2/s;
     ze    := -u*sn2*cn1/s - v*sn2*sn1/s - w*cn2/s + d;
     xs    := (vd)*(xe/ze);
     ys    := (vd)*(ye/ze);
     xs    := xs*vx + tx;
     xs    := scf*xs;
     ys    := -ys*vy + ty;
     x     := round(xs); y  := round(ys);
end;

procedure putpixel3d(u,v,w,color:integer);
var x,y : integer;
begin
     trans(u,v,w,x,y);
     putpixel(x,y,color);
end;

procedure line3d(u1,v1,w1,u2,v2,w2:integer);
var x,y,x1,y1   : integer;
begin
     trans(u1,v1,w1,x,y);
     trans(u2,v2,w2,x1,y1);
```

```
     line(x,y,x1,y1);
end;

procedure lineto3d(u,v,w:integer);
var x,y   : integer;
begin
     trans(u,v,w,x,y);
     lineto(x,y);
end;

procedure moveto3d(u,v,w:integer);
var x,y   : integer;
begin
     trans(u,v,w,x,y);
     moveto(x,y);
end;

procedure rotate_z(var x,y,z:real; deg:real);
var u,v,w   : real;
    radius  : real;
begin
     radius := deg * pi /180;
     u := x;
     v := y;
     x := u*cos(radius)-v*sin(radius);
     y := u*sin(radius)+v*cos(radius);
end;

procedure rotate_x(var x,y,z:real; deg:real);
var u,v,w   : real;
    radius  : real;
begin
     radius := deg * pi /180;
     v := y;
     w := z;
     y := v*cos(radius)-w*sin(radius);
     z := v*sin(radius)+w*cos(radius);
end;

procedure rotate_y(var x,y,z:real; deg:real);
var u,v,w   : real;
    radius  : real;
begin
     radius := deg * pi /180;
     u := x;
     w := z;
     x := u*cos(radius)+w*sin(radius);
     z := w*cos(radius)-u*sin(radius);
end;

procedure shift(var u,v,w:integer; tx,ty,tz:integer);
begin
     u := u + tx;
     v := v + ty;
     w := w + tz;
end;

begin
end.
```

```
unit simm1;
interface
uses crt,graph;
type
    source = record
           command : char;
           n_data  : real;
           s_data  : string;
           t_data  : string;
           end;

   point2d  = record
            x1,x2 : real;
            z1,z2 : real;
            u1,u2 : real;
            end;

    format = array[0..100] of source;

var
   code1          : format;
   obj            : array[-1..100] of point2d;
   wide,long      : real;

function readfile:integer;

implementation

function readfile:integer;
var
   ptr              : text;
   buffer           : char;
   code,count       : integer;
   flag             : integer;
   filename,buffer1: string;
   f_error          : word;

function fileexit : word;
begin
     setvisualpage(0);
     setactivepage(0);
     setfillstyle(solidfill,green);
     bar(190,190,550,220);
     rectangle(192,192,548,218);
     outtextxy(200,200,'Enter source file : ');
     gotoxy(48,15);
     readln(filename);
     {$I-}
     assign(ptr,filename);
     reset(ptr);
     {$I+}
     fileexit := ioresult;
     if filename = '' then fileexit := 1;
end;

begin
     repeat
           f_error := fileexit;
           if (f_error <> 0) then
```

```
                begin
                     setfillstyle(solidfill,blue);
                     bar(190,190,550,220);
                     rectangle(192,192,548,218);
                     outtextxy(220,200,
                     ' File not found    Esc to Cancel ');
                     buffer :=  readkey;
                     if buffer = #27 then
                        f_error := 10;
                     writeln;
                end;
     until (f_error = 0) or (f_error = 10) ;
     count := -1;
     if f_error = 0 then
     begin
        while not eof(ptr) do
              begin
                   read(ptr,buffer);
                   buffer := upcase(buffer);
                   case buffer of
                        'A'..'Z'      : flag := 1;
                        '0'..'9','.' : flag := 2;
                   else flag := 0;
                   end;
                   if flag = 1 then
                   begin
                        val(code1[count].s_data,code1[count].n_data,code)
                        count := count + 1;
                        code1[count].command := buffer;
                        code1[count].s_data  := '';
                   end;
                   if flag = 2 then code1[count].s_data :=
                     concat(code1[count].s_data,buffer);
              end;
        val(code1[count].s_data,code1[count].n_data,code);
        close(ptr);
     end;
     for code := 0 to count do
         code1[code].t_data :=
         concat(code1[code].command,' ',code1[code].s_data);
     readfile := count;
end;

