ปีการศึกษา 2530

ROBOT ARM: SIMULATE

โดย

นายวิทยากร ชินโย
นายมานพ แซ่คู

อาจารย์ที่ปรึกษา
อาจารย์กิตตินันท์ รัตนธำรง

ปริญญานิพนธ์ปีการศึกษา 2530

ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าลาดกระบัง

เรื่อง ROBOT ARM SIMULATE

ผู้จัดทำ

1. นายวิทยากร ชินโย
2. นายมานพ แซ่คู

.....อาจารย์ที่ปรึกษา
(อาจารย์กิตตินันท์ รัตนธำรง)

.....อาจารย์ที่ปรึกษา
( )

.....อาจารย์ที่ปรึกษา
( )

โปรแกรมแขนหุ่นยนต์จำลอง

นายวิทยากร ชินโย
นายมานพ แซ่คู
อาจารย์กิตตินันท์ รัตนธำรง อาจารย์ที่ปรึกษา
ปีการศึกษา 2530

บทคัดย่อ

โปรแกรมจำลองการทำงานของหุ่นยนต์นี้ เขียนขึ้นมาจากภาษา ปาสคาล ในโปรแกรมจะประกอบไปด้วยโปรแกรมย่อยต่างๆมากมาย ซึ่งจะทำให้แสดงภาพจำลองการทำงานของหุ่นยนต์บนจอกราฟฟิกสี (Color Graphics Monitor) โปรแกรมย่อยต่างๆเหล่านั้นถูกสร้างขึ้นมาจากกฎเกณฑ์ทางคณิตศาสตร์ โปรแกรมจำลองแขนหุ่นยนต์จำลองการทำงานของแขนหุ่นยนต์จริง

ROBOT ARM SIMULATE

Witthayakorn Chinyo

Manop Saekoo

Kittinan Rattanathamrong Advisor

Abstract

This thesis is robot arm simulate. In this project which is programed by PASCAL language. The program is composed of many subroutines that will display scene of Robot arm simulate on IBM Color graphics monitor. Many subroutines are created by principle of mathematic. It is used to simulate on working of real arm robot.

วัตถุประสงค์

1. เพื่อทำการจำลองการทำงานของแขนหุ่นยนต์ให้เคลื่อนไหวในลักษณะต่างๆ
2. ต้องการให้แสดงผลเป็นภาพ 3 มิติเพื่อให้ได้ภาพที่เหมือนจริง
3. การควบคุมการเคลื่อนไหวของแขนหุ่นยนต์ดังกล่าวให้สามารถใช้ได้ทั้ง Keyboard และ Joystic
4. ใช้ภาษา Pascal ในการเขียน Program ดังกล่าว

สารบัญ

หน้า

สารบัญ

บทที่ 1

The Micro Computer and its graphic Capabilities

1.1 Micro Computer graphic Application

Computer Graphic คือ คอมพิวเตอร์และอุปกรณ์ที่เกี่ยวกับ graphic นำมาใช้รวมกันในรูปแบบที่ผู้ใช้สามารถที่จะจับต้องส่วนของข้อมูลที่มองเห็นได้ เหตุผลพื้นฐานในการใช้ graphic ทั้งที่สร้างขึ้นโดยคอมพิวเตอร์หรือด้วยมือนั้นก็คือ graphic นั้นให้รายละเอียดของข้อมูลมากกว่าข้อมูลแบบอื่น เช่น แบบ Text ซึ่งผู้ใช้สามารถเข้าใจได้ง่ายกว่า และในการใช้ข้อมูลแบบ graphic นั้น จะต้องมีการเตรียมวิธีการแก้ปัญหาที่มีประสิทธิภาพ ส่วนเนื้อหาที่จะกล่าวถึงในบทนี้จะเป็นรูปแบบของ เนื้อเรื่องย่อและข้อมูลทางกายภาพที่จะช่วยทำให้เข้าใจเนื้อหาโดยตลอดเกี่ยวกับเรื่อง graphic

Micro Computer Graphic มีการใช้งานอย่างกว้างขวาง เช่น ทางด้านอุตสาหกรรม ธุรกิจ การศึกษา แต่ส่วนใหญ่แล้วเราจะแบ่งการใช้งาน graphic เป็น 2 พวกใหญ่ คือ Visualization funtion และ Application Area

1.1.1 รูปแบบต่าง ๆ เมื่อแบ่งตาม Visualization Funtion

Buchi ได้แบ่งการใช้งาน Computer graphic ในรูปแบบของ Visualization funtion ที่สร้างขึ้นโดยอุปกรณ์ Computer graphic เป็นส่วนย่อย ๆ ได้ดังนี้

- แสดงผล Graphic ของผลลัพธ์ที่เกิดขึ้น
- แทนที่กระดาษหรืองานอื่น เช่น กระดาษเขียนแบบ
- แสดงรายละเอียดของข้อมูลจำนวนมาก
- สร้างภาพของสิ่งที่ไม่มีอยู่ในโลกหรือวัตถุที่สร้างไม่ได้เพื่อการเรียนรู้ในการออกแบบ
- การสื่อสารทางด้าน Visual ระหว่างผู้ใช้กับ Machine

เพื่อแทนที่การสื่อสารแบบ Alphanumeric (Text)

- เลียนแบบกระบวนการ และทดสอบเพื่อหาข้อเท็จจริงก่อนที่จะมีการทดสอบจริง
- เลียนแบบภาพจริงบนพื้นโลก
- การแสดง Graphic และการสังเกตุการเลียนแบบของ Model ในทางทฤษฎี
- สร้างการออกแบบทางภาพศิลปะ
- ความบันเทิง

เพื่อให้เหมาะสมกับรูปแบบทั้งหมดของ Visualization funtion ลักษณะทาง graphic device และ Software routine นั้นจะมาในรูปแบบ Stype และ Package ที่แตกต่างกัน ส่วน Graphic ใดที่ถูกวาดขึ้นโดยวิธีการของ Computer รูปแบบของ Graphic ที่ได้นั้นจะมีคุณภาพดีหรือไม่ดีขึ้นอยู่กับ graphic device Computer aided design หรือการเลียนแบบภาพจริงที่มีอยู่ในโลกนั้นจะต้องใช้ CRT Moniter ที่มีความละเอียดสูงมาก หรือจะใช้เป็นแบบระบบ Graphic ที่สมบูรณ์แบบ โดยรูปแบบของระบบ Graphic ที่สมบูรณ์แบบนั้นจะมีความละเอียดสูงกว่าระบบ Graphic ที่พบในจอ Monitor ของคอมพิวเตอร์ทั่ว ๆ ไปที่ใช้ใน Computer ขนาดเล็กจึงทำให้การใช้งาน Microcomputer ทั่วไปในงานบางอย่างในปัจจุบันนี้มีขีดจำกัด แต่อย่างไรก็ตามในการใช้งาน Microcomputer ก็กำลังเติบโตไปตามเทคโนโลยีที่ทันสมัยในอนาคต

1.1.2 <u>รูปแบบต่าง ๆ เมื่อแบ่งตาม Application Area</u>

และอีกวิธีหนึ่งที่แบ่งการใช้งาน Computer นั้นจะอยู่ในรูปแบบของ Application Array ซึ่งการใช้งาน Graphic ส่วนมากในด้านนี้จะมีประโยชน์เพียงอย่างใดอย่างหนึ่งหรือมากกว่า Visualization funtion ตามที่กล่าวไว้แล้วจากข้างบน ซึ่งได้แก่

<u>Scientific and Statistical Graphic</u>

การประยุกต์ใช้งานในกลุ่มนี้จะรวมทั้ง graph และchart ที่นำมาใช้แทนข้อมูลทางสถิติหรือข้อมูลตัวเลขต่าง ๆ ลักษณะของตัวอย่างในกลุ่มนี้ เช่น graph 2 มิติ และ 3 มิติ ของฟังชั่นทางคณิตศาสตร์, ฟังชั่นทางเศรษฐ-ศาสตร์ รวมทั้ง Histograms, bar charts, Piechart จะแสดงให้เห็นในรูป (1.1) ตัวอย่างทั้งหมดเหล่านี้จะนำมาใช้ในการแสดงลักษณะของข้อมูลที่มีจำนวนมาก เพื่อที่จะได้ทำให้มีความเข้าใจในเหตุการณ์ที่มีความยุ่งยาก และมีจำนวนมาก

Command and Control

รูปแบบของตัวอย่างในกลุ่มนี้จะรวมทั้งสิ่งอื่นที่มีการสร้างรายละเอียดข้อมูลจริง ๆ ที่เกี่ยวข้องกับการทำงานของการดำเนินการผลิต กรรมวิธีทางเคมี, ระบบส่งจ่ายกำลัง ลักษณะดังกล่าวของ graphic แบบนี้สามารถที่จะปรับปรุงการแสดงผลการบันทึกข้อมูลและการวิเคราะห์อย่างเป็นลำดับย่อย ๆ

Computer Aid design (CAD)

เป็นส่วนที่ทำหน้าที่แทนกระดาษอย่างเช่น กระดาษเขียนแบบ ก็เป็นการใช้งานแบบหนึ่งที่มีประโยชน์อย่างมากในระบบ graphic การใช้งานในกลุ่มนี้จะรวมถึงการออกแบบทางกล, ทางไฟฟ้า, ส่วนประกอบทางด้านอิเล็ค-โทรนิคส์, ระบบปลูกสร้างจากส่วนประกอบต่าง ๆ สำหรับตัวอย่างระบบ computer graphic ที่มีอยู่ทุกวันนี้จะนำมาใช้ในการสร้างวงจรทางไฟฟ้า (ดูรูป 1.2) โดยปกติระบบ graphic ที่มีประสิทธิภาพการทำงานสูงจะนำมาใช้ในงานจริงเพราะว่าระบบ graph ที่มีประสิทธิภาพการทำงานสูงจะให้ความละเอียดของจอภาพสูงมาก

Topological definition

การใช้งานในกลุ่มนี้ได้แก Program flow chart, Chenical และ Physical Process diagram,Organization chart,electronic และลักษณะอื่นที่คล้ายคลึงกัน มี software package จำนวนมากใน Microcomputer ที่จะส่งเสริมการใช้งานในลักษณะดังกล่าวนี้ของ Computer graphic แต่จะมี

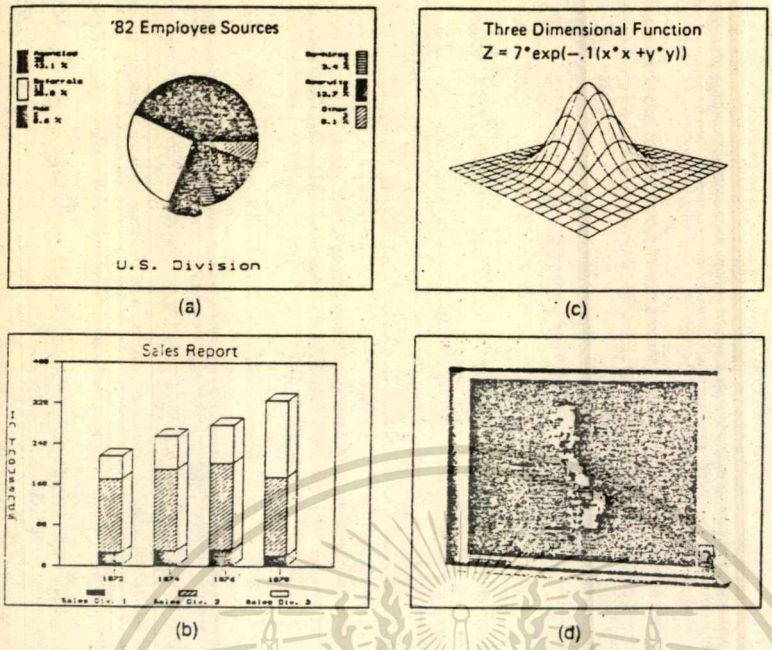


FIGURE 1.1 Scientific and statistical graphics: (a) pie chart, (b) bar chart, (c) 3-D function, and (d) computer mapping. Figures (a), (b), and (c) are obtained from the HP7470 Plotter driven by the IBM PC. Figure (d) is obtained from the Tektronix 4100 Series of computer display terminals. (Courtesy of Tektronix.)

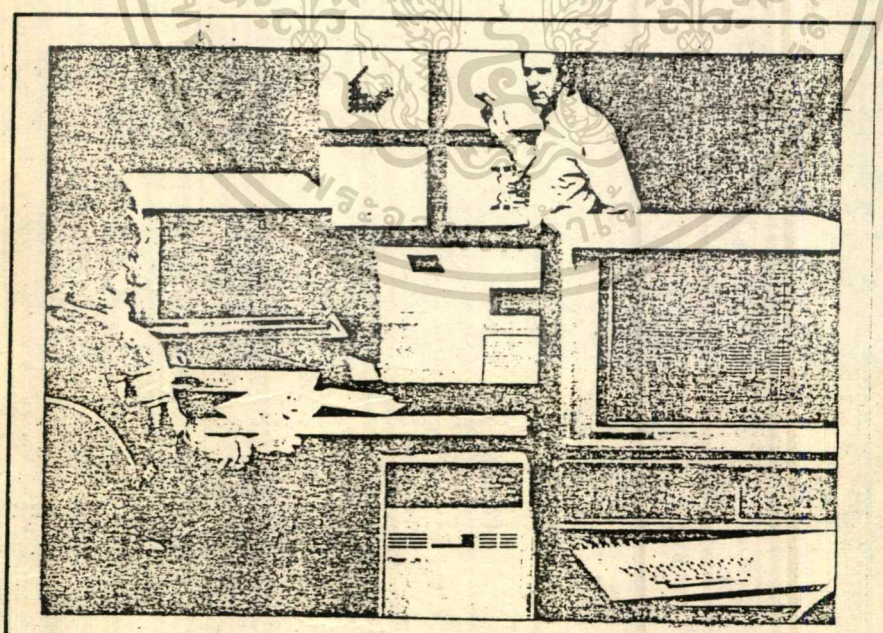


FIGURE 1.2 Computer-aided design: electronic schematic. (Courtesy of Tektronix.)

ในธุรกิจขนาดเล็ก

Geometic design

ในส่วนนี้จะแสดงถึงส่วนที่วิเศษที่สุดของ Computer graphic โดยที่ไม่เพียงแต่จะทำให้วัตถุอยู่ในรูปแบบของ 3 มิติ แต่มันยังช่วยหมุนวัตถุเพื่อยอมให้ผู้ใช้สังเกตุวัตถุที่มุมต่าง ๆ รูปแบบดังกล่าวนี้ถูกนำมาใช้ในงานที่เกี่ยวกับการแสดงผลโครงสร้าง (เช่น สะพาน) Curve Surface หรือส่วนประกอบทาง Mechanis เล็ก ๆ การใช้งานในกลุ่มนี้จะต้องใช้ระบบ graphic ที่มีความสมบูรณ์แบบเพื่อช่วยให้การแสดงผลเร็วขึ้น แต่การประยุกต์ใช้งานในด้านนี้มีขอบเขตจำกัด ดูรูป (1.3)

Pictorial Communication

Graphic ในกลุ่มนี้จะมีทั้ง Textual Material และ Picture การกระจายข่าว โดยการใช้ teletext Magazine format ก็เป็นตัวอย่างหนึ่งของกลุ่มนี้ Graphic จะช่วยส่งเสริมให้ความสามารถในการอ่านวัตถุดีขึ้น และรวมถึงรายละเอียดของวัตถุได้อย่างแน่นอน เทคนิคเดียวกันนี้ยังนำมาใช้ในการเตรียมเอกสารทาง Technical โดยเฉพาะวัตถุประสงค์นี้จะนำมาใช้ใน Group Discussion

Graphic Arts and Animation

Microcomputer นำมาใช้เป็นเครื่องมือเปลี่ยนแปลงรูปแบบสำหรับ Artists และนักออกแบบ Crusi (3) ก่อให้เกิดรูปแบบทิศทางของ Computer Arts. Musgrave ได้อธิบายถึงระบบสีหนึ่งที่นำมาใช้ในการออกแบบส่วน Animation นั้นถูกนำมาใช้กับ Computer game (รูป 1.4) Color Graphic และ Animation ยังถูกนำมาใช้ในการแสดง Processing Operating, Product Assembly steps, Information routine

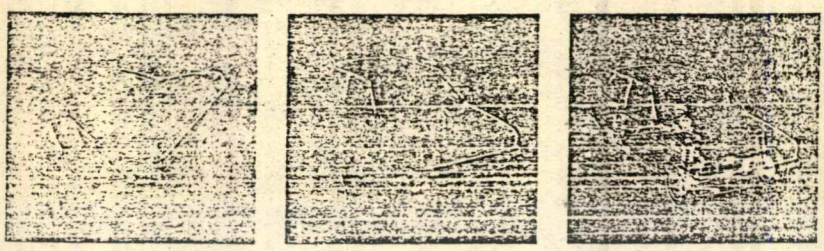


FIGURE 1.3 Geometric modeling: (a) This schematic of the body design has been defined as a separate segment, which can be stored in or recalled from local memory. Once it is stored locally, any display can be revised, then redrawn at about 150K mm/sec. (b) Repositioning scaling, or rotating this refresh segment requires only a single command from the host, rather than the usual string of commands. Result: a marked improvement in user interactivity and communications efficiency. (c) By using the local segments capability of the Tektronix 4114, this automobile schematic can be defined as one or more discrete graphic elements to be retained in the terminal's memory and redrawn by a simple command from the host computer. (Photographs courtesy

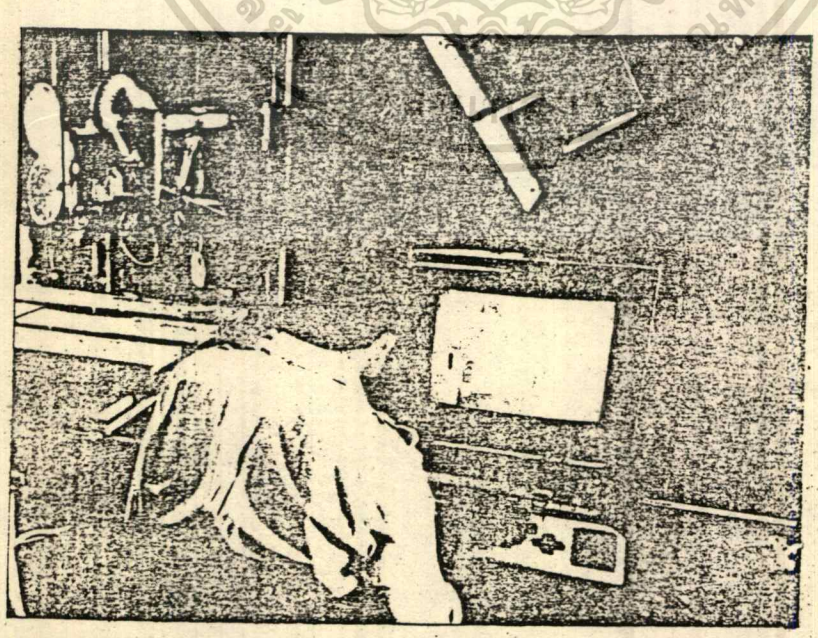


FIGURE 1.4 Graphic art and animation. (Photograph courtesy of Chromatics, Inc.)