begin
     obj[-1].x1 := 0;
     obj[-1].x2 := 0;
end.
```

```
    program simm2;
uses crt,graph,g3d,simm1,bgidriv,bgifont;
type
   point3d  = record
              x1,x2,x3,x4 : integer;
              y1,y2,y3,y4 : integer;
              z1,z2,z3,z4 : integer;
              end;

var page  : integer;
    dx,dz : integer;
    n_obj : integer;
    i     : integer;
    diman : integer;
    obj3d         : array[0..100] of point3d;
    com_code      : string;
    s1,s2         : integer;

procedure rotatex(angle:integer);
var
   loop : integer;
   buf  : real;
begin
     buf := 0;
     for loop := 0 to n_obj do
     begin
          rotate_x(obj[loop].x1,buf,obj[loop].z1,angle);
          rotate_x(obj[loop].x2,buf,obj[loop].z2,angle);
     end;
end;

procedure rotatey(angle:integer);
var
   loop : integer;
   buf  : real;
begin
     buf := 0;
     for loop := 0 to n_obj do
     begin
          rotate_y(obj[loop].x1,buf,obj[loop].z1,angle);
          rotate_y(obj[loop].x2,buf,obj[loop].z2,angle);
     end;
end;

procedure rotatez(angle:integer);
var
   loop : integer;
   buf  : real;
begin
     buf := 0;
     for loop := 0 to n_obj do
     begin
          rotate_z(obj[loop].x1,buf,obj[loop].z1,angle);
          rotate_z(obj[loop].x2,buf,obj[loop].z2,angle);
     end;
end;

procedure shift_org(dx : real; i : integer);
var co    : integer;
```

```
    loop  : integer;
    tempx,tempz  : real;
    angle,buf    : real;
begin
     loop := 0;
     if i = 0 then
     begin
          while obj[loop].u2 <> dx do
              loop := loop + 1;
          tempx := obj[loop].x2;
          tempz := obj[loop].z2;
     end
     else
     begin
          while obj[loop].u1 <> dx do
              loop := loop + 1;
          tempx := obj[loop].x1;
          tempz := obj[loop].z1;
     end;
     for co  := 0 to n_obj do
         begin
              obj[co].x1 := obj[co].x1 - tempx;
              obj[co].x2 := obj[co].x2 - tempx;
              obj[co].z1 := obj[co].z1 - tempz;
              obj[co].z2 := obj[co].z2 - tempz;
         end;
     if i = 1 then
     begin
          if obj[loop].z2 <> 0 then
          begin
            if obj[loop].x2 <> 0 then
              angle := arctan(obj[loop].z2/obj[loop].x2)
            else
              angle := pi/2;
            angle := angle * 180/pi;
            for co := 0 to n_obj do
              begin
                  rotate_y(obj[co].x1,buf,obj[co].z1,angle);
                  rotate_y(obj[co].x2,buf,obj[co].z2,angle);
              end;
          end;
     end
     else
     begin
          if obj[loop].z1 <> 0 then
          begin
            if obj[loop].x1 <> 0 then
              angle := arctan(obj[loop].z1/obj[loop].x1)
            else
              angle := pi/2;
            angle := angle * 180/pi;
            for co := 0 to n_obj do
              begin
                   rotate_y(obj[co].x1,buf,obj[co].z1,angle);
                   rotate_y(obj[co].x2,buf,obj[co].z2,angle);
              end;
          end;
     end;
end;
```