1.2 The IBM Personal Computer

ความมุ่งใจของเรานั้นจะใช้เครื่อง IBM PC เป็นเครื่องมือพื้นฐานในการแสดงการสร้างภาพ computer graphic ด้วยเหตุนี้เองจึงเป็นสิ่งจำเป็นที่จะกล่าวถึงลักษณะโดยทั่ว ๆ ไปทางด้าน Hard ware ของเครื่อง IBM ที่ใช้ในการกำเนิดภาพ Graphic

1.2.1 Component of IBM PC

IBM PC ประกอบด้วยส่วนที่สำคัญ 2 ส่วนคือ System unit และ Keyboard ใน System จะประกอบไปด้วยอุปกรณ์ให้เลือกหลาย ๆ อย่าง และรวมทั้ง disk drive 1 หรือ 2 ตัว ที่มี Adapter (Card) ด้วย (disk drive จะถูกสร้างอยู่ภายใน system Unit) , Monochrom display Adapter, color display graphic printer , Adapter, Printer Adapler game control Adapter หน่วยความจำเพิ่มเติม 256 K byte. Block Diagrame จะแสดงไว้ดังรูป (1.5)

System unit

กลุ่มของการคำนวณหลักเรียกว่า System unit บน Board ก็จะประกอบไปด้วย Central Prodessing unit, ROM 40 K, RAM 64 K และยังประกอบไปด้วยวงจร electronic ที่ต้องนำไป drive ลำโพงขนาดเล็กและ Audio cassett ที่อยู่นอก Board.

Microprocessor คือ เบอร์ 8088 ออกแบบโดยบริษัท Intel CPU 8088 ใช้เมื่อนกับ CPU 8086 และสามารถเข้าหาข้อมูลในหน่วยความจำที่ความเร็วถึง 4 MHz ในขณะที่ CPU 8088 มีลักษณะภายในเหมือนกับ CPU 8086 ทุกอย่างแต่มันสามารถติดต่อกับอุปกรณ์ภายนอก (เช่น RAM และ ROM) ได้ทีละ 8 Bit เพราะฉะนั้น CPU 8088 จึงมีขนาดใหญ่กว่า CPU ขนาด 8 Bit แต่เล็กกว่า CPU ขนาด 16 Bit

Keyboard

Keyboard นั้นจะถูกต่อเข้ากับระบบด้วยสาย cable Key ต่าง ๆ บน Keyboard จะมีถึง 83 Keys ซึ่งทั้ง 83 Keys นี้จะใช้ในรูปแบบของข้อมูลทั่ว ๆ ไป และfuntion อื่นที่ทำหน้าที่ Process ต่าง ๆ ในการออกแบบ โดยทั่ว ๆ ไปแล้วจะมีลักษณะคล้ายกับ type writer และ calculator และยังมี calculator stype ten Key Pad คือ left right up down และ Home Key ในการ control ตัว Cursor ส่วนลักษณะทางถ่ายภาพของ Keyboard เป็นดังรูป 1.6

display unit and adapter

หนึ่งในจำนวนอุปกรณ์ที่จะต้องใช้คือ color/ graphic display Adapter แบบต่าง ๆ และ display unit จะต้องนำมาใช้รวมกัน ถ้าเราไม่ต้องการใช้ color display unit เราก็ใช้ Monochrome display unit ที่มี Monochrome / Parallel Printer Adapter ชื่อของ Monochrome / Parallel Printer Adapter หมายความถึงว่าตัวมันสามารถรวมทั้ง 2 funtion อยู่ใน Adapter เดียวกัน คือ หน่วย display และหน่วย printing ส่วนที่เป็น Displaying ก็ทำหน้าที่ drive monochrome display และส่วนที่เป็น Printing ก็จะทำหน้าที่ Drive printer : IBM Monochrom display เป็นจอภาพที่มีสีเขียวที่มีความละเอียดสูงและคุณภาพดีจะพบในเครื่องคอมพิวเตอร์ที่มีราคาแพง รูปร่างของDisplay จะมีขนาดเส้นทะแยงมุมยาว $11\frac{1}{2}$ นิ้ว และยังมีรูปแบบของสีต่าง ๆ ให้เลือก จอภาพนี้จะแสดงตัวอักษร 25 บรรทัด ๆ ละ 80 ตัวอักษร

IBM color display มีขนาดเส้นทะแยงมุมยาว 13 นิ้ว สามารถแสดงสีได้ทั้งหมด 16 สี และกำหนดรูปแบบของตัวอักษรโดยใช้ Matrix ขนาด 8 x 8 display แบบนี้จะรับสัญญาณ Video ที่มี Bandwidth กว้างถึง 14 MHz Screen จะถูก Refresh ที่ความถี่ 60 Hz พร้อมกับความละเอียดของเส้นแนวตั้งถึง 200 เส้น โดยรูปแบบพื้นฐานแล้วเป็น Moniter แบบ RGB คือต้องใช้สัญญาณ Red, green, blue, video signal,Intensity แยกจากกัน

Color / Graphic Moniter Adapter ถูกนำมาใช้ drive color

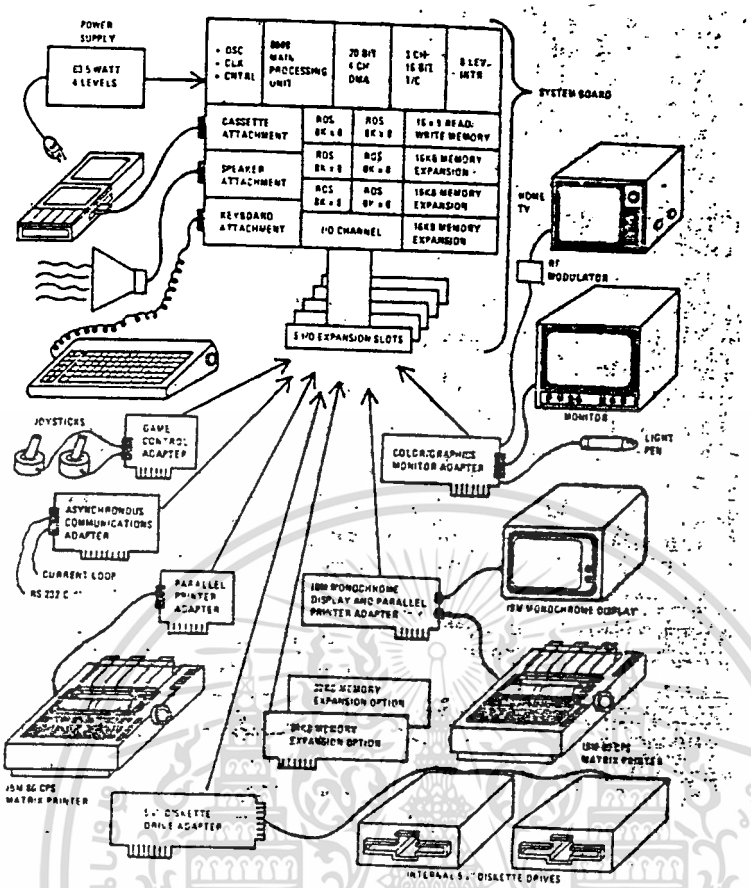


FIGURE 1.5 System block diagram for the PC. (From the *IBM PC Technical Reference Manual*. Used by permission.)

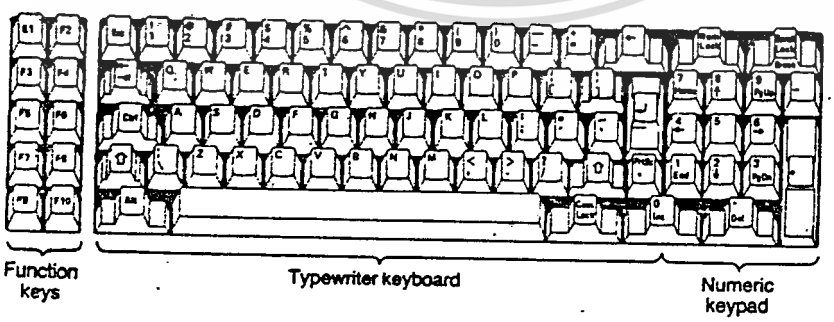


FIGURE 1.6 Keyboard diagram of the IBM PC. (From the *IBM PC BASIC Reference Manual*. Used by permission.)

display และก็ยังสามารถทำงานได้ที่ความถี่television มาตรฐานได้ด้วย (15, 750 Hz) รูปแบบของ Monitor มาตรฐานก็ยังนำมาใช้ทำเป็น Home television ได้โดยที่ผู้ใช้จะต้องมี Radio frequency Madulatex นอกเหนือจากนี้ภายใน Adapter ก็ยังมี light Pen Input Port เพื่อทำให้อุปกรณ์ในส่วนนี้ทำหน้าที่สร้างภาพ Graphic บนจอภาพ ส่วนรายละเอียดการทำงานของ color/graphic Adapter จะกล่าวถึงในส่วนหลังต่อไป

Graphic Printer

Printer คือ หน่วยของdot Matrix ที่ติดอยู่กับหัวพิมพ์ที่เคลื่อนที่ได้ โดยที่ Printer จะถูกต่อชนานกับ Parallel Printer Adapter (หรือ Monochrome display / Parallel Printer Adapter) ซึ่ง Printer นี้จะพิมพ์ได้ถึง 80 ตัวอักษร ใน 1 วินาที และหัวพิมพ์สามารถพิมพ์ตัวอักษรในขณะที่เคลื่อนที่กลับได้เช่นเดียวกับที่เคลื่อนที่ไป Mode การพิมพ์ก็มีอยู่มากมาย เช่น Normal, Compressed, emphasised, double strike, subscript, Superscript, double width และ Underline Printing, graphic Printer นี้ยังมีรูปแบบของหน้าที่Printer หลาย ๆ อย่าง เช่น Graphic character, Several Bit Image graphic mode โดยที่ Several Bit Image graphic mode นี้จะให้ผู้ใช้นำเอาข้อมูลรายละเอียดบนจอภาพสู่ Printer และยังสามารถเลือกการทำวานได โดยที่ผู้ใช้จะต้องติดตั้ง IBM PC Color Printer ซึ่ง Printer ชนิดนี้จะช่วยให้พิมพ์ตัวอักษรจาก 35 จนถึง 200 ตัวต่อวินาทีและมีความสามารถพิมพ์ได้ทั้ง 2 ทิศทางในลักษณะของ Character หลาย ๆ แบบ โดยทั่วไปมีพื้นฐานของการ Printeng อยู่ 3 แบบ คือ dato processing, text near letter และเมื่อมีการใช้กำลังควบคุม, Printer ก็จะสามารถเปลี่ยนแถบริบบอน 4 แถบ (yellow, red,blue, black) เพื่อใช้สร้างสี 7 สี รวมทั้งสีดำ Printer แบบนี้ยังช่วยให้การพิมพ์ภาพ graphic มีคุณภาพสูง โดยการให้ color graphic screen เอาข้อมูลจาก color Monitor ได้ด้วย

The Asynchronous Communication Adapter

Asynchronous communication Adapter คือหน้าต่างของระบบที่ทำการติดต่อสื่อสารกับโลกภายนอก ถึงแม้ว่าความมุ่งหมายของส่วนนี้จะออกแบบเพื่อนำมาใช้ Interface กับข่ายสายโทรศัพท์โดยผ่านตัว Modem Adapter ตัวนี้ยังสามารถต่อเข้าโดยตรงกับ Serial Printer อีกก็ได้หรือ graphic Ploter ก็ได้

Game Control Adapter

Adapter ในส่วนนี้จะยอมให้ผู้ใช้สามารถใช้ Joystick หรือ Paddle ได้โดยสามารถนำ Joystic มาต่อได้ 2 ตัว และ Paddle 4 ตัว เพื่อให้สามารถเป็นอุปกรณ์ในการเล่น game ได้

1.2.2 Component of IBM PC XT

โดยรูปแบบแล้ว IBM PC XT ก็คือ IBM PC ที่มี Power Supply ขนาดใหญ่ขึ้นและมี fixed disk (หรือ Hard disk), XT มีการเปลี่ยนแปลงระบบการทำงานและการออกแบบที่แตกต่างไปจากเดิมเล็กน้อย การออกแบบพื้นฐานและวงจรในส่วนใหญ่นั้นยังคงเหมือนเดิม ลักษณะการออกแบบจะมี RAM 128K บน Main board และยังมี drive ที่ใช้กับ floppy disk double side 1 ตัว และ disk drive ขนาด 10 MB ทั้งสองนี้ถูกติดตั้งอยู่ในระบบ และก็ยังมี disk และ diskadapter, Asynchronous Adapter Card ทั้ง 3 นี้จะนำมาใส่ใน I/O unit 8 slot ที่มีอยู่บน XT (PC มี I/O unit เพียง 5 slot เท่านั้น) ลักษณะของ XT เป็นดังรูป 1.7

XT System unit

ใน Main board ของ XT นั้นจะใช้ 64 K RAM chip แตกต่างไปจาก PC เดิมซึ่งใช้ 16K RAM chip ซึ่งจะทำให้หน่วยความจำบนเครื่อง XT มีขนาดถึง 256 K โดยที่ไม่ใช้ card ขยายเพิ่มเติม XT ยังยอมให้มีการขยายหน่วยความจำภายในได้ถึง 640 K โดยต้องเพิ่ม Memory card ซึ่งมีหน่วยความจำมากถึง 384K ส่วนของ casset port บนเครื่อง Pel ก็จะไม่มีอยู่บนเครื่อง XT และบนเครื่อง XT ก็ยัง Color Trimmer Capaciter สำหรับใช้ในการปรับแต่ง

RGB O/P Power Supply บนเครื่อง XT มีกำลังทั้งสิ้น 130 W ซึ่งมากเป็น 2 เท่าของ PC (63.5W) และบน System board ก็ยังมี ROM ขนาด 40 K และวงจรใน XT ถูกออกแบบขึ้นใหม่เพื่อจะนำมาใช้กับ Dos Version 2.00 และ Basic Version 2.00

IBM fixed disk system

ระบบ XT' fixed disk คือระบบ Hard disk แบบ Winche - stor ที่มีหน่วยความจำ 1024000000 ไบท์ จำนวนหน่วยความจำมีค่า =5500 double space typewritten pageพื้นผิวทั้ง 4 หน้า ใน Hard disk หมุนด้วยความเร็ว 3600 rpm. แต่ละพื้นผิวหน้ามี 17 sector ๆ ละ 572 byte , 306 track, XT ใช้เวลา 90 วินาที Access time ในการ Transfer ข้อมูลได้ถึง 5 ล้านบาทต่อวินาที

1.2.3 8087 Math Coprocessor

CPU 8088 มีใช้ทั้งใน PC และ XT ได้ออกแบบมาเพื่อให้เป็น Processor ร่วมกันกับ CPU 8087 . CPU 8087 มีลักษณะพิเศษ คือ สามารถทำงานได้ทั้งที่เป็นการบวก, การลบ, การคูณ, การหาร และก็การคำนวณที่ยุ่งยาก เช่น funtion ทางตรีโกณมิติ

CPU 8087 จะมี Register ขนาด 80 Bit 8 ตัว ซึ่งมีความหนาแน่นเท่ากับ Register ขนาด 16 บิท 40 ตัว ที่มีอยู่ใน Processor ทั่ว ๆ ไปเนื้อที่ใน Register เหล่านี้ใช้ในการเก็บค่าคงที่และผลลัพธ์ชั่วคราวที่เกิดขึ้นในระหว่างการคำนวณ เพราะฉะนั้นจึงเป็นการลดเวลา access timeในการเข้าหาหน่วยความจำและยังเป็นการเพิ่มความเร็วได้ดีพอกับ bus ที่มีอยู่ในระบบ

.Processor ตัวนี้ยังถูกออกแบบเพื่อยอมให้ CPU 8088 สามารถส่งผ่านการทำงานมายังตัวมันได้ และในเวลาต่อมาก็จะรับผลลัพธ์กลับเมื่อการประมวลผลในตัวมันเสร็จเรียบร้อยแล้ว แต่ CPU 8087 ก็ไม่สามารถคำนวณ funtion ควบคุมด้วยตัวมันเองหรือทำงานโดยไม่มี CPU 8088

CPU ทั้ง 2 ตัวนี้จะทำงานกันอย่าง Synchronize และจะทำงานได้มากกว่าตัวหนึ่งทำงานตัวเดียวโดด ๆ บนเครื่อง PC และ XT จะไม่มีตัว 8087 มาให้แต่จะมี Socket ของ 8087 ไว้ในระบบ (ดูในรูป 1.8)

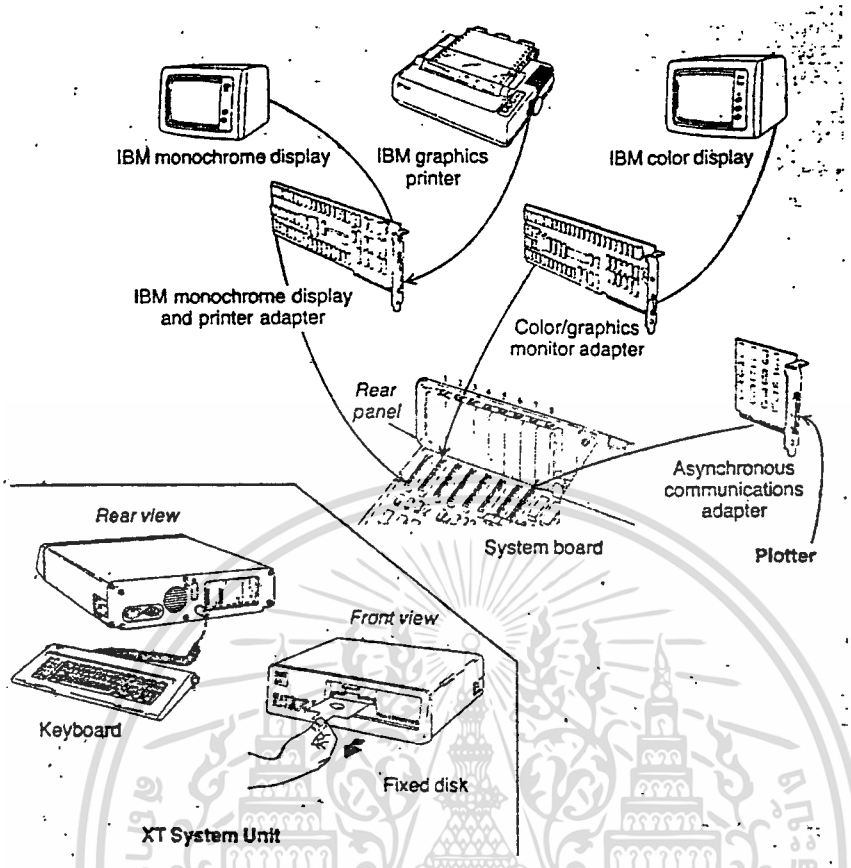


FIGURE 1.7 A typical XT configuration

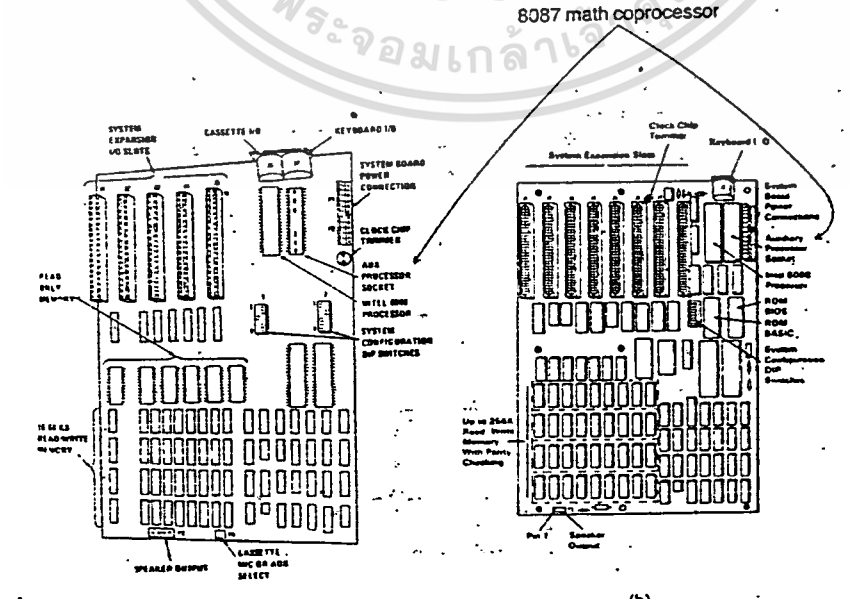


FIGURE 1.8 System board diagram with math coprocessor: (a) PC model and (b) XT model. (From the *IBM PC Technical Reference Manual*. Used by permission.)

ประโยชน์ของการมี 8087 ในระบบ PC และ XT จะเกี่ยวข้องเป็น factor ที่สำคัญอยู่ 3 อย่างคือ Precision, range, speed

Precision

เมื่อมีการนำเอาเครื่องไมโครคอมพิวเตอร์ ขนาด 16 Bit มาใช้ขั้นตอนการทำงานที่เกี่ยวกับทางคณิตศาสตร์ที่ยุ่งยากบนเครื่องไมโครคอมพิวเตอร์จะต้องใช้เวลาเป็นจำนวนมาก ส่วนในเครื่องไมโครคอมพิวเตอร์ ขนาด 8 Bit จะถูกจำกัดในด้านขนาดของจำนวน Digit ที่ตัวมันเองสามารถจะจัดการได้ แต่ในไมโครคอมพิวเตอร์ ขนาด 16 Bit เช่นPC และ XT สามารถจัดการเกี่ยวกับจำนวนบิทได้ยาวถึง 32 Bit ซึ่งค่า 32 บิทเหล่านี้จะทำให้ค่าที่คำนวณได้ถูกต้องเป็น 2 เท่า และมีความแม่นยำเป็นจำนวนถึง 17 Digit ด้วยค่าต่าง ๆ ที่อยู่ในช่วง 4.19×10^{-307} จนถึง 1.67×10^{308} ค่าเหล่านี้คือ ตัวเลขจำนวนจริง เพราะว่าค่าเหล่านี้จะประกอบด้วยส่วนที่เป็นเลขจำนวนเต็มและส่วนที่เป็นเศษส่วน (ค่าเหล่านี้บางทีเรียกว่า floating Point Number โดยที่จุดทศนิยมลอยอยู่รอบ ๆ ภายในตำแหน่งหน่วยความจำที่เลขจำนวนนี้เก็บอยู่) ตารางที่ 1.1 แสดงให้เห็นถึงค่าที่มากที่สุด และค่าที่น้อยที่สุดใน DATA แต่ละแบบ

Precision จะกำหนดโดยขนาดของ Mantissa ซึ่งเป็นส่วนที่ใช้ตัวเลขจำนวนมากที่สุดของเลขจำนวนนั้นถ้าเราไม่คิดถึงจุดทศนิยมที่ตำแหน่งนั้น ๆ โดยปกตินั่นเอง Precision จะไม่มีปัญหาเกี่ยวกับเลขจำนวนเต็มแต่ Precision นี้จะมีผลที่สำคัญที่จะต้องพิจารณาในระบบเลขจำนวนจริง เพราะว่ามีทศนิยมไม่รู้จบเกิดขึ้น เมื่อตัวเลขจำนวนจริงไม่สามารถกำหนดได้โดยค่าที่แน่นอนด้วยจำนวน digit ที่ไม่จำกัดจำนวนเช่นตัวอย่าง 1 หารด้วย 6 ก็จะได้ผลลัพธ์ 1.6666..... ซึ่งการแทนค่าที่ถูกต้องนั้นจะแทนด้วย digit ที่ไม่สิ้นสุด แต่ตัวเลขตัวนี้จะไม่สามารถนำมาใช้ในการคำนวณได้โดยง่าย ดังนั้นจึงต้องมีการกำหนดจำนวน digit ที่น้อยที่สุดของทศนิยมไม่รู้จบเพื่อจำมันจะถูกนำมาเก็บในหน่วยความจำที่มีอยู่อย่างจำกัดบนเครื่อง computer ได้ CPU 8087 จะยอมให้มีค่า Mantissa จำนวนถึง 64 Bit ซึ่งมีความถูกต้องถึง 1 ใน 2^{64} แต่ความถูกต้องเหล่านี้ไม่ค่อยนำมาใช้ในการคำนวณแบบปกติแต่มันจะมี

ความสำคัญในระหว่างสภาวะการคำนวณอย่างต่อเนื่องของตัวเลขจำนวนมาก ที่มีหลายขั้นตอนที่เกี่ยวข้องกับทศนิยมไม่รู้จบและ Truncate Operating

Range

Range จะเป็นการกำหนดความมากน้อยของค่าตัวเลขที่แสดงได้ เช่นตัวเลขจำนวนเต็ม range จะถูกกำหนดโดยจำนวนที่ Bit ที่ใช้ ถ้าใช้ 16 Bit ค่าสูงสุดและต่ำสุดของเลขจำนวนเต็มเราสามารถแทนได้อยู่ในช่วง -32768 จนถึง 32768 แต่สำหรับตัวเลขจำนวนจริง range จะกำหนดโดยขนาดของ Exponent สำหรับ CPU 8087 แล้ว exponent จะมีค่ายาวถึง 15 Bit ซึ่งจะแทนขยายความกว้างของตัวเลขที่จะแสดงได้อยู่ในช่วง 3.4×10^{-4932} จนถึง 1.2×10^{-4932}

Speed

ความเร็วเป็นสิ่งสำคัญสำหรับงานบางอย่างทางคณิตศาสตร์และทางวิทยาศาสตร์ ใน computer graphic, 3-D graphic และ Animation Routine ที่จะคำนวณหารายละเอียดเกี่ยวกับตำแหน่งที่จะต้องใช้การคำนวณทางคณิตศาสตร์อย่างมากมายอย่างต่อเนื่อง และก็จะมีประโยชน์มากเมื่อใช้ CPU 8087, CPU 8087 สามารถเพิ่มความเร็วของ 8088 Machine language Programe เป็นอัตราส่วนถึง 10 ถึง 1000 เท่า Cooke (2) ได้รายงานว่า 8087 จะสามารถทำงานได้เร็วกว่า 8088 ในรูปแบบการคำนวณแบบเลขจำนวนจริงทุกชนิด

1.2.4 Method and Hardware Requipment to Creatgraphic

IBM PC สามารถนำมาใช้ในทาง graphic ที่แตกต่างกันได้ 3 รูปแบบคือ character graphic, Plotter graphic, Pixel graphic ทั้งนี้จะขึ้นอยู่กับรูปแบบ graphic ที่ต้องใช้ ผลลักษณะทาง Hardware ของทั้ง 3 แบบ ที่สามารถเปลี่ยนแปลงได้ดังแสดงในตาราง 1.2

Character Graphic

Charater ที่นำมาแสดงผลนั้นได้ทำขึ้นโดยใช้กลุ่มตัวอักษรของระหัส ASC II เพิ่มเติมซึ่งระหัสเหล่านี้มีอยู่แล้ว ในเครื่อง PC และมีถึง 128

code ที่มีรูปแบบของ display ที่มีลักษณะต่าง ๆ กัน ครั้งหนึ่งของโคดเหล่านี้ ได้นำมาใช้เป็นตัวอักษรในการ drawing นอกจากนี้ก็ยังมีกลุ่มตัวอักษร เส้นที่จะนำมาใช้ในการวาดเส้นเดียว, เส้นคู่ของรูปสี่เหลี่ยม, graph, chart และถ้าเราใช้ภาษา Basic เราก็สามารถเข้าหาโคดเหล่านี้โดยการใช้ข้อความ "Print CHR #(n) " เมื่อ n คือตัวเลข code ของ ASC II รูปแบบของกลุ่มตัวอักษร graphic ที่มีประโยชน์ในการวาด grawing ถูกแสดงในตาราง 1.3

Plolter graphic

ในการใช้ Plolter graphic จะต้องต่อกับอุปกรณ์แบบพิเศษที่เรียกว่า Plolter ซึ่งถูก drive โดยเครื่อง PC Plolter หลายแบบนั้นสามารถต่อเข้ากับเครื่อง PC ได้โดยที่แต่ละ Plolter ก็จะมีคำสั่งทาง graphic เป็นของตัวเอง โดยปกติแล้วคำสั่งเหล่านี้จะแตกต่างไปจากคำสั่ง graphic ที่มีอยู่บนเครื่อง PC สำหรับตัวอย่างเช่น HP-7470 เป็น Serial graphic plolter ที่ต่อเข้ากับเครื่อง PC โดยใช้ HP-232-C cable ซึ่งเป็นส่วนเพิ่มเติมมาจาก PC's Asynchronous Communication Adapler และผู้ใช้ต้องใช้ภาษา Hewlett Packard Graphic Language (HP - GL) ไปขับ Plolter Plolter language ส่วนมากถูกเขียนด้วยภาษา Basic แต่ยังมีคำสั่งทาง graphic โดยเฉพาะบางอย่างที่มีความแตกต่างในเครื่อง PC แต่ละแบบ

Pixel Graphic

เป็นอีกวิธีหนึ่งซึ่ง display screen จะถูกแบ่งเป็นรูปสี่เหลี่ยมซึ่งเป็นส่วนประกอบเล็ก ๆ ของภาพและมีจำนวนมาก โดยเราจะเรียกสี่เหลี่ยมเล็กนี้ว่า dot หรือ Pixel ในแต่ละ Pixel สามารถจะ on หรือ off ได้ และในการ on หรือ off จุด ๆ หนึ่งบนจอภาพนั้นหมายถึงเราจะสร้าง computer ให้สามารถ Turn on ได้ในตำแหน่งตามที่เราต้องการ ส่วนการอ้างถึงแต่ละ Pixel จะอ้างโดย Coordinate โดยเหตุนี้เราก็กำหนด Pixel ในรูปแบบของ (X,Y) เมื่อ X คือตำแหน่งที่นับไปทาง column และ Y คือ ตำแหน่งที่นับไปตามแนว ROW ของตารางจอภาพ บนเครื่อง PC ใน

TABLE 1.1 Data representation with the 8087

Data type	Bits	Significant digits (decimal)	Approximate range (decimal)
Word integer	16	4	$-32{,}768 \leq X \leq +32{,}767$
Short integer	32	9	$-2 \times 10^{9} \leq X \leq +2 \times 10^{9}$
Long integer	64	18	$-9 \times 10^{18} \leq X \leq +9 \times 10^{18}$
Packed decimal	80	18	$-99\ldots99 \leq X \leq +99\ldots99$ (18 digits)
Short real†	32	6-7	$8.43 \times 10^{-37} \leq \lvert X\rvert \leq 3.37 \times 10^{38}$
Long real†	64	15-16	$4.19 \times 10^{-307} \leq \lvert X\rvert \leq 1.67 \times 10^{308}$
Temporary real	80	19	$3.4 \times 10^{-4932} \leq \lvert X\rvert \leq 1.2 \times 10^{4932}$

†The short and long real data types correspond to the single and double precision data types.
Source: *IBM PC XT Technical Reference Manual.* Used by permission.

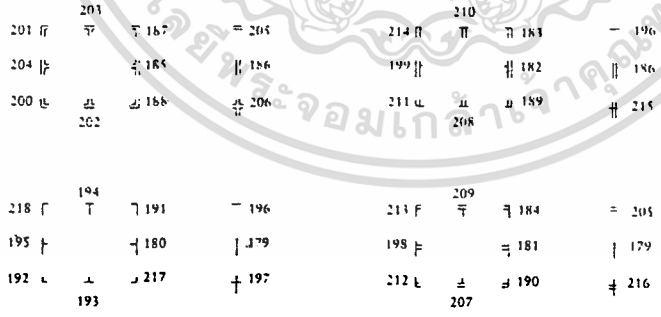
TABLE 1.2 Minimum hardware configuration

Hardware	Character graphics	Pixel graphics	Plotter graphics
System unit†	X	X	X
Keyboard	X	X	X
Monochrome display unit	X		X
Color/graphics display unit		X	
Monochrome/parallel printer adapter	X		X
Color/graphics monitor		X	
IBM graphics printer	X	X	
Printer cable	X	X	
Plotter			X
Asynchronous communication adapter			X
RS 232-C Cable			X

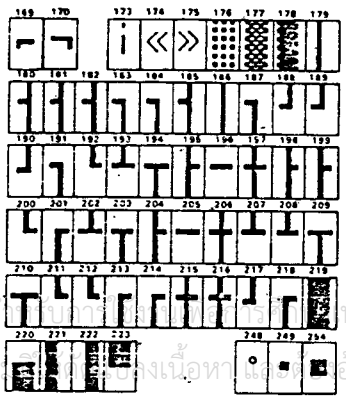
†Includes two disk drives, 64K RAM

TABLE 1.3 Character chart with ASCII decimal values

For monochrome display unit



For graphics printer



From the *IBM PC Technical Reference Manual.*

การสร้างภาพ graphic จะถูกสร้างด้วย Pixel โดยการควบคุมผ่านทาง Memory Mapped display format โดยที่แต่ละตำแหน่งหน่วยความจำจะมีลักษณะตรงกับแต่ละตำแหน่งบนจอภาพPC'S Memory Mapped Screen ถูกเชื่อมโยงเข้ากับหน่วยความจำ computer ที่จะถูก scan อย่างต่อเนื่องโดยวงจรบน color / graphic Adapter เพื่อสร้างสัญญาณ Vidio Signal ที่จะทำให้เกิดภาพบนจอภาพ และด้วยกรรมวิธี Memory Mapped Screen เราสามารถอ่านหรือเขียนข้อมูลที่แสดงผลได้โดยตรงโดยการเข้าหา display Memory ลักษณะดังกล่าวนี้ทำให้เรา สามารถจับต้องส่วนต่าง ๆ ของรูปภาพได้โดยง่ายและเราจะใช้วิธีดังกล่าวนี้โดยตลอดเนื้อหา graphic

ส่วนในรูป 1.9 แสดงให้เห็น graphic O/P ที่ได้โดยการใช้วิธีนี้

1.3 Characteristic of the color / graphic Adapter

อุปกรณ์Hardware ที่เรียกว่า color / graphic Adapter จะเป็นส่วนที่ทำให้เครื่อง PC นำเอาภาพ color graphic ออกสู่จอภาพ Adapter นี้ก็คือแผ่นวงจรที่วางอยู่ใน slot ใดๆ slot หนึ่งภายใน System board ในส่วนนี้เราจะแสดงให้เห็นรายละเอียดวิธีการที่ Adapter ทำการสร้างรูปแบบ display และสีของจอภาพที่ปรากฏขึ้นบนจอภาพ โดยลำดับแรกเราจะกล่าวถึง Memory Mapped ของ 8088 เพื่อแสดงให้เห็นว่า PC เก็บส่วนของ graphic และข้อมูล display ไว้ในตำแหน่งที่ใดในหน่วยความจำ

1.3.1 Memory Requirement

โดยความจริงแล้ว graphic O/P ของ IBM PC ก็คือส่วนแสดงผลส่วนหนึ่งของหน่วยความจำคอมพิวเตอร์ และเหตุนี้เองเราจะเริ่มต้นพิจารณา IBM PC graphic ด้วยการเรียนรู้วิธีการแบ่งและใช้หน่วยความจำ ซึ่งในบางครั้งเราอาจจะมีการเปลี่ยนแปลงรูปแบบหน่วยความจำเพื่อนำมาใช้กับ computer ลักษณะรูปแบบมาตรฐานของ Memory Map ในเครื่อง IBM ถูกแสดงไว้ในตาราง 1.4 ในแต่ละหน่วยความจำจะเก็บรายละเอียดของข้อมูล 8 Bit (1 byte) ตำแหน่งที่อยู่ของข้อมูล 1 Byte เรียกว่า Address ค่า Address และค่าข้อมูลอาจจะเรียกในเทอมของเลขฐานสิบและเลขฐานสิบหก

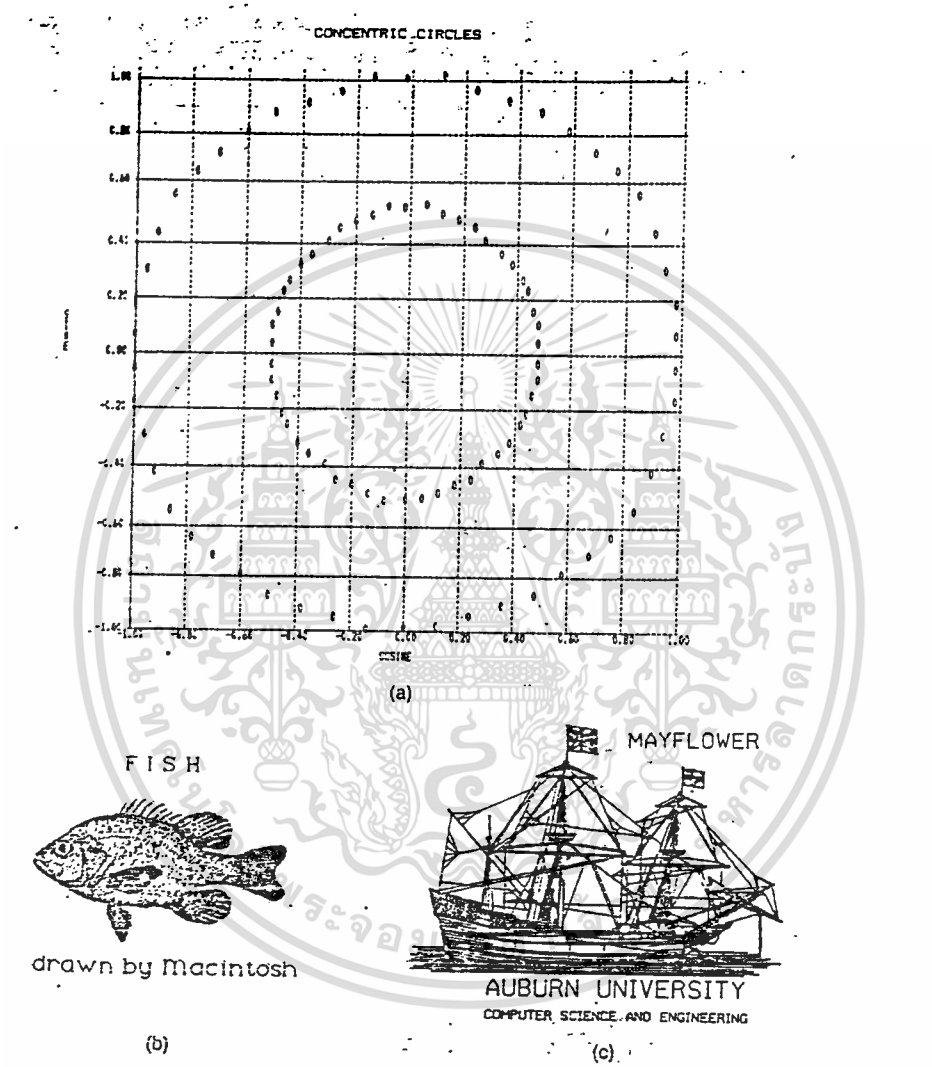


FIGURE 1.9 (a) Method of creating graphics: character graphics. (This figure is obtained from the IBM PC Printer.) (b) Raster graphics: A fish drawn on the Macintosh™ (Macintosh is a trademark licensed to Apple Computer, Inc.). (c) Plotter graphics: A ship plotted by HP7470A PLOTTER

TABLE 1.4 System memory map (increments of 16KB).
Source: IBM PC and XT technical reference manuals.