```
procedure rotate_c(dl : real; dc : real; i : integer);
var  loop,count   : integer;
     buf,angle    : real;
begin
     loop := 0;
     if i = 0 then
     begin
          while obj[loop].u2<> dl do
              loop := loop + 1;
          for count := 0 to loop do
          begin
              rotate_y(obj[count].x1,buf,obj[count].z1,dc/2);
              rotate_y(obj[count].x2,buf,obj[count].z2,dc/2);
          end;
          for count := loop+1 to n_obj do
          begin
              rotate_y(obj[count].x1,buf,obj[count].z1,-dc/2);
              rotate_y(obj[count].x2,buf,obj[count].z2,-dc/2);
          end;
     end
     else
     begin
          while obj[loop].u1 <> dl do
              loop := loop + 1;
          for count := loop-1 downto 0 do
          begin
              rotate_y(obj[count].x1,buf,obj[count].z1,-dc/2);
              rotate_y(obj[count].x2,buf,obj[count].z2,-dc/2);
          end;
          for count := n_obj downto loop do
          begin
              rotate_y(obj[count].x1,buf,obj[count].z1,dc/2);
              rotate_y(obj[count].x2,buf,obj[count].z2,dc/2);
          end;
     end
end;


procedure makeobj3d;
var
   c : integer;
begin
     for c := 0 to n_obj do
          begin
              obj3d[c].x1 := round(obj[c].x1);
              obj3d[c].x2 := round(obj[c].x2);
              obj3d[c].x3 := round(obj[c].x2);
              obj3d[c].x4 := round(obj[c].x1);
              obj3d[c].y1 := -round(wide/2);
              obj3d[c].y2 := -round(wide/2);
              obj3d[c].y3 := round(wide/2);
              obj3d[c].y4 := round(wide/2);
              obj3d[c].z1 := round(obj[c].z1);
              obj3d[c].z2 := round(obj[c].z2);
              obj3d[c].z3 := round(obj[c].z2);
              obj3d[c].z4 := round(obj[c].z1);
          end;
end;
```

```
function make_object(num : integer):integer;
var
   flag,start  : real;
   final,old   : real;
   loop,step   : integer;
   nobj        : integer;
   ch          : char;

procedure sort; {bubble sort}
var
   tempx,templ : real;
   a,b         : integer;
begin
     for a := 1 to nobj do
         for b := nobj downto a-1 do
         if obj[b-1].x1 < obj[b].x1 then
         begin
              tempx := obj[b-1].x1;
              templ := obj[b-1].u1;
              obj[b-1].x1 := obj[b].x1;
              obj[b-1].u1 := obj[b].u1;
              obj[b].x1 := tempx;
              obj[b].u1 := templ;
         end;
end;

procedure sort1; {bubble sort}
var
   tempx,templ : real;
   a,b         : integer;
begin
     for a := 1 to nobj do
         for b := nobj downto a-1 do
         if obj[b-1].x2 < obj[b].x2 then
         begin
              tempx := obj[b-1].x2;
              templ := obj[b-1].u2;
              obj[b-1].x2 := obj[b].x2;
              obj[b-1].u2 := obj[b].u2;
              obj[b].x2 := tempx;
              obj[b].u2 := templ;
         end;
end;

procedure command_r;
begin
     if flag = 0 then flag := 1
        else flag := 0;
end;

procedure command_x;
var temp : real;
begin
     if flag = 0 then
     begin
          obj[nobj].x1 := -start;
          obj[nobj].x2 := -code1[loop].n_data;
          obj[nobj].u1 := code1[loop].n_data;
```