PC model	Memory	XT model
Function	Start address Decimal Hex	Function
16–64 KB READ/WRITE memory on system board	0 00000 16K 04000 32K 08000 48K 0C000	
Up to 192 KB memory in I/O channel	64K 10000 • • • • • • 240K 3C000	128-256K R/W memory on system board
384 KB future R/W memory expansion in I/O channel	256K 40000 • • • • • • 624K 9C000	384K R/W memory expansion I/O channel
Reserved	640K A0000 656K A4000 672K A8000 688K AC000	
112 KB Graphics/ display video buffer — Monochrome	704K B0000 720K B4000	Monochrome — 128K reserved
Color/graphics	736K B8000 752K BC000	Color/graphics
192 KB memory expansion area	768K C0000 784K C4000 800K C8000 816K CC000 832K D0000 848K D4000 864K D8000 880K DC000 896K E0000 912K E4000 928K E8000 944K EC000	Fixed disk control 192K Read-Only memory expansion and control
Reserved	960K F0000 976K F4000 992K F8000	
48 KB base system ROM	1008K FC000	64K base system ROM, BIOS, and BASIC

00000	BIOS interrupt vectors (0H–1FH)
00080	DOS interrupt vectors (20H–3FH)
00100	User interrupt vectors (40H–7FH)
00200	BASIC interrupt vectors (80H–FFH)
00400	BIOS data area
00500	BASIC and DOS data area
00600	62.5 K user raw memory

From the *IBM PC* and *XT Technical Reference Manuals.*

(&H นำหน้า) เครื่อง PC สามารถมีหน่วยความจำได้ถึง 1024 K byte และค่า Address อยู่ในช่วง 0 จนถึง 1024 K (หมายเหตุ เพื่อให้เป็นรูปแบบจะให้ 1 K= 1024)

หน่วยความจำ 64 K byte ต่ำสุดจะเป็นส่วนที่เก็บ Bios (Basic Input Output system) routine และ R/W Memory และอยู่บน System board (Bios routine ประกอบด้วย Input Output service routine ที่สร้างขึ้นบนเครื่อง PC โดยที่ routine เหล่านี้ก็คือ Interupt Service routine ที่สนับสนุนเป็น Software ขึ้นโดยการใช้ IBM PC Macro Assembler)

หน่วยความจำ 19.2 K ไบท์ต่อมาก็คือหน่วยความจำเพิ่มเติมบน System board ซึ่งเราสามารถขยายได้ถึง 256 K byte ส่วนหน่วยความจำอีก 384 K ไบท์ถัดมาที่เริ่มต้นในตำแหน่ง Address 40000 ก็ใช้เป็นหน่วยความจำ Read/write สำหรับการขยายต่อไปในอนาคต

หน่วยความจำที่เพิ่มเติมมาอีก 16 K byte ต่อมาอีก IBM จะสำรองไว้สำหรับใช้ในอนาคต (กลุ่มของหน่วยความจำ 16 K ที่สำรองไว้อีกส่วนหนึ่งจะเริ่มต้นที่ตำแหน่ง F0000) หมายเหตุว่า XT จะสำรองหน่วยความจำไว้ถึง 128 K byte และที่ตำแหน่ง Address B0000 คือจุดเริ่มต้นของหน่วยความจำขนาด 16 K byte ที่ PC และ XT เก็บค่า Monochrome Screen display data ไว้ และที่อีกตำแหน่ง Address B8000 ถึงจุดเริ่มต้นของกลุ่มหน่วยความจำอีก 16 K byte ที่ใช้อยู่ภายใน color/ graphic Adapter chip ทั้งหมดที่ประกอบด้วยหน่วยความจำนี้จะอยู่บน color / graphic Adapter ซึ่งก็หมายความว่า color / graphic Adapter จะทำหน้าที่เป็นส่วนของหน่วยความจำขนาด 16 K byte ที่เพิ่มเติมมาจากหน่วยความจำหลักของ 8088 บน System board

ส่วนหน่วยความจำอีก 192 K byte ต่อมาที่เริ่มต้นในตำแหน่ง C0000 เตรียมไว้ในการขยาย ROM หรือหน้าที่อื่น และหน่วยความจำที่เหลืออีก 8 K byte เริ่มต้นในตำแหน่ง Address F4000 นั้นเพื่อให้ผู้ใช้ระเตรียม Program Rom chip ซึ่งเราสามารถใส่ลงภายใน socket ที่สำรองไว้บน system board และหน่วยความจำสุดท้ายอีก 40 K ที่ตำแหน่ง Address สูงสุดเป็นส่วนที่เก็บ

ตัวแปร cassetle basic และ Bios ROM chip หมายเหตุว่า XT มีการกำหนดหน่วยความจำแตกต่างไปเล็กน้อยคือ มีการใช้ส่วนส่วนหนึ่งของส่วนสุดท้ายในหน่วยความจำหลักเพื่อใช้ในการควบคุม fixed disk

1.3.2 Hareware Overview

.Types of display Supperts

Color / graphic Adapter นำมาใช้กับอุปกรณ์ Display 2 แบบคือ direct drive Moniter และ Composite Monitor รวมทั้งเครื่องรับ TV ที่มี RE Modulator ด้วยลักษณะของ Display อาจจะเป็นได้ทั้ง color หรือ Monochrome

Direct drive Moniter หรือเรียกว่า RGB Moniter จะมีสัญญาณ I/P video 3 I/P คือ red, green, blue และสัญญาณเหล่านี้จะเป็นสัญญาณแบบ Digital เพื่อว่ามันทั้งหลายจะสามารถแสดงสภาวะ on หรือoff ได้และไม่ขึ้นต่อกัน Moniter แบบนี้สามารถแสดงสีได้ถึง 8 สี RGB digital Moniter บางแบบจะมีสัญญาณ I/P ถึง 4 เส้นสัญญาณ ซึ่งสัญญาณเส้นที่ 4 นั้นใช้ควบคุมความสว่างของสีทั้งสามและทำให้สีที่แสดงผลเพิ่มขึ้นเป็น 16 สี ส่วนใน RGB Analog Moniter นั้นความเข้มของลำแสง (bean) จะเป็นสัดส่วนตาม Voltage ที่จ่ายให้แก่ Video I/P Moniter แบบนี้จะมีราคาแพงกว่า digital Moniter ในทางปฏิบัติแล้ว Moniter แบบนี้สามารถแสดงสีได้อย่างมากมาย color/graphic Adapter ทำหน้าที่เพียงสร้างสัญญาณ Binary RGB และ Intensity เท่านั้น มันจึงไม่ได้ถูกนำมาใช้กับ RGB Analog Moniter

Composite Monitor จะรับแต่สัญญาณรวมที่ประกอบด้วยสัญญาณ Sync กับรายละเอียดของภาพ เครื่องรับ TV จะถูกประกอบขึ้นจาก Monitor ที่มีการนำเอาสัญญาณภาพออกจาก RF Carrier Monitor แบบนี้จะมีราคาถูกกว่า RGB Monitor เพราะว่าความละเอียดของภาพไม่ดี

Major hareware component

ส่วนที่สำคัญของ color/graphic Adapter ได้แก่ IC 6845 CRT

control หน่วยความจำ display 16 Kbyte buffer, character generater, Mode Select Register, color Select Register, timing generator และ Composite color generator ส่วน Block diagram จะแสดงให้เห็นถึงเส้นทางเดินของข้อมูลที่สำคัญที่ต่ออยู่ระหว่างส่วนประกอบเหล่านี้ ดังแสดงในรูป 1.10

IBM Screen ที่ใช้เป็นแบบ Raster display ซึ่งประกอบด้วยกลุ่มของเส้น Horizontal ที่ทำขึ้นมาจาก Pixel PC 6845 CRT controller จะเป็นตัวทำให้วงจรอิเล็คทรอนิคไปทำการ drive raster display และนอกจากนั้น CRT controller ยังได้สร้างสัญญาณ Horizontal และ Vertical Video Timing และสัญญาณ Address เพื่อใช้ในการเข้าหาข้อมูลใน display buffer และ character generator

Color / graphic Adapter มี Mode พื้นฐานการทำงานอยู่ 2 แบบ คือ Alphanumeric (Text) และ graphic Mode ทั้งสองนี้จะถูกควบคุมโดย Mode Select Register ใน Text Mode จะมีการแสดงผลให้เลือกได้ 2 แบบ คือ 40 Column และ 80 Column ในแต่ละแบบที่เลือกตัวอักษรจะถูกแสดงให้เห็นแถวละ 25 ตัว ส่วนใน graphic Mode นั้น adapter นั้นก็มีการแสดงผลให้เลือกอีก 2 แบบคือ Medium graphic Resolution และ High graphic Resolution ใน Medium Resolution จะมีเส้นราสเตอร์ 200 เส้นแต่ละเส้นมี 320 Pixel (หรือเรียกอีกแบบหนึ่งว่า 300 x 200 graphic (Mode) ใน High Resolution Mode ก็มีเส้นราสเตอร์ 200 เส้น แต่ละเส้นจะประกอบด้วย Pixel 640 Pixel (หรือเรียกอีกแบบว่า 640 x 200 graphic Mode)

display buffer จะอยู่ภายในพื้นที่หน่วยความจำของ Processor โดยจุดเริ่มต้นอยู่ที่ตำแหน่ง Address & H B8000 โดยที่ตัวมันจะใช้หน่วยความจำ RAM ถึง 16 K byte และสามารถถูกเข้าหาข้อมูลภายในได้ทั้ง Central processing unit และ graphic control unit. ข้อมูลที่เก็บอยู่ใน display buffer จะถูกอ่านอย่างต่อเนื่องและแปรเป็นความหมายก่อนนำ

ออกสู่จอภาพ ความแตกต่างระหว่าง Text mode และ graphic mode ก็คือ DATA ที่เก็บอยู่ภายใน Display buffer จะถูกแปลความหมายในรูปแบบอย่างไร

Character generater ประกอบขึ้นจาก ROM ขนาด 8 K byte ซึ่งภายในจะประกอบด้วยรูปแบบต่าง ๆ ที่ใช้ในการสร้างตัวอักษรใน Text mode และใน Text mode นี้จะทำการกำเนิดตัวอักษรที่แตกต่างกัน 3 แบบแต่ละแบบกำหนดตัวอักษรได้ 256 ตัวอักษร โดยที่ 2 แบบแรกจะนำมาใช้ใน color / graphic Adapter และอีกแบบหนึ่งจะนำมาใช้ใน Monochrome Adapter รูปที่ 1.11 แสดงให้เห็นองค์ประกอบของ character generater ซึ่งใน ROM 2 Kbyte แรกประกอบด้วย 8 แถวบนของกลุ่มตัวอักษรแบบ dot ขนาด 7 x 9 ที่ใช้ใน Monochrome และ 2K byte ถัดมาจะประกอบด้วย 8 แถวล่างและใช้เพียง 6แถวแรกเท่านั้น และ 2 Kbyteชุดที่ 3 และชุดที่ 4 จะประกอบด้วยรูปแบบของ dot Matrix ขนาด 5x 7 และ 7x 7 ตามลำดับและนำมาใช้ใน color / graphic Adapter

เนื่องจาก character generater ไม่สามารถถูกอ่านและเขียนได้โดย Software control ดังนั้น Adapter จึงจะต้องทำงานโดยวิธีการต่าง ๆ เพื่อที่จะสร้างตัวอักษรใน graphic mode และตัวอักษรใน graphic mode นั้นจะอยู่ในรูปแบบของการโปรแกรม Pixel และต้องใช้ข้อมูล 8 Byte ในการสร้างตัวอักษรแต่ละตัว ซึ่งในหน่วยความจำ 8 Byte นี้จะกลุมเนื้อที่การแสดงผลของ Pixel ขนาด 8 x 8 ใน ROM Bios ของเครื่อง PCในส่วนหนึ่งของที่มีอยู่จะประกอบด้วย character Look up table ที่ไดตระเตรียม code ของตัวอักษร 128 อัวชุดแรก ที่มีรูปแบบเหมือนกันกับที่มีอยู่ภายใน character generator ใน graphic mode นั้น Bios จะทำการอ่านรูปแบบ Palter จาก Table และก็จะนำไป Turn of, turnoff Pixel ในตำแหน่งที่เหมาะสมในการอ่านตัวอักษรจาก Screen Memory Bios Program จะต้องเอารูปแบบข้อมูลใน display buffer มาทำการเปิด table look up เพื่อที่จะกำหนดได้ว่าตัวอักษรอะไรแสดงผลออกทางจอภาพ

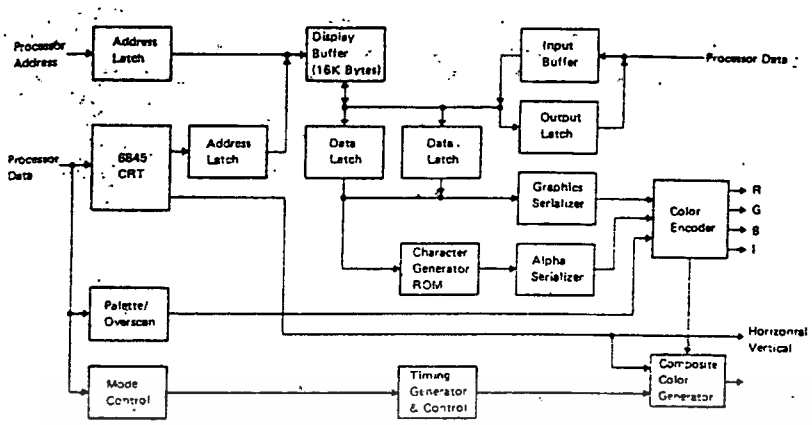


FIGURE 1.10 Color/graphics adapter block diagram. (From the *IBM PC Technical Reference Manual.* Used by permission.)

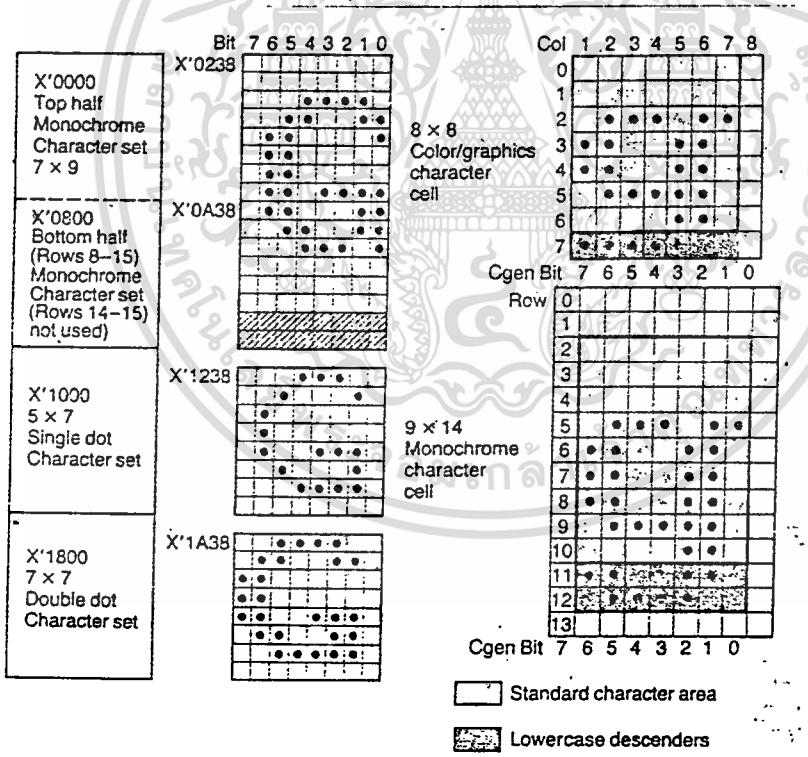


FIGURE 1.11 The character generator: (a) Memory organization and (b) character cell layout. (Reprinted from *PC Technical Journal,* copyright ©1984, Ziff-Davis Publishing Co.)

Timing generater

จะเป็นตัวสร้าง Timing Signal เพื่อนำมาใช้ใน IC 6845 CRT controller และ dynamic Memory และมันยังแก้ปัญหาของการปะทะกันระหว่าง CPU และ graphic controller ในเวลาที่ทั้งสองเข้าหาข้อมูลใน display buffer โดยทั่ว ๆ ไป color / graphic Adapler จะดำเนินกรรมวิธีเกี่ยวกับข้อมูลในรูปแบบดังต่อไปนี้

ข้อมูลที่อยู่ภายใน display buffer จะถูกอ่านโดย CPU มาเป็นจำนวน 2 ไบท์ที่เวลาหนึ่ง ๆ และนำเข้าไปเก็บไว้ภายใน 8 Bit data latch 2 ตัว จากที่นี้ข้อมูลก็จะผ่านไปยัง graphic หรือ alpha Serializer ซึ่งจะเป็นตัวนำเอา Pixel หนึ่งตัวที่เวลาใด ๆ (สำหรับ Alphanumeric Mode นั้นลำดับแรกข้อมูลจะผ่านไปยัง Character generater ก่อนที่จะถึง alpha Serializer) และจากที่นี่ data ก็จะผ่านไปยัง Color encoder ซึ่งเป็นที่ที่ข้อมูลจะถูกเปลี่ยนเป็นรายละเอียดของ composite video เพื่อเป็น O/P ส่งต่อไปยังจอภาพ

1.3.3 Programming the 6845 CRT Controller

นอกเหนือไปจากหน่วยความจำ display buffer ขนาด 16 K byte แล้ว Adapler ยังมี Port I/O อยู่อีก 7 Port ซึ่งสามารถโปรแกรมควบคุมได้ (ดูในตาราง 1.5) เราจะกล่าวโดยย่อถึง 3 Port ใน 7 Port ที่เกี่ยวข้องกับการโปรแกรม graphic

ต่อจากนั้นจะใช้ Bios Program เพื่ออ่านและเขียนข้อมูลจาก I/O Port เหล่านี้ ในการเข้าหา Bios Program เราต้องใช้ IBM macro Assembler I/O Port นี้สามารถถูกเข้าหาได้โดยภาษา Basic ได้อีกด้วย ด้วยการ execute คำสั่ง out เพื่อส่งข้อมูลไปยัง output port หรือคำสั่ง INP เพื่อรับข้อมูล 1 Byte จาก Input Port

6845 Address Register คือ Register ที่เขียนได้อย่างเดียว มีขนาด 5 Bit นำมาใช้ในการเลือก data Register ภายในตัวใดตัวหนึ่งใน 18 data Register ภายในทั้งหมดของ 6845 CRT controller ด้วย

TABLE 1.5 I/O ports for color/graphics adapter

I/O Port Address	Device name	Read/ Write	Active data bits
&H3D4	6845 address register	W	- - - 4 3 2 1 0
&H3D5	6845 data register	R/W	- - - 4 3 2 1 0
&H3D8	Mode select register	W	7 6 5 4 3 2 1 0
&H3D9	Color select register	W	- 5 4 3 2 1 0
&H3DA	Status register	R	- - - - 3 2 1 0
&H3DB	Clear light-pen latch	R/W	- - - - - - - -
&H3DC	Set light-pen latch	R/W	- - - - - - - -

Mode-Register (&H3D8)

Bit 0	High-resolution dot clock (80 character alpha)
1	Graphic select
2	Black and white select
3	Enable video
4	640 graphics select
5	Alpha blink enable

Color Select Register (&H3D9)

Bit 0	Blue	Alpha border/graphics background
1	Green	Alpha border/graphics background
2	Red	Alpha border/graphics background
3	Intensity	Alpha border/graphics background
4	Intensity	Alpha background/medium resolution foreground
5	Medium resolution foreground color select (blue)	

Status Register (&H3DA)

Bit 0	Display inactive
1	Light-pen trigger set
2	Light-pen switch open
3	Vertical sync

From T. V. Hoffmann, "The IBM Color/Graphics Adapter," Reprinted from *PC Technical Journal* 1(1), 1983. Copyright © 1983, Ziff-Davis Publishing Company. Used by permission.

การเขียนตัวเลข Register นั้นเข้าไปยัง Port นี้ต่อมา Select : Register จะถูกอ่านหรือถูกเขียนโดยผ่านทาง 6845 DATA Register ตาราง 6.1 สรุปค่าต่าง ๆ ที่จำเป็นต้อง Load เข้ามาภายใน 6845 CRT controller register เพื่อควบคุม Mode สนับสนุนการทำงานด้วย adapter หมายเหตุ 10 Register แรกจะเป็นการกำหนดตัวอักษรและ Screen format ค่าเหล่านี้ต้องถูก set ไว้เพื่อใช้ในการกำเนิด Timing Interral ที่เหมาะสมให้แก่ Monitor. Vedio Timing จะถูกโปรแกรมใน Term ของ Character Time ซึ่งโดยทั่วไปแล้ว จะมีค่าเป็น 8 เท่าของ Horizontal dimension และเปลี่ยนแปลงไปตามการ Program ตัวเลขและ Scan line ต่อตัวอักษรในแต่ละ Vertical dimension

1.3.4 Programming the Mode Select Register

Mode Select Register คือ Register ที่เขียนได้เท่านั้นขนาด 6 Bit ทำหน้าที่ควบคุมรูปแบบ Screen display แต่ละบิทจะควบคุมรูปแบบการทำงานแบบใดแบบหนึ่งของ display electronics เมื่อรวมกันทั้งหมดแล้วมันจะสร้าง Mode การทำงานพื้นฐานของ Adapter ดังต่อไปนี้

Bit 0 เลือก clock dot ที่เป็นรูปแบบใดรูปแบบหนึ่งอาจจะเป็น 7 MHz หรือ 14 MHz ซึ่งจะเป็นตัวกำหนดความเร็วของข้อมูลที่ออกจาก character generator และส่งไปยังจอภาพ เมื่อข้อมูลถูกอ่านออกจาก display buffer แล้ว

A = 1 เลือก Mode 80x25 text Mode

A = 0 เลือก Mode 40x25 text Mode

Bit 1 จะเลือก Mode การทำงานระหว่าง Text และ graphic

A = 1 เลือก 320x200 graphic Mode

A = 0 เลือก Text Mode

Bit 2 เลือกโหมดสีหรือขาวดำ

A = 1 เลือก Mode สีขาวดำ

A = 0 เลือก Mode สี

Bit 3 เลือกบังคับให้ Video Signal แสดงผลออกจอภาพหรือไม่แสดงผลออกจอภาพ

A = 1 Enable video signal

A = 0 disable video signal

ท่านควรจะ disable video signal เมื่อท่านกำลังเปลี่ยนแปลง Mode หรือทำการโปรแกรม 6845 CR - controller

Bit 4 Enable High resolution graphic

A = 1 เลือก 640 x 200 graphic Mode ในรูปสีขาวดำ

A = 0 disable High resolution graphic

Bit 5 ทำให้เกิดการกระพริบใน Text Mode

A = 1 ทำให้เกิดการกระพริบสำหรับตัวอักษรและจำกัดสี Background เป็น 8 สี ใน Mode นี้ Bit 4 ของ Color Select Register ที่ I/O Address & H 3D9 จะเป็นส่วนควบคุมความเข้มของสี Background ทั้งหมด

A = 0 ไม่ทำให้เกิดการกระพริบและทำให้สีของ Background มีถึง 16 สีบิทนี้ไม่มีผลใน graphic Mode

การรวมกันของบิทต่างเหล่านี้สามารถนำมาใช้ Set Screen Mode โดยเปลี่ยนแปลงการกำหนดบิท แต่มีเพียง 7 แบบของจำนวนทั้งหมดที่ใช้ในทางปฏิบัติทั้ง 7 แบบนี้ถูกรวบรวมไว้แล้วในตาราง 1.7 Mode 1 หรือ Mode 3 จะถูกเลือกโดยอัตโนมัติเมื่อมีการเปิดเครื่องโดยขึ้นอยู่กับการ Set Switch บน System board ค่าที่แสดงใน Mode 4, 5, ๓ จะถูกเขียนใน Mode Select Register โดยใช้ Bios Routine Bios ยังเก็บค่า Mode Register ในสภาวะปัจจุบันและค่า Register สีใน Segment 0 ที่ตำแหน่ง &H465 และ B466 ตามลำดับ เพราะฉะนั้นเราสามารถเปลี่ยน Screen Mode โดยการเปลี่ยนแปลงค่าภายใน Mode Register กับ Interrut Service ที่เตรียมไว้แล้วด้วย Bios Routine

สำหรับในรูปแบบภาษา Basic เราก็สามารถเข้าหา Mode Register โดยการใช้คำสั่ง Basic ที่แสดงไว้แล้วในตาราง 1.7 คำสั่ง width

TABLE 1.6 The 6845 register description

	Register	Units	R/W	40×25 Alpha	80×25 Alpha	320/640 ×200 Graphics	Maximum programmable value
00	Horizontal total	Char	W	56	113	56	255
01	Horizontal displayed	Char	W	40	80	40	255
02	Horizontal sync position	Char	W	45	90	45	255
03	Horizontal sync width	Char	W	10	10	10	15
04	Vertical total	Char row	W	31	31	127	255
05	Vertical total adjust	Scan line	W	6	6	6	31
06	Vertical displayed	Char row	W	25	25	100	127
07	Vertical sync position	Char row	W	28	28	112	127
08	Interlace mode	—	W	2	2	2	3
09	Max scan line address	Scan line	W	7	7	1	31
10	Cursor start	Scan line	W	6	6	6	127
11	Cursor end	Scan line	W	7	7	7	31
12	Start address (high)	—	W	0	0	0	63
13	Start Address (low)	—	W	0	0	0	255
14	Cursor address (high)	—	R/W	—	—	—	63
15	Cursor address (low)	—	R/W	—	—	—	255
16	Light-pen (high)	—	R	—	—	—	63
17	Light-pen (low)	—	R	—	—	—	255

Note: All values are in decimal.

TABLE 1.7 Description of mode select register (&H3D8)

BIOS

Mode	description	Bit	5	4	3	2	1	0	Hex	Dec	BASIC statement(s)
0	40 × 25 Alpha B/W		1	0	1	1	0	0	2C	44	SCREEN 0,0: WIDTH 40
1	40 × 25 Alpha color		1	0	1	0	0	0	28	40	SCREEN 0,1: WIDTH 40
2	80 × 25 Alpha B/W		1	0	1	1	0	1	2D	45	SCREEN 0,0: WIDTH 80
3	80 × 25 Alpha color		1	0	1	0	0	1	29	41	SCREEN 0,1: WIDTH 80
4	320 × 200 Graphics color		(1)	0	1	0	1	0	2A	42	SCREEN 1,0
5	320 × 200 Graphics B/W		(1)	0	1	1	1	0	2E	46	SCREEN 1,1
6	640 × 200 Graphics B/W		(0)	1	1	1	1	0	1E	30	SCREEN 2

Note: The meaning of the mode register bits is as follows:

Bit 0 = High resolution
Bit 1 = Graphics
Bit 2 = Black and white
Bit 3 = Enable video
Bit 4 = 640-dot graphics
Bit 5 = Blink enable

จะ set ความกว้างของจอการแสดงผลตามแนว column และคำสั่ง screen จะทำการเลือก display mode และ lnable หรือ disable colorในการอธิบายคำสั่งทาง graphic เหล่านี้เราจะใช้รูปแบบตามต่อไปนี้ (และรูปแบบดังกล่าวนี้จะถูกนำมาใช้ในคำสั่งทาง graphic อื่น ๆ และจะนำมาอธิบายโดยตลอดเนื้อหานี้)

- ตัวหนังสือเป็นตัวใหญ่คือ Key words และต้องถูกใส่เข้าไปตามที่แสดงไว้
- ตัวหนังสือตัวเล็กจะถูกกำหนดโดยผู้ใช้
- ข้อความในสี่เหลี่ยมปีกกาจะมีหรือไม่มีก็ได้
- เครื่องหมายวรรคตอนทั้งหมดที่นอกเหนือจากสี่เหลี่ยมปีกกา (เช่น ลูกน้ำ, เท่ากับ, ปิดสั้น) ต้องนำมารวมในข้อความนั้นด้วย

ต่อมาคำสั่ง screen, width ควรจะถูกใส่ตามนี้

SCREEN mode, burst : WIDTH size

เมื่อ

Mode = 0สำหรับ Text mode

= 1 สำหรับ Medium resolution

= 2 สำหรับ High resolution

burst= 0 ทำให้ไม่มีสีใน Text mode

= 1ทำให้มีสีใน Medium resolution และไม่มีผลต่อ High Resolution

Size= 40 หรือ 80

สำหรับตัวอย่างคำสั่ง Basic เช่น

SCREEN 0, 1 : WIDTH 80

จะทำการ set จอภาพให้มี 80 column และอยู่ใน Text mode และสีเป็นขาวดำ

1.3.5 Programming the color Select Register

สีใดจะถูกกำหนดโดยค่าแต่ละ Bit ของprimary color (R,G,B) และถ้ามีบิทที่ 4 ก็คือความเข้ม เราสามารถเรียกสีรวมนี้ว่า IRGB color primary color นี้สามารถนำมาผสมกันได้รูปแบบรวม 8 อย่าง ซึ่งแสดงใน

ตาราง 1.8 (หมายเหตุ ถ้า Primary color ทั้ง 3 สีนำมารวมกันจะทำให้เกิดแสงสีขาวขึ้น) ในแต่ละคู่สี Primary color จะทำให้เกิด Comple - mentary color คือ สี cyan, magenta, yellow ส่วน Intensity จะทำให้สีเหล่านี้ไม่มีสีที่เข้มจนเกินไป สีดำจะมีอยู่ในความสว่างบางส่วน ตาราง 1.8 ยังแสดงให้เห็น 16 สี IRGB มาตรฐานที่มีอยู่บนเครื่อง PC (หมายเหตุ เราจะใช้เพียง 4Bit ของหน่วยความจำที่จะแทนสีของแต่ละสีทั้ง 16 สีเหล่านี้ จากการสังเกตุเมื่อค่าที่เพิ่มขึ้นของ Primary color จะเหมาะสมกับค่าที่เพิ่มขึ้นของ code ของสีเหล่านี้ สำหรับตำแหน่ง เช่น cyan คือส่วนผสมของแสงสีน้ำเงินและแสงสีเขียว เพราะฉะนั้นสีนี้จะแทนด้วยค่า Binary code ดังนี้ คือ 0001 (blue) + 0010(green) = 0011 (cyan)

Color Selection Scheme

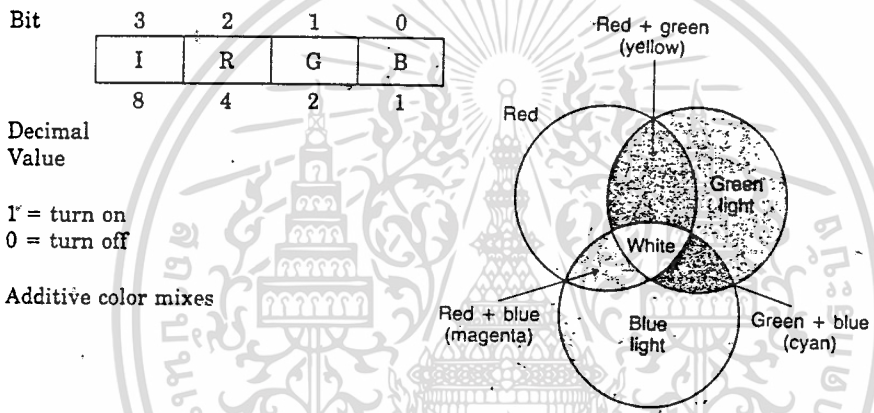
display buffer และ Color Select Register ถูกนำมาใช้ร่วมกันในการควบคุมสีทั้งหมดจอภาพทั้งใน Mode Text และ Mode graphic

ในการแสดงผลตัวอักษรใน Text mode เราจะกำหนด Parameter ได้ 3 แบบ คือ foregroung, Background,border ตามที่แสดงไว้ในรูป 12 foreground ก็คือ ตัวอักษรของมันเอง และ Background ก็คือสีล้อมรอบตัวอักษรนั้นและ Barder ก็คือขอบเขตด้านนอกของพื้นที่ Text display buffer กำหนดสี forground และ Background ของแต่ละตัวอักษร Color Select Register ทำการเลือกสีของ Barder เหตุผลอย่างหนึ่งของเครื่องคอมพิวเตอร์ เช่น IBM PC ที่ต้องมี Barder ก็คือว่าให้คอมพิวเตอร์เหล่านี้ทำงานได้เหมือนกับ color television และต้องใช้วิธีการของ oversceemim

ใน graphic Mode320x 200 เราอาจจะกำหนด Parameter ที่เกี่ยวข้องกับสีได้ 2 แบบคือ background และ Palelte Background คือสีของ Pixel ที่ Turn off แล้วใน Pallelte นั้นจะมีสีให้เลือก 2 กลุ่มแต่ละกลุ่มมีสีอยู่ 3 สี จะทำงานเมื่อ Pixel ถูก Turn on และแต่ละ Pixel บนจอภาพจะถูกกำหนดด้วยตัวเลข 0 ถึง 3 รูป 1.13 แสดงให้เห็น code ตัวเลขที่แทน

TABLE 1.8 IRGB colors

Color code number	I	R	G	B	Color name	Composition
0	0	0	0	0	Black	
1	0	0	0	1	Blue	Blue
2	0	0	1	0	Green	Green
3	0	0	1	1	Cyan	Green + blue
4	0	1	0	0	Red	Red
5	0	1	0	1	Magenta	Red + blue
6	0	1	1	0	Brown	Red + green
7	0	1	1	1	White (light gray)	Red + green + blue
8	1	0	0	0	Dark gray	Int
9	1	0	0	1	Light blue	Int + blue
10	1	0	1	0	Light green	Int + green
11	1	0	1	1	Light cyan	Int + green + blue
12	1	1	0	0	Light red	Int + red
13	1	1	0	1	Light magenta	Int + red + blue
14	1	1	1	0	Yellow	Int + red + green
15	1	1	1	1	Intense white	Int + red + green + blue



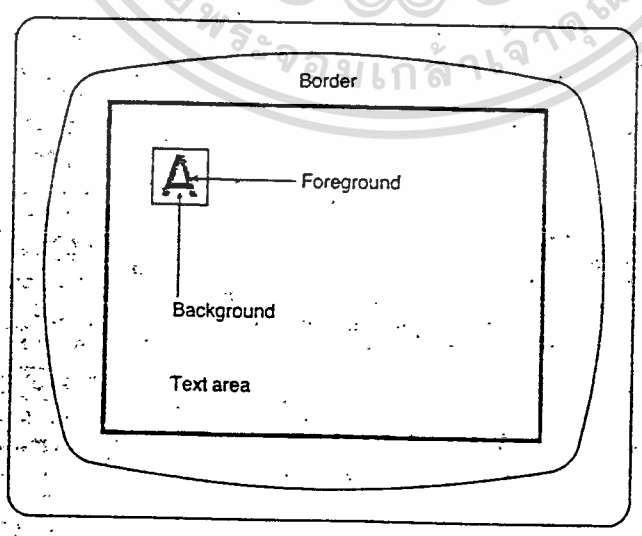


FIGURE 1.12 Color selection in text mode

ที่สีบนจอภาพในรูปแบบของ Palelte ที่แตกต่างกัน 2 กลุ่มนี้ จากตารางนี้เราสามารถเห็นได้ว่าตัวเลข 0 จะถูกกำหนดให้เป็นสี Back ground แต่ตัวเลขตัวอื่นก็มีสีให้เลือกอีก 2 สีตามแต่การเลือก Palelte display buffer จะกำเนิด code ตัวเลขสี (0 ถึง 3) สำหรับแต่ละ Pixel ต่อมา Color Select Register จะกำหนดสีของ background สำหรับ Pixel ที่ Turn off และตัวเลข Palelte (ระหว่าง 0 ถึง 1) สำหรับ Pixel ที่ Turn on

ใน Mode graphic 640 x 200 แต่ละ Pixel ที่ Turn on จะถูกกำหนดให้มีเพียงสี foreground เพียงสีเดียวเท่านั้น display buffer จะเป็นตัวกำหนดว่า Pixel นั้นจะ turn on หรือไม่ต่อจากนั้น Color Select Register จะเลือกสี foreground ของแต่ละ Pixel ที่เราได้ turn on แล้ว