```
          obj[nobj].u2 := code1[loop].n_data;
          start := code1[loop].n_data;
     end
     else
     begin
          temp := code1[loop].n_data;
          obj[nobj].x2 := -final;
          code1[loop].n_data := final - code1[loop].n_data+old;
          obj[nobj].x1 := -code1[loop].n_data;
          final := code1[loop].n_data;
          obj[nobj].u1  := final;
          obj[nobj].u2  := final;
          old := temp;
     end;
     obj[nobj].z1 := 0;
     obj[nobj].z2 := 0;
     nobj := nobj + 1;
end;

procedure command_l;
begin
     long := code1[loop].n_data;
     final := long;
end;

begin
     flag := 0; start := 0;
     nobj := 0; old := 0;
     step := 0;
     for loop := 0 to num do
     begin
          ch := code1[loop].command;
          case ch of
               'X' : command_x;
               'R' : command_r;
               'L' : command_l;
               'W' : wide := code1[loop].n_data;
          end;
     end;
     obj[nobj].x1 := -start;
     obj[nobj].x2 := -final;
     obj[nobj].z1 := 0;
     obj[nobj].z2 := 0;
     obj[nobj].u1 := 0;
     obj[nobj].u2 := 0;
     sort;sort1;
     make_object := nobj;
end;

procedure plate2d;
var
    c           : integer;
    x1,x2,z1,z2 : integer;
    kx,ky       : real;
begin
     if n_obj > 0 then
     if com_code <> '' then
     begin
          setfillstyle(solidfill,green);
```

```
          bar(0,25,100,40);
          outtextxy(10,30,com_code);
     end;
     setcolor(brown);
     line(320,0,320,349);
     line(0,175,639,175);
     setcolor(white);
     if n_obj > 0 then
     for c := 0 to n_obj do
     begin
          x1 := round(obj[c].x1);
          x2 := round(obj[c].x2);
          z1 := round(obj[c].z1);
          z2 := round(obj[c].z2);
          setcolor(white);
          kx := 64/23;
          ky := 35/15;
          x1 := round(x1*kx*D/500)+320;
          x2 := round(x2*kx*D/500)+320;
          z1 := round(z1*ky*D/500)+175;
          z2 := round(z2*ky*D/500)+175;
          line(x1-Dx,z1+Dz,x2-Dx,z2+Dz);
     end;
end;

procedure plate;
var c : integer;
begin
     setcolor(white);
     if n_obj > 0 then
     for c := 0 to n_obj do
         begin
              moveto3d(obj3d[c].x1,obj3d[c].y1,obj3d[c].z1);
              lineto3d(obj3d[c].x2,obj3d[c].y2,obj3d[c].z2);
              lineto3d(obj3d[c].x3,obj3d[c].y3,obj3d[c].z3);
              lineto3d(obj3d[c].x4,obj3d[c].y4,obj3d[c].z4);
              lineto3d(obj3d[c].x1,obj3d[c].y1,obj3d[c].z1);
         end;
end;

procedure show_t;
var loop,ed : integer;
    x,y     : integer;
begin
     if s1 > 2 then
     begin
          x := 10;
          y := 45;
          ed := s1 + 6;
          if ed > s2 then ed := s2;
             for loop := s1 to ed do
             begin
                  outtextxy(x,y,code1[loop].t_data);
                  y := y+20;
             end;
          if com_code <> '' then
          begin
               setfillstyle(solidfill,green);
               bar(0,20,100,35);
```

```
                outtextxy(10,25,com_code);
            end;
      end;
end;

procedure display1;
var  old  : integer;
begin
        old := page;
        if page = 0 then page := 1
           else page := 0;
        setvisualpage(old);
        setactivepage(page);
        cleardevice;
        setcolor(cyan);
        rectangle(0,0,639,349);
        outtextxy(5,340,'F1 HELP');
        show_t;
        if diman = 2 then plate
        else plate2d;
end;

procedure code_r(dx:real);
var
   angle,buf : real;
   co,loop   : integer;
begin
     loop := 0;
     if i = 0 then
        while obj[loop].u2 <> dx do
              loop := loop + 1
     else
        while obj[loop].u1 <> dx do
              loop := loop + 1;
     if obj[loop].z1 <> 0 then
        begin
             if obj[loop].x1 <> 0 then
               angle := arctan(obj[loop].z1/obj[loop].x1)
             else
                angle := pi/2;
             angle := angle * 180/pi;
             for co := 0 to n_obj do
                begin
                     rotate_y(obj[co].x1,buf,obj[co].z1,-angle);
                     rotate_y(obj[co].x2,buf,obj[co].z2,-angle);
                end;
        end;
     rotatez(180);
     if i = 0 then i := 1
        else i := 0;
end;

procedure sequen(loop:integer);
var
   ch    : char;
begin
     com_code := code1[loop].t_data;
     ch := code1[loop].command;
     case ch of
```

```
          'X' : shift_org(code1[loop].n_data,i);
          'C' : rotate_c(code1[loop-1].n_data,180-code1[loop].n_data,i);
          'I' : rotatex(180);
          'R' : code_r(code1[loop-2].n_data);
     end;
     makeobj3d;
     display1;display1;
end;

procedure fit;
var temx,temy    : integer;
    tenx,teny    : integer;

procedure fit1;
var c            : integer;
    tx1,ty1      : integer;
    tx2,ty2      : integer;
    tx3,ty3      : integer;
    tx4,ty4      : integer;

begin
     temx := 0;tenx := 0;
     temy := 0;teny := 0;
     if n_obj > 0 then
     for c := 0 to n_obj do
         begin
              trans(obj3d[c].x1,obj3d[c].y1,obj3d[c].z1,tx1,ty1);
              trans(obj3d[c].x2,obj3d[c].y2,obj3d[c].z2,tx2,ty2);
              trans(obj3d[c].x3,obj3d[c].y3,obj3d[c].z3,tx3,ty3);
              trans(obj3d[c].x4,obj3d[c].y4,obj3d[c].z4,tx4,ty4);

              if tx1 > temx then temx := tx1;
              if tx2 > temx then temx := tx2;
              if tx3 > temx then temx := tx3;
              if tx4 > temx then temx := tx4;
              if ty1 > temy then temy := ty1;
              if ty2 > temy then temy := ty2;
              if ty3 > temy then temy := ty3;
              if ty4 > temy then temy := ty4;

              if tx1 < tenx then tenx := tx1;
              if tx2 < tenx then tenx := tx2;
              if tx3 < temx then tenx := tx3;
              if tx4 < tenx then tenx := tx4;
              if ty1 < teny then teny := ty1;
              if ty2 < teny then teny := ty2;
              if ty3 < teny then teny := ty3;
              if ty4 < teny then teny := ty4;
         end;
end;
var ready                  : integer;
    d1,d2,d3,d4            : real;
    td1,td2                : real;
    old_d                  : real;
    slope1,slope2,count    : integer;
begin
     ready   := 0;
     d1      := D; d2      := D;
     d3      := D; d4      := D;
```

```
     slope1  := 0; count   := 0;
     repeat
           old_d := D;
           fit1;
           if temy < 300 then d1 := d1 - 10
              else if temy > 350 then d1 := d1 + 10;
           if temx < 600 then d2 := d2 - 10
              else if temx > 640 then d2 := d2 + 10;
           if teny < 0 then d3 := d3 + 10
              else d3 := d3 - 10;
           if tenx < 0 then d4 := d4 + 10
              else d4 := d4 - 10;
           if d1 > d2 then td1 := d1
              else td1 := d2;
           if d3 > d4 then td2 := d3
              else td2 := d4;
           if td1 > td2 then D := td1
              else D := td2;
           if old_d > D then slope1 := -1
              else if old_d < D then slope1 := 1
                   else slope1 := 0;
           if count = 0  then slope2 := slope1;
           if (slope1 <> slope2) or
              ((slope1 = 0) and (slope2 = 0)) then ready := 1;
           d1 := D; d2 := D;
           d3 := D; d4 := D;
           count := count + 1;
     until ready = 1;
end;

function getkey:integer;
var
   n : byte;
begin
     n := mem[$40:memw[$40:$001a]+1];
     memw[$40:$001a] := memw[$40:$001c];
     getkey := n;
end;