ในตอนนี้เราแสดงวิธีการที่ Color Select Register สามารถกำหนดสีใน Mode text และ Mode graphic ได้อย่างไร (เราจะแสดงวิธีการที่ display buffer จัดการเกี่ยวกับรายละเอียดของสีในบทที่ 2) Color Select Register ก็คือ Register ที่เขียนได้เท่านั้นและมีขนาด 6 Bit พร้อมกับ I/O Port Address &H 3D9 (985 ในเลขฐาน 10) มันสามารถถูกเขียนโดยใช้คำสั่ง out ของ CPU 8088 เรายังสามารถเข้า Register เหล่านี้ได้โดยใช้คำสั่ง out ในภาษา basic ตาราง 1.9 กำหนดค่าที่จำเป็นต้องถูกกำหนดไว้ใน Color Select Register เพื่อควบคุมตำแหน่งสีของการทำงานใน Mode ต่าง ๆ (หมายเหตุโดยเฉพาะในเครื่อง PC จะไม่ Mode High resolution color ในภาษา Basic) แต่เราสามารถสร้างสี foreground โดยการเพิ่มคำสั่งนี้เข้าไปในโปรแกรมตามนี้

```
OUT &HD 39, n
```

เมื่อ n คือสีจาก 1 ถึง 15 ที่ให้ไว้ในตาราง 1.8 สำหรับตัวอย่างเช่น

```
OUT &HD39, 2
```

จะทำให้สี foreground เป็นสีเขียวใน Mode High resolution graphic

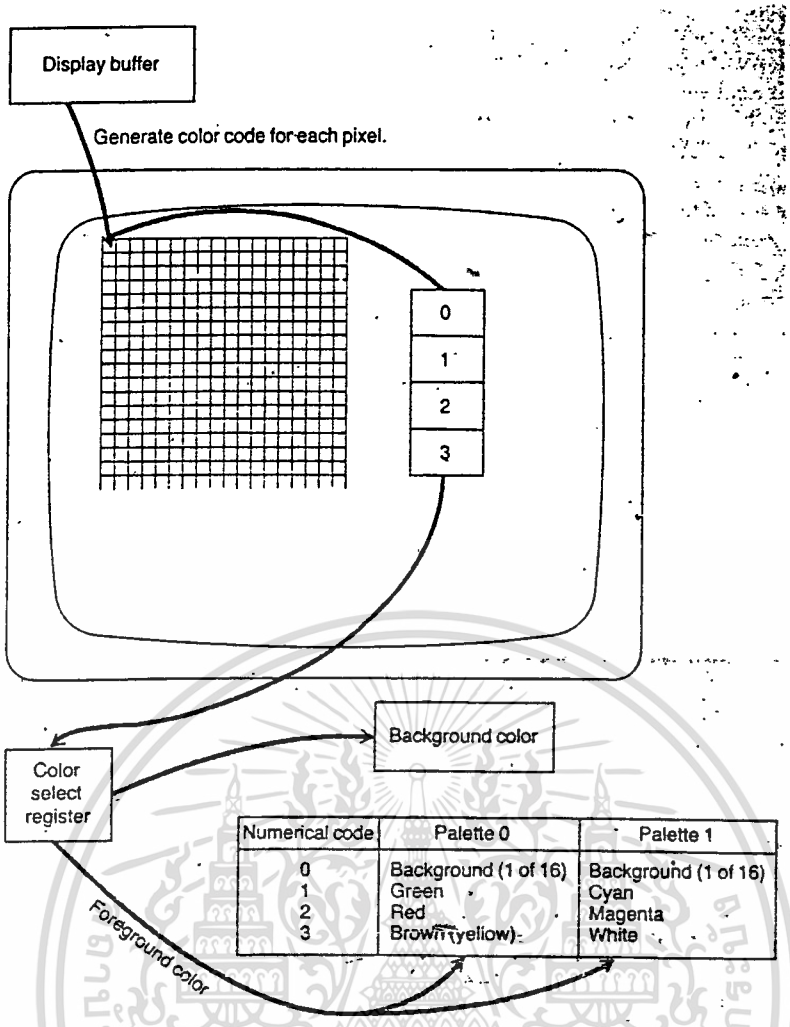
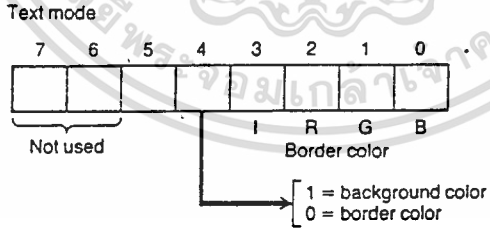
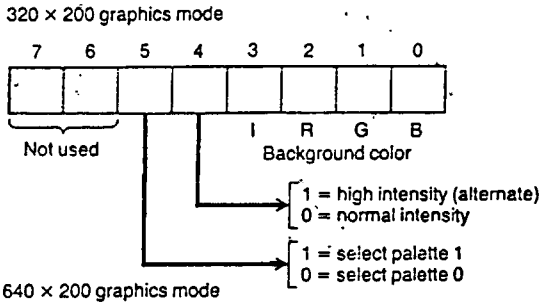
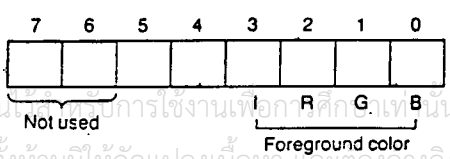


FIGURE 1.13 Color selection in 320 × 200 mode

TABLE 1.9 Description of the color select register







เราจะสิ้นสุดบทนี้โดยการสรุปลำดับของเหตุการณ์ต่าง ๆ ในการโปรแกรม Color / graphic Adapter

1. กำหนด Mode การทำงาน
2. Reset "video enable" bit ใน Mode Select Register
3. Program 6845 เพื่อเลือก mode
4. Program Mode Color Selector รวมทั้งมีการ enable Video

บทที่ 2

โดยเนื้อหาสำคัญในบทนี้จะเกี่ยวข้องกับระบบ Software ของ IBM PC ระบบ Software คือกลุ่มของคำสั่งทาง graphic ที่เขียนโดย Application Program ในการกำหนดภาพบนจอภาพ และยังจัดการเกี่ยวกับการทำงานภายใน graphical โดยลำดับแรกเราจะต้องตรวจสอบวิธีการที่เราสามารถที่จะเข้าไปใน SCREEN Memory และควบคุม Screen memory ในการสร้างภาพในกรรมวิธีต่าง ๆ เราจะแสดงให้เห็นด้วยคำสั่งเกี่ยวกับ graphic ที่มีอยู่บนเครื่อง IBM PC ต่อมาเราจึงกล่าวถึงวิธีการที่จะสร้างภาพอย่างรวดเร็วโดยการเชื่อมโยงกับ Subroutine ภาษาเครื่อง

2.1 Memory Mapped Graphic in text mode

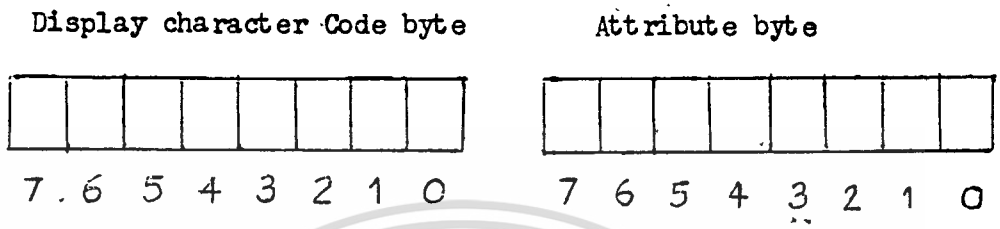
ตามที่กล่าวมาแล้วในบทที่ 1.2.4 รูปภาพบนเครื่อง PC ถูกสร้างขึ้นโดยการควบคุม Pixel ผ่านทาง Memory Mapped display โดยที่แต่ละตำแหน่ง Memory Mapped display จะมีลักษณะเข้ากับตำแหน่งที่เหมาะสมบนตำแหน่งของจอภาพ ด้วยกรรมวิธี Memory Mapped Screen เราสามารถเขียนหรืออ่านข้อมูลที่แสดงผลออกทางจอภาพได้โดยตรงอย่างง่ายโดยการเข้าสู่ display Memory ที่อยู่ภายในหน่วยความจำ display buffer ขนาด 16 K ของ Color/graphic Adapter

ในการสร้างระบบที่มีความสามารถทางด้าน color และ graphic บนเครื่อง PCให้ตรงกับความต้องการของเรา มีความจำเป็นที่ต้องรู้วิธีการเข้าหาและควบคุม Screen Memory display ซึ่งในแต่ละตำแหน่งของหน่วยความจำจะมีข้อมูลขนาด 1Byte และข้อมูลขนาด 1 ไบท์ ที่แสดงผลทางจอภาพยังขึ้นอยู่กับ Mode การทำงานเกี่ยวกับ graphic อย่างไร

2.2.1 Memory Requirment

ลำดับแรกเราจะเริ่มพิจารณาใน Mode Text ก่อน สำหรับแต่ละตำแหน่งของจอภาพจะใช้เนื้อที่ในหน่วยความจำ 2 ไบท์ และข้อมูล 2 ไบท์นี้จะเป็นตัวกำหนดว่าตัวอักษรอะไรที่ต้องนำออกแสดงผลและสีจะแสดงออกเป็น

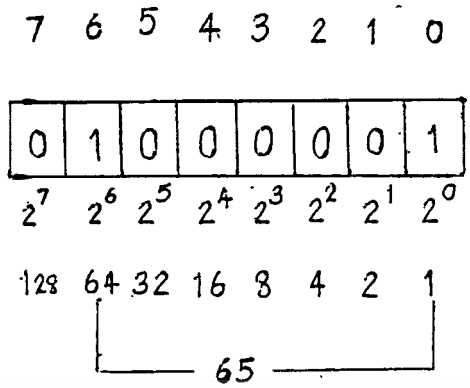
สีอย่างไร เพราะฉะนั้นทุกตำแหน่งตัวอักษรที่แสดงผลจะถูกกำหนดด้วยรูปแบบ 2 Byte ดังนี้



มีตัวอักษร 2000 ตัวบนจอภาพใน Mode 80 Column (80x 25) เพราะว่าแต่ละตัวอักษรใช้เนื้อที่ 2 Byte ดังนั้นจึงต้องใช้ Screen Memory ขนาด 4000 byte ใน Text mode (2000 byte สำหรับ 40 x25 mode).

<u>display character code byte</u>

ในแต่ละ Byte มี 8 ตำแหน่งและสามารถเก็บค่าที่แตกต่างกันได้ถึง 25 ค่า หรือจะแทนด้วยตัวอักษรได้ 256 แบบที่แตกต่างกัน เมื่อ byte เหล่านี้ถูกใช้ในการเก็บค่าตัวอักษร จึงต้องมีการกำหนด code เป็นตัวเลขต่างๆ เพื่อแทนลักษณะของตัวอักษรแต่ละตัว code ที่ใช้โดยทั่วไปโดยส่วนมากแล้วเป็น ASC II เพราะว่า ASC II มีเพียง 128 code เท่านั้น และค่าตัวอักษรทั้ง 128 ตัวก็จะเก็บอยู่ในโค๊ดเหล่านี้ และจะมีที่ที่หนึ่งสำหรับเก็บอีก 128 ตัวที่เหลือ และโค๊ดที่เหลือเหล่านี้ ในบางครั้งเรียกว่า Extend ASC II ASC II code นั้นจะถูกกำหนดเป็นมาตรฐาน แต่ Extend ASC II จะเปลี่ยนแปลงไปตามรูปแบบของเครื่องคอมพิวเตอร์ ตาราง 2.1 แสดงให้เห็นความสัมพันธ์ระหว่าง ASC II code และตัวอักษรที่อยู่บนเครื่อง PC(ดูภาพผนวก A ของตารางการเปลี่ยนแปลงเลขฐานสิบหก) สำหรับตัวอย่างถ้าเรามีรูปแบบของ Bit ใน 1 Byte ดังนี้ 0100 0001 เป็น code ของตัวอักษร เราสามารถเปลี่ยนโค๊ดนี้เป็นค่าเลขฐานสิบ= 65 ตามนี้คือ



จากตาราง 2.1 เรารู้ว่าตัวอักษร A จะมีค่าเลขฐานสิบ = 65 ∴ การกำหนดค่า 65 ไปในตำแหน่ง code ของตัวอักษรก็จะมีค่าเท่ากับการเก็บค่า A ลงในหน่วยความจำ

Attribute byte

นอกเหนือจากโคดตัวอักษรขนาด 8 บิทแล้ว ทุก ๆ ตำแหน่งตัวอักษรบนจอภาพก็ยังมี code attribute ขนาด 8 บิท ใน code นี้จะเป็นตัวกำหนดสีของตัวอักษรและพื้นสีของตัวอักษร, ความเข้มของสีและตัวอักษรที่แสดงผลว่าเป็นตัวกระพริบหรือไม่กระพริบ (ดูรูป 2.1) เพื่อให้เห็นว่า code เหล่านี้ทำอะไรบ้าง ลำดับแรกเราต้องรู้ว่า แต่ละสัญญาลักษณ์ตัวอักษรจะประกอบไปด้วย ตัวอักษรของมันเอง (foreground) และเนื้อที่สี่เหลี่ยมผืนผ้ารอบ ๆ ตัวอักษรนั้น (Blackground) ใน 4 บิทต่ำ (0,1,2,3) ของ Attribyte Byte จะเป็นตัวกำหนดสีของตัวอักษร (foreground) 3 บิท (4,5,6) จะเป็นตัวกำหนดสีของ Blackground ของตัวอักษร ส่วนตัวที่ 7 นั้นจะเป็นตัวกำหนดให้ตัวอักษรกระพริบหรือไม่กระพริบ รูป 2.1 แสดงให้เห็นรูปแบบของ Attribute byte สำหรับตัวอักษรแสดงผลในรูปแบบสี

แต่ละบิทใน Attribute byte นี้โดยแท้จริงจะนำไปควบคุมสีแต่ละบิท (R,G,B) ต่อมาก็จะถูกนำมารวมกันโดยวงจร Video เพื่อจะทำให้เกิดสี foreground และ Blackground (ส่วนวงจรที่ทำให้เป็นสีหรือวงจร filter นั้นจะขึ้นอยู่กับวิธีการลบแสงสี เมื่อสีทั้งหมดถูกรวมโดยวิธีนี้จะทำให้สีมีความเข้มน้อยลง).

TABLE 2.1. ASCII character codes for the PC

DECIMAL VALUE	↓	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
→	HEXADECIMAL VALUE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	BLANK (NULL)	►	BLANK SPACE	0	@	P	‘	p	Ç	É	á	▒	└	╨	∝	≡
1	1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	β	±
2	2	☻	↕	"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
3	3	♥	‼	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
5	5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
6	6	♠	▬	&	6	F	V	f	v	å	û	ª	╢	╞	╓	μ	÷
7	7	•	↨	'	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
8	8	◘	↑	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
9	9	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	•
10	A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
11	B	♂	←	+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
12	C	♀	∟	,	<	L	\	l	¦	î	£	¼	╝	╠	▄	∞	n
13	D	♪	↔	-	=	M	]	m	}	ì	¥	¡	╜	═	▌	φ	2
14	E	♫	▲	.	>	N	^	n	~	Ä	₧	«	╛	╬	▐	∈	■
15	F	☼	▼	/	?	O	_	o	△	Å	ƒ	»	┐	╧	▀	∩	BLANK 'FF'

In alpha modes, each display position is defined by a character code/attribute pair. The character code is always the even-addressed byte, the attribute is the next higher odd-addressed byte.

Attribute byte

7	6	5	4	3	2	1	0
I/ B	Red	Grn	Blu	I	Red	Grn	Blu
Background				Foreground			

Attribute byte format

Bit 0–3	Foreground color
Bits 4–6	Background color
Bit 7	Background intensity

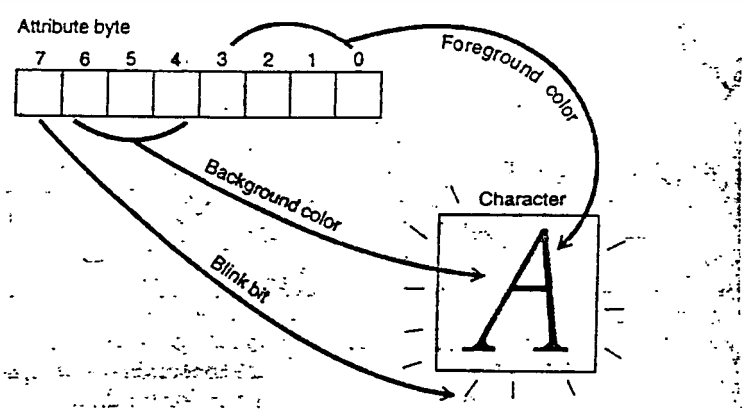


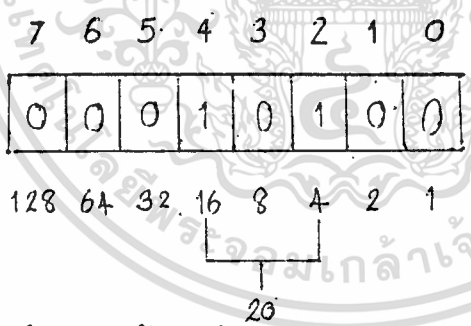
FIGURE 2.1 Attribute byte format used in text mode

เพราะว่า 4 บิทที่ถูกแบ่งไว้เพื่อเป็นตัวกำหนดสีของ foreground ทั้ง 4 บิทนี้สามารถแทนด้วยตัวเลขต่าง ๆ กันได้ 16 ตัวคือ จาก 0 ถึง 15 ลักษณะของโค๊ดสีเหล่านี้จะเหมือนกับโค๊ดสีที่แสดงในตาราง 1.8 เพราะฉะนั้นสีทั้ง 16 สีนี้จะถูกนำมาใช้กับตัวอักษร foreground และ 3 Bit ต่อมาจะใช้แทนสีของ Blackground ของตัวอักษร จึงทำให้ได้สีของ Blackground ที่แตกต่าง ๆ กัน 8 สี และเนื่องจากวิธีการแบ่งบิทเหล่านี้ทำให้ได้สูตรที่นำมาใช้ในการคำนวณหา Color code Attribute ของแต่ละตำแหน่งตัวอักษร ดังนี้ Attribute byte = 128 (blink) + 16 (blackground color code)

+ foreground color code (2.1)

เมื่อบิทที่แสดงการ Blink มีค่า 0 จะแทนตัวอักษรที่ไม่มีการกระพริบและ 1 จะแทนตัวอักษรที่มีการกระพริบ สำหรับตัวอย่าง ถ้าเราต้องการเขียนตัวอักษรที่ไม่กระพริบที่มีตัวอักษรเป็นสีแดง และสีพื้นรอบ ๆ เป็นสีน้ำเงิน ค่าของ Attribute จะมีค่าดังนี้

Attribute Value = 128 (0) + 16 (1) + 4 = 20



ดังที่กล่าวมาแล้วในส่วน 1.3.5 Color Select Register จะควบคุมสีขอบเขตของจอภาพใน Mode Text และจะเลือกได้ 1 ใน 16 สี ตามที่แสดงไว้ในตาราง 1.8 เพราะว่า Color Select Register นี้จะมี Memory เป็นของตัวเองในการเก็บค่าข้อมูลขอบเขตสี และข้อมูลของสีดังกล่าวนี้จะไม่ถูก Map เข้าไปใน display buffer

เมื่อเราเปิดเครื่อง Computer Color Graphic Adapter จะทำการ set ความเข้มของแสดงทั้งจอภาพให้อยู่ในสภาวะปกติ คือ ตัวอักษรที่ไม่กระพริบสีขาว บนพื้นสีดำ อย่างไรก็ตามเราก็สามารถกำหนด Attribute ใหม่ได้ โดยที่เราต้องเปลี่ยนแปลงค่าที่ set ไว้เหล่านี้ที่ตำแหน่งที่เลือกไว้บนจอภาพ เราจะ

แสดงให้เห็นการทำในรูปแบบนี้ในตอนหน้าต่อไป

2.1.2 Memory Mapping

แต่ละตำแหน่งตัวอักษรบนจอภาพใน Text mode จะถูกกำหนดโดยข้อมูล 2 Byte ในเนื้อที่ส่วนหนึ่งของ Memory ที่เรียกว่า display buffer chips ที่ประกอบด้วยหน่วยความจำส่วนนี้จะอยู่บน Color / graphic Adapter ซึ่งหมายความว่า Adapter จะทำหน้าที่เป็นหน่วยความจำเพิ่มเติมขนาด 16K byte ของระบบหน่วยความจำหลัก

Offset

โดยทั่วไปของ computer Arithmetic และ Notation (เครื่องหมาย) ส่วนประกอบของข้อมูล Byte แรกนั้นจะเริ่มต้นที่ตำแหน่งศูนย์ และจะเรียกตำแหน่งนี้ว่าเป็น Zero Origin หรือ Base การนับจะเริ่มต้นจาก 0 เมื่อค่าของตำแหน่งใน Memory นั้นสัมพันธ์ให้เป็นจุดตั้งต้นอีกจุดหนึ่ง สำหรับ Color Graphic แล้ว หน่วยความจำจะเริ่มขึ้นที่ตำแหน่ง &HB 8000 (ดูตาราง 4.1) ตำแหน่งที่สัมพันธ์กับฐานนี้จะมีความหมายว่าเป็น offset การเริ่มต้นจากตำแหน่งฐานใดในหน่วยความจำ , Byte ที่หนึ่งจะอยู่ที่offset 0 และByte ที่ 1 ก็จะอยู่ที่ offset 1

The Mapping

ใน Text mode 2 byte แรกของ display buffer จะถูก Map ให้เข้ากับมุมบนด้านซ้ายมือของจอภาพ (คือตำแหน่งที่ตัวอักษรตัวแรกตั้งอยู่) 2 ไบท์ต่อมาก็จะถูก Map ให้เข้ากับตำแหน่งตัวอักษรถัดไปคือ ไปยังตำแหน่งขวามือของตำแหน่งตัวอักษรตัวแรก กรรมวิธีดังกล่าวนี้จะดำเนินต่อไปตลอดจนสิ้นสุดแถวแรก และเมื่อถึงตัวอักษรตัวสุดท้าย การ Map หน่วยความจำจะต้องกลับมาตั้งต้นที่ตำแหน่ง column แรกของบรรทัดต่อไปและจะถูก Map ให้เข้ากับ byte คู่ต่อมาของ column สุดท้ายของบรรทัดที่แล้ว วิธีการดังกล่าวนี้จะดำเนินต่อไปทีละแถวจนกระทั่งถึงแถวล่างสุดของจอภาพ กรรมวิธีที่สมบูรณ์ของ Memory

Mapping ถูกแสดงให้เห็นดังรูป 2.2 สังเกตุว่าตัวอักษรตัวแรกของไบท์แต่ละคู่ (ซึ่งประกอบด้วยตัวอักษร 1 ตัว) จะมีค่าตำแหน่ง Address เป็นคู่ในขณะที่ Attribute byte มีค่าตำแหน่ง Address เป็นคี่ (จุดเริ่มต้นของ display buffer จะถูกโปรแกรมโดยผ่าน IC 6845 CRT controller และต้องมีตำแหน่ง Address เป็นคู่)

สำหรับในแต่ละตำแหน่งตัวอักษร เราสามารถใช้สูตรต่อไปนี้ในการคำนวณหาตำแหน่ง Bytes ต่างที่อยู่ภายใน Screen Memory ได้ดังนี้

character Address ไบท์แรก = 2 width (ROW-1) + 2(column-1)

Attribute Address ไบท์สอง = First byte + 1 (2.2)

เมื่อ Width = 40 หรือ 80

ในที่นี้เราใช้รูปแบบของ IBM PC โดยการกำหนดแต่ละแถวแต่ละ column เริ่มต้นที่ตำแหน่ง 1 ไม่ใช่ตำแหน่ง 0 อย่างไรก็ตามสูตรหน่วยความจำข้างบนนี้ให้ค่าเป็น Memory offsets นั่นคือ จำนวน Byte จะเริ่มนับจากจุดเริ่มต้นของส่วนหน่วยความจำ ค่า Address เหล่านี้ของถูกสร้างขึ้น เพื่อที่จะนำไปใช้ในการเข้าหาข้อมูลของ Memory ในส่วนนั้น ๆ

2.1.3 Accesing Memory from Basic

โดยการใช้คำสั่ง peek,poke เป็นเครื่องมือในการสำรวจ Screen Memory Map คำสั่งPeek และ Poke จะเข้าสู่ตำแหน่งหน่วยความจำสัมพันธ์ที่ระบุไว้เป็นส่วนของ Segment บนเครื่อง IBM PC คำสั่ง DEF SEG จะ set ค่า Segment ปัจจุบันให้เป็นจุดที่เริ่มต้นของ graphic Screen คำสั่ง Peek และ Poke จะอ้างถึงตำแหน่ง Address จริงในการทำงานของตัวมันเป็นค่า offset หนึ่งภายในส่วน Segment โดยเฉพาะคำสั่ง Peek (n) จะส่งค่าข้อมูล 1 Byte ที่อ่านจากตำแหน่ง Memory (n) และคำสั่ง Poke n,m จะเขียนข้อมูล 1 Byte เข้าไปตำแหน่งหน่วยความจำn

เมื่อ n จะแทนค่า offset จาก Segment ในสภาวะปัจจุบัน (ตามที่กำหนดโดยคำสั่ง DEF-SEG) และ m จะแทน DATA ที่ถูกเขียนลงไป ค่าของ n และ m สามารถกำหนดได้ทั้งในรูปแบบฐานสิบและฐานสิบหก

ตัวอย่าง สมมุติว่าเราต้องการเขียนตัวอักษร A ในแถวที่ 3 คอลัมภ์ที่ 4 ของจอภาพ ตัวอักษร A จะเป็นสีแดง และพื้นเป็นสีน้ำเงิน เราสามารถจะทำได้ตามนี้

ขั้นตอนที่ 1 กำแหน่งของAddress ที่จะใส่ค่า "A" จะใช้สมการ 2.2

Character Address = 80 (3-1) + 2 (4-1) = 166

Attribute Address = 166 + 1 = 167

ขั้นตอนที่ 2 กำหนดค่า code ของตัวอักษรและค่า Attribute โดยใช้ตาราง 2.1 และสมการ (2.1)

ค่า ASC II code ของ "A" = 65 ตาราง 2.1

ไม่กระพริบ

ค่าAttribute ตัวอักษรสีแดง = 20 สมการ 2.1

พื้นสีน้ำเงิน

ขั้นตอนที่ 3 ตั้งค่า Segment ปัจจุบันให้เป็นจุดเริ่มต้นของ Screen Memory

DEF seg = &HB 8000

(display Buffer อยู่ที่ตำแหน่ง Address B8000 ฐานสิบหก เพราะว่า Segment) จะถูกกำหนดในรูปแบบ 16 Byte ตัวอักษรตัวสุดท้ายจะถูกตัดทิ้งไปเมื่อกำหนดด้วย DEF-SEG

เมื่อไรก็ตามที่เรา Poke เข้าไปในหน่วยความจำภายใน ในช่วงหน่วยความจำ B 800:0000 ถึง B 800 : 3 FFF (0 ถึง 16384) จะทำให้เกิดผลการเปลี่ยนแปลงบนจอภาพขึ้น

ตัวอย่างโปรแกรมภาษา Basic ต่อไปนี้ แสดงวิธีการเข้าหาข้อมูลและข้อมูลจากScreen Memory ใน Text mode

```
10 CLS                          ; Clear the Screen
20 Screen 0 : width 40          ; Set the Screen in text mode
30 DEF SEG = &HB800             ; Define Current Segment
40 Poke 166, 65                 ; write "A"
50 Poke 167, 20                 ; set the Attribute coloe
60 A = Peek (166)               ; Read character "A"
70 B = Peek (167)               ; Read the Attribute
80 Poke 1832, A                 ; write Another "A"
90 Poke 1833; B                 ; set the Attribute
100 END
```

เมื่อโปรแกรมนี้ถูก execute เราจะมองเห็นตัวอักษร A ที่ไม่กระพริบพิมพ์อยู่ที่จอภาพในตำแหน่งทั้ง 2 ดังรูป 2.3

เราสามารถเพิ่มขอบเขตสีในจอภาพ (ภายนอกเนื้อที่ Text โดยการเพิ่มคำสั่งต่อไปนี้ลงในโปรแกรม

```
OUT &H3D9, 14 ; yellow Border
```

Using the color statement

ในภาษา Basic คำสั่ง color เป็นคำสั่งที่ใช้เลือกสีของ foreground และ Blackgroun และ Border ได้โดยใช้คำสั่ง color เพียงคำสั่งเดียว รูปแบบของข้อความในประโยคคำสั่งจะเป็นดังนี้

เมื่อ foreground = ตัวเลขที่แสดงถึงค่าโค๊ดสีของ forego-und อยู่ในช่วง 0 ถึง 15 (บวกด้วย 16 เข้ากับตัวเลข foreground จะทำให้ตัวอักษรกระพริบได้)

blackground = ตัวเลขที่แสดงค่าโค๊ดสีของ Balckground อยู่ในช่วง 0 ถึง 7

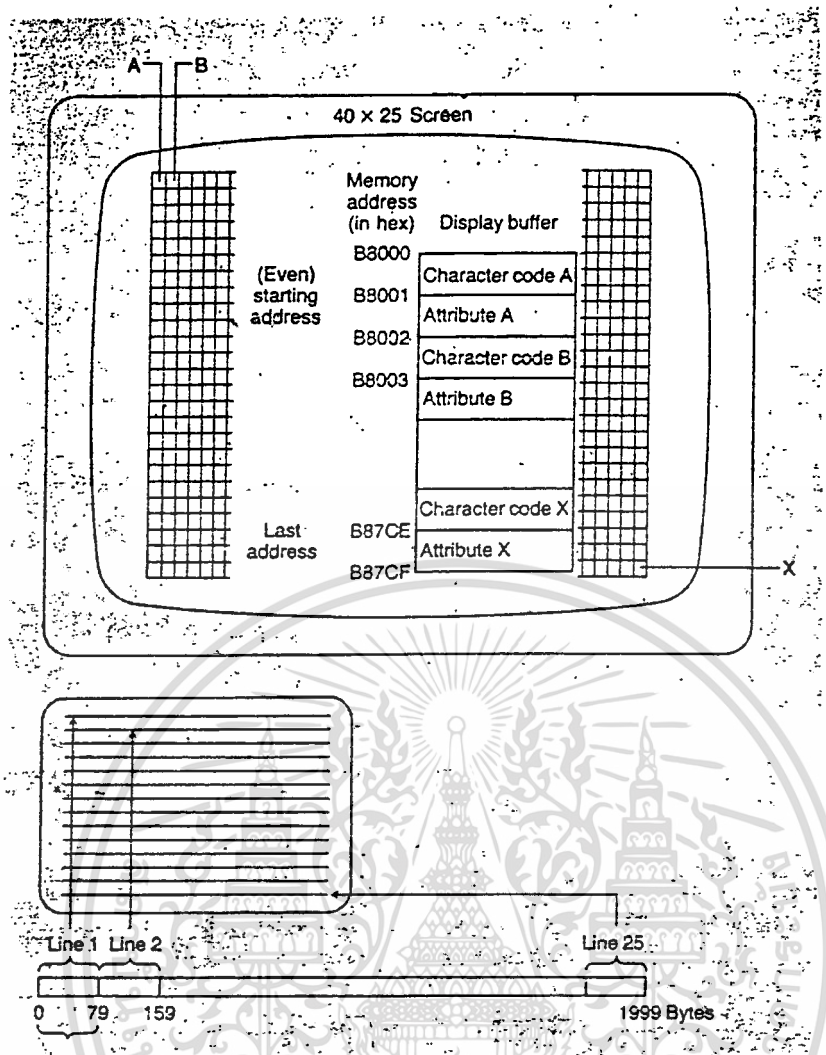


FIGURE 2.2 Display memory map in text mode. (Example of a 40 × 25 screen.)

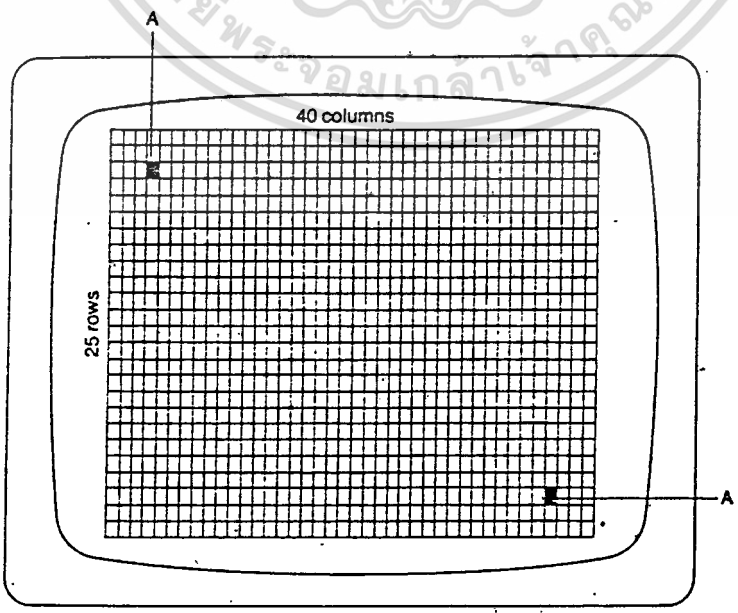


FIGURE 2.3 Memory-mapped character display on the IBM PC

border = ตัวเลขที่แสดงค่าโคดสีของ Border (ขอบจอ) อยู่ในช่วง 0 ถึง 15

ตัวเลขข้างบนทั้งหมดควรเลือกใช้ให้ถูกต้องกับโคดสีที่ให้ไว้ในตาราง 1.8 ถ้าใช้กับ Color Graphic Adapter เราก็สามารถทำให้ตัวอักษรกระพริบได้ โดยการบวก 16 เข้ากับค่าสี forground

สำหรับตัวอย่างคำสั่ง color 14 16, 2, 4 จะทำการ set ค่าสีของ foreground ให้เป็นสีเหลืองกระพริบได้, พื้นสีน้ำเงิน, ขอบจอเป็นสีแดง ค่าต่างที่กำหนดนี้ได้ถูกรวบรวมไว้แล้วในตาราง 2.2

2.1.4 Multiple Text Page

ตามที่ได้เห็นมาแล้วในส่วนที่แล้ว การ Map หน่วยความจำจะใช้หน่วยความจำ 2 Byte ต่อ 1 ตัวอักษร 1 และ Memory Map จะต้องใช้หน่วยความจำถึง 4000 ไบท์ใน Text Mode ขนาด 80 x25 และใน Mode 40 x 25 จะใช้หน่วยความจำ 2000 byte ส่วน display buffer จะมีขนาด 16 K byte และมีขนาดมากเพียงพอสำหรับ text byte ที่มีขนาด 4 ถึง 8 page

PC จะใช้หน่วยความจำส่วนมากนี้ เมื่อหน่วยความจำนี้ไม่ได้นำมาใช้ใน graphic display โดยที่จะนำมาเตรียมไว้ใช้กับ Text Mode ที่มี Multiple Screen Image ในหน่วยความจำ Multiples Images นี้จะเรียกว่า Pages

display page เริ่มนับจาก 0 ถึง 3 สำหรับใน Text Mode 80x25 หรือจะเริ่มนับจาก 0 ถึง 7 ใน Text Mode ขนาด 40x 25 Page 0 จะตั้งอยู่ในจุดเริ่มต้น display Memory แล้วตามด้วย Page อื่น ๆ ต่อมา การเริ่มต้นของแต่ละ Page ถูกตั้งโดยตัวเลขวนรอบในรูปแบบ Binary สำหรับตัวอย่างใน Text Mode 80 x25 แต่ละ Page ต้องการหน่วยความจำ 4000 byte Page0 เริ่มต้นที่ตำแหน่งสัมพันธ์ 0 ในหน่วยความจำ Color graphic (หรือในเลขฐานสิบหกจะอยู่ในตำแหน่ง Address &HB 8000) อย่างไรก็ตาม Page 1 จะไม่เริ่มต้นที่ตำแหน่งที่ติดอยู่กับ Page 0 คือที่ตำแหน่งสัมพันธ์ 4000 แต่ Page 1

จะเริ่มต้นที่ขอบเขต Kilobyte ถัดไปคือ ที่ระยะขจัด 4 Kbyte หรือ 4096 byte (1K = 1024) .'. ฉะนั้นจึงมีหน่วยความจำ 96 Byte ที่ไม่ได้ใช้ในแต่ละ Page รูป 2.4 แสดงให้เห็น Page ทั้งหมดที่วางอยู่ใน display Memory

การควบคุม display Page จะทำได้ในภาษา Basic โดยคำสั่งดังนี้ Screen 0 (brust), , (apage) (vpage)

เมื่อ

0 = ตัวเลขที่แสดง Text Mode ที่แสดงความกว้าง (40 หรือ 80)

burst = ตัวเลขที่แสดงถึงสีใน Text Mode ค่า 0 จะไม่ให้ค่าสี (จะใช้กับภาพสีขาวดำเท่านั้น ค่าตัวเลข 0 ที่ไม่ใช่ 0 นำมาใช้ในภาพสี)

apage ตัวเลขที่แสดงอยู่ในช่วง 0 ถึง 7 สำหรับใน Mode 40x 25 หรือ 0 ถึง 3 สำหรับใน Mode 80x 25 โดย apage จะเลือก page ที่ถูกเขียนลงไปด้วยคำสั่ง output ของ Screen โดย Page ดังกล่าวนี้จะเรียกว่า Active Page

Vpage ตัวเลขที่แสดงอยู่ในย่านเดียวกัน apage ข้างบน โดยที่จะทำการเลือกPage ที่จะนำออกแสดงผลสู่จอภาพ หรือในบางครั้งเรียกว่า Visual Page (Page ที่มองเห็นได้) อาจแตกต่างไปจาก Active Page

สำหรับตัวอย่างคำสั่ง Basic "Screen 0,1,0,0 "คำสั่งนี้จะทำการเลือก Text Mode พร้อมกับมีสี และ set ค่า Active Page และ Visual Page เป็น 0 ถ้ามีการกำหนด apage และ Vpage ขึ้นใหม่ ก็ทำให้ display Page ที่มองเห็นเปลี่ยนแปลงไป โดยการจับต้องส่วนของ Active page และ Visual page เราสามารถแสดงผล page ใด page หนึ่งในขณะที่มีการสร้าง page อื่น ๆ สำหรับตัวอย่างต่อไป คำสั่ง Basic "Screen, 1, 2" จะทำให้ mode และ color คงเดิมในขณะที่มีการ set ค่า Active page เป็น 1 และ Visual page = 2 คำสั่งนี้สามารถทำให้เราปรับค่า Visual page โดยทันทีทันใด

โปรแกรมต่อไปแสดงให้เห็นถึงขั้นตอนในการสร้าง Multiple text mode pageใน Text mode โดยที่เราจะสร้าง ทั้ง 2 ใน mode 40 x25

TABLE 2.2 BASIC color codes in text mode

Code decimal[1]	Color		
	Foreground	Color/effect background	Border[2]
0	Black	Black	Black
1	Blue	Blue	Blue
2	Green	Green	Green
3	Cyan (greenish blue)	Cyan	Cyan
4	Red	Red	Red
5	Magenta (purplish)	Magenta	Magenta
6	Brown	Brown	Brown
7	White	White	White
8	Gray		Gray
9	Light blue		Light blue
10	Light green		Light green
11	Light cyan		Light cyan
12	Light red		Light red
13	Light magenta		Light magenta
14	Yellow		Yellow
15	Bright white		Bright white
16	Black		
17	Blinking blue		
18	Blinking green		
19	Blinking cyan		
20	Blinking red		
21	Blinking magenta		
22	Blinking brown		
23	Blinking white		
24	Blinking gray		
25	Blinking light blue		
26	Blinking light green		
27	Blinking light cyan		
28	Blinking light red		
29	Blinking light magenta		
30	Blinking yellow		
31	Blinking bright white		

[1]Code decimal numbers 16–31 are obtained with bit 7 on.
[2]Controlled by Color Select Register.