function exit_ok : integer;
var temp : integer;
begin
     setvisualpage(0);
     setactivepage(0);
     setfillstyle(solidfill,green);
     bar(190,190,550,220);
     rectangle(194,194,546,216);
     outtextxy(310,200,'EXIT Yes OR No ');
     repeat
     until keypressed;
     temp := getkey;
     if temp = 21 then
        exit_ok := -1
     else
        exit_ok := 0;
end;

procedure about;
begin
```

```
     setfillstyle(solidfill,green);
     bar(213,116,426,232);
     rectangle(218,121,421,227);
     outtextxy(250,150,' SIMULATE  CNC');
     outtextxy(250,170,'  VERSION 1.0');
     outtextxy(250,190,'     KMITL');
end;

procedure onhelp;
var ch : char;
begin
     setvisualpage(0);
     setactivepage(0);
     setfillstyle(solidfill,green);
     bar(10,22,200,320);
     rectangle(12,24,198,318);
     outtextxy(10,30, ' UP         CURSOR UP    ');
     outtextxy(10,50, ' DOWN       CURSOR DOWN  ');
     outtextxy(10,70, ' LEFT       CURSOR LEFT  ');
     outtextxy(10,90, ' RIGHT      CURSOR RIGHT ');
     outtextxy(10,110,' ZOOM IN    PAGE UP      ');
     outtextxy(10,130,' ZOOM OUT   PAGE DOWN    ');
     outtextxy(10,150,' F1         HELP         ');
     outtextxy(10,170,' F2         FILE         ');
     outtextxy(10,190,' F3         2 DIMENSION  ');
     outtextxy(10,210,' F4         3 DIMENSION  ');
     outtextxy(10,230,' F5         ROTATE       ');
     outtextxy(10,250,'                         ');
     outtextxy(10,270,' SPACEBAR   STEP         ');
     outtextxy(10,290,' QUIT       ESC          ');
     ch := readkey;
end;

var
   count,driver,mode        :  integer;
   head,tail,loop,step,ch   :  integer;

begin
     driver := VGA ; mode := VGAMED;
     Dx := 0; Dz := 0; D := 500; page := 0;i := 0; diman := 0;
     theta := 100; phi := 252; ch := 0;n_obj := -1;s1 := 0; s2 := 0;
     loop := 0; count := 0;
     ch := registerBGIdriver(@EGAVGADriverProc);
     initgraph(driver,mode,'');
     setviewport(0,0,639,349,true);
     about;
     readln;
     clearviewport;
     display1;display1;
     repeat
           head := memw[$40:$001a];
           tail := memw[$40:$001c];
           if head <> tail then
           begin
                ch := getkey;
                if ch = 75 then
                   begin THETA := THETA+10; DX := DX+10; end;
                if ch = 77 then
                   begin THETA := THETA-10; DX := DX-10; end;
```

```
            if ch = 72 then
               begin PHI := PHI+10; DZ := DZ-10; end;
            if ch = 80 then
               begin PHI := PHI-10; DZ := DZ+10; end;
            if ch = 73 then D := D+20;
            if ch = 81 then D := D-20;
            if ch = 59 then  onhelp;
            if ch = 60 then
               begin
                    count := readfile;
                    com_code := '';
                    if count > 0 then
                    begin
                         n_obj := make_object(count);
                         makeobj3d;
                         loop := 2;
                         s2 := count;
                    end;
               end;
            if ch = 61 then
               begin diman := 1;dx := 0; dz := 0; end;
            if ch = 62 then
               begin
                    diman := 2;
                    if n_obj > 0 then fit ;
               end;
            if ch = 57 then
               if loop <= count then
                 begin
                      sequen(loop);
                      loop := loop+1;
                      s1   := loop;
                 end
               else
               begin
                    com_code := '';
                    s1 := 0; s2 := 0;
               end;
            if ch = 63 then
            begin
                 rotatey(10);
                 makeobj3d;
            end;
            if ch = 1  then ch := exit_ok;
            display1;
            display1;
         end;
      until ch = -1;
      closegraph;
end.
```