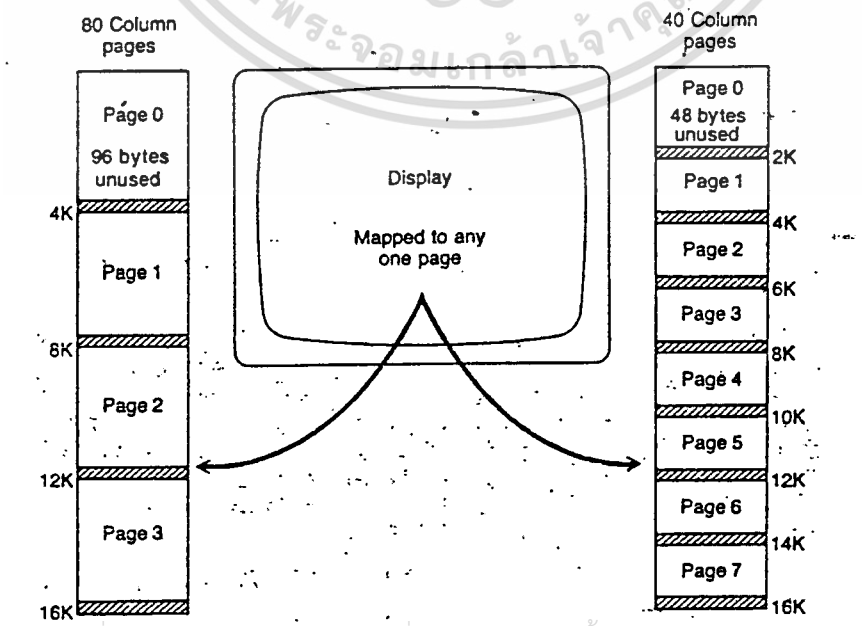


FIGURE 2.4 Multiple text pages in text mode

แต่ละ text ประกอบด้วยข้อความสั้น ๆ และข้อความสั้นเป็นดังต่อไปนี้

Page 0		Page 1
10 J.M. WHITE	10	A.B. BLACK
15		15

แต่ละข้อความเริ่มต้นที่แถวที่ 10 และ column ที่ 15 Program ที่ได้จะเป็นไปตามนี้

```
10 CLS
20 Screen 0,0,0 : width 40          ; set both Active and
30 Screen 0,1,0,0 : color 1,2,14    ; visual page to zero
40 Locate 10,15 :print "J.M.white",; write on to page 0
50 Screen 0,1,1,0                   ; set Active page =1
60 Locate 10,15 :print "A.B.Black"; write on to page 1
70 Screen 0,1,0,0                   ; Display page 0
80 IF Inkey $ = "  " Then 80        ; Pause: Hit anykey to continue
90 Screen 0,1,1,1                   ; Display page 1
```

2.2 Memory-Mapped Graphin in Graphic Mode

2.2.1 Memory Requirement

ลำดับแรกเราจะต้องตรวจสอบหน่วยความจำที่ต้องนำมาใช้ใน High Resolution graphic โดยที่หน่วยความจำทั้งหมดนี้จะมีการกำหนดในรูปแบบ Pixel ซึ่งสามารถกำหนดสภาวะ on หรือ off ได้ เราใช้ Bit เดียวในการควบคุม Pixel 1 Pixel ใน High Resolution Graphic จะมีจุดถึง 640 x 200 หรือ 128,000 Pixel ในการควบคุมจุดทั้งหมดเราจะใช้ 1 Bit ต่อ 1 Pixel , เราก็ต้องใช้ทั้งหมด 128,000 Bit เมื่อขนาด 8 บิทเป็น 1 ไบท์เราก็ต้องการข้อมูล 80 ไบท์ต่อ 1 เส้น และ 16000 ไบท์ต่อจอภาพ หน่วยความจำที่ใช้นี้เกือบจะเต็มหน่วยความจำของ Color / graphic Adapter และจะมีหน่วยความจำเหลืออยู่ 384 ไบท์ที่ไม่ได้ใช้

(16 K = 16384 ไบท์) หน่วยความจำที่ไม่ได้ใช้นี้จะต้องถูกตัดทิ้งไป ก็จึงไม่มีการกำหนดรูปแบบสีให้แก่ Pixel เหล่านี้ (นอกเหนือไปจากการกำหนดสีขาวดำได้โดยการทำให้ Pixel Turn on หรือ Turn off)

ส่วนใน Medium resolution Mode จะมีจำนวน Pixel เพียงครึ่งเดียวของ High resolution graphic mode โดยตลอดจอภาพ เพราะว่าใน resolution ทั้ง 2 แบบ จะใช้หน่วยความจำ display buffer ขนาด 16 K byte ในการแสดงภาพบนจอภาพ เราจึงสามารถใช้เนื้อที่ 2 Bit ต่อ 1 Pixel ก็ได้ 2 Bit (C0,C1) จะถูกนำมาใช้รวมกันและสามารถกำหนดเป็นค่าต่าง ๆ ได้ถึง 4 ค่า (0,1,2,3) จึงทำให้ได้สีทั้งหมด 4 สีใน Medium Resolution Mode รูป 2.5 แสดงให้เห็นวิธีการกำหนด Pixel ทั้งหมดใน display buffer ในแต่ละ graphic mode ได้อย่างไร

2.2.2 Memory Mapping

ใน graphic mode display buffer จะถูกแบ่งออกเป็น 2 Block ละ 8 K จะดูเห็นว่าค่อนข้างแปลกคือว่า เส้น Raster คู่ทั้งหมดจะถูกเก็บภายใน 8 Kbyte แรกของ Screen Memory Mop และเส้น Raster คี่จะอยู่ใน 8 Kbyte ต่อมา ที่ต้องทำในรูปแบบนี้เพราะว่า IC 6845 จะควบคุม CRT วาดเส้นราสเตอร์คู่ทั้งหมดก่อนและเส้นคี่ที่เหลือตามมา โดยทางปฏิบัติจะเรียกวิธีนี้ว่า Interlacing ถ้าเราต้องการที่จะทำ graphic จากภาษาที่ไม่มีคำสั่งทาง graphic เช่น Fortan, Pascal, Assembly โดยที่เราจำเป็นต้องรู้การ Interlacing เมื่อมีการ plot จุดขึ้น

รูป 2.6 ได้รวบรวม graphic Memory Map และ Pixels Format ตัวอย่างใน High Resolution graphic ก็เป็นตัวอย่างหนึ่ง ที่มีเส้นราสเตอร์คู่จำนวน 100 เส้น และเส้นราสเตอร์คี่จำนวน 100 เส้น และจะใช้หน่วยความจำ 80 K byte ในการควบคุมเส้น Raster แต่ละเส้นโดยที่ Bit แรกสุดจะควบคุม Pixel (0,0) และต่อมาก็ควบคุม (0,1), (0,2) และต่อไปเรื่อยจนถึง Pixel สุดท้ายของเส้น 0 (0,639) Pixel ต่อมาจะถูกตามด้วย (2,0), (2,1) และต่อไปเรื่อยจนกระทั่งถึง Block เส้นคู่เส้นสุดท้ายและ

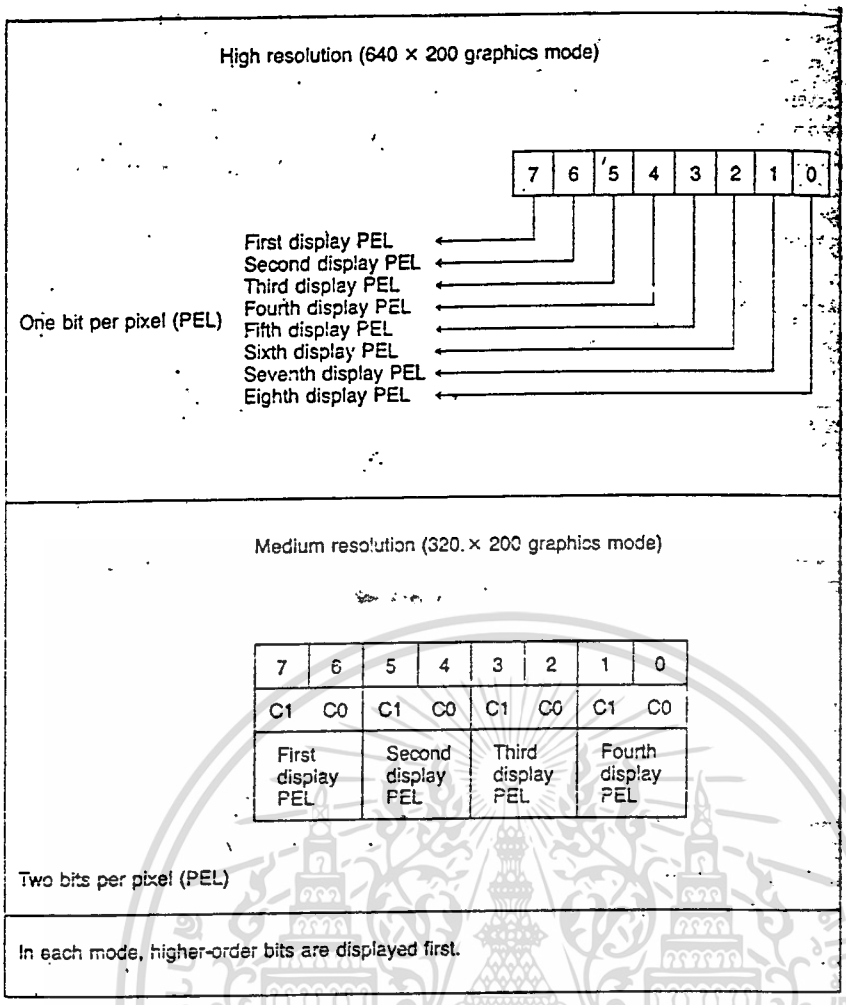


FIGURE 2.5 Bit assignments in graphics modes

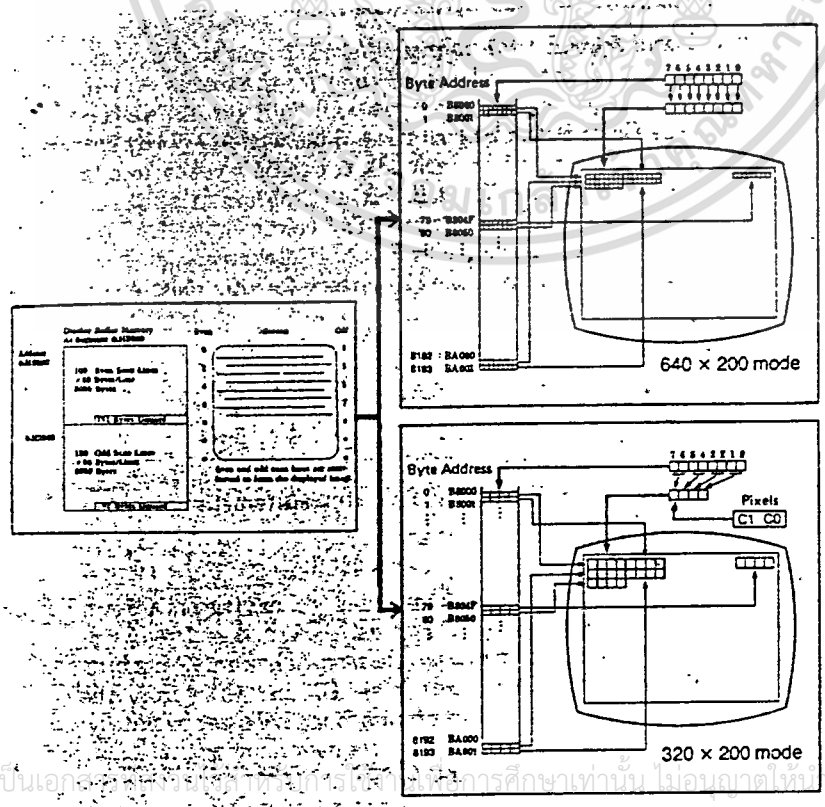


FIGURE 2.6 Memory mapping in graphics modes. (From M. Waite and C. L. Morgan, *Graphics Primer for the IBM PC*, Osborne/McGraw-Hill, 1983.)

ต่อไปก็ขึ้นขอบเขต Kilo byte ต่อไปก็เริ่มต้นด้วย Block เส้นคี่

โดยการใช้คำสั่ง Poke จะทำให้มีการเข้าใจ Interlacing ดียิ่งขึ้น สำหรับตัวอย่างโปรแกรมภาษา Basic นี้จะทำการ polt จุด 2 จุด โดยที่จุดเหนืออยู่บนอีกจุดหนึ่งและโปรแกรมเป็นดังนี้

```
10 Screen 2
20 DEF SEG = &HB 800
30 Poke 500, 3
40 Poke 500 + &H2000, 3
```

ค่าระยะขจัดที่แตกต่าง 8 K byte +(&H2000) ก็คือค่า offset ระหว่างส่วนที่เป็นเส้นคู่และส่วนที่เป็นเส้นคี่ของ Screen Memory

2.2.3 Accessing Memory from Basic

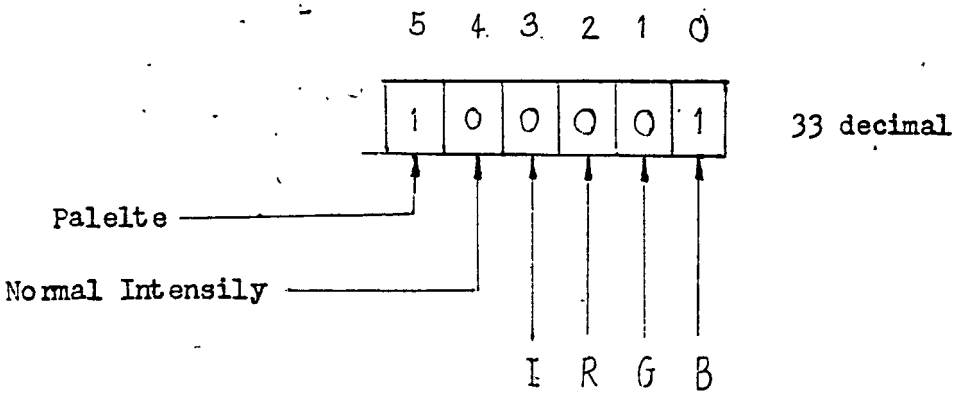
โดยการใช้คำสั่ง Peek และ Poke สำคับแรกเราต้องโปรแกรม Color Select Register เพื่อกำหนดรูปแบบของ Blackground color ของ Pixel ที่เราจะ Turn off และใส่ค่าตัวเลข Palelte ของ Pixel ที่เราจะ Turn on ตามที่ได้เห็นมาแล้วในส่วนที่ 1.3.5 Color Select Register คือ Register ที่เขียนได้เท่านั้นมีขนาด 6 Bit พร้อมกับต้องอ้าง Address I/O ที่ตำแหน่ง &H3D9 (หรือ 985 ในตัวเลขฐาน 10) โดย Register ตัวนี้จะถูกเขียนโดยใช้ Bios Roctine หรือภาษา Basic ก็ได้ ในที่นี้เราจะใช้ภาษา Basic ในการ Program Register ตัวนี้โดยที่เราสามารถเลือกสีของ Blackground สีใดสีหนึ่งในสี 16 สี ที่แสดงไว้ในตาราง 1.8 และต้องทำการเลือกค่า Palelte 1 ตัวจากที่มีอยู่ 2 ตัวตามที่แสดงไว้ในตาราง 2.3 สำหรับตัวอย่างจะเลือก Blackground สีน้ำเงิน และกำหนดค่า Pallet =1 รูปแบบ Bit ต่างในนี้จะมีลักษณะตามนี้

TABLE 2.3 Medium-resolution (320 × 200) color sets

SR 4 t)	Pixel data C1 (Red)	C0 (Green)	CSR bit 5 (Blue)	IRGB color number	Color name	BASIC Palette number	BASIC Color code
	0	0	(0)		Background	Palette 0	0
	0	1	0	2	Green		1
	1	0	0	4	Red		2
	1	1	0	6	Brown		3
	0	0	(1)		Background	Palette 1	0
	0	1	1	3	Cyan		1
	1	0	1	5	Magenta		2
	1	1	1	7	White		3
	0	0	(0)		Background	Palette 2	
	0	1	0	10	Light green		
	1	0	0	12	Light red		
	1	1	0	14	Yellow		
	0	0	(1)		Background	Palette 3	
	0	1	1	11	Light cyan		
	1	0	1	13	Light magenta		
	1	1	1	15	Intense white		

e: Palettes 2 and 3 cannot be accessed from the COLOR graphics statement. We must use the BIOS routine (or OUT statement) ccess these additional palettes. CSR stands for Color Select Register.

rce: T. V. Hoffmann, "The IBM Color/Graphics Adapter," Reprinted from *PC Technical Journal* 1 (1), 1983. Copyright © 1983, Davis Publishing Company.



ในการ Set ค่าเหล่านี้โดยใช้ภาษา Basic เราจะเขียนโปรแกรมได้ดังนี้

```
10 Screen 1, 0
20 OUT &H3D9, 33
```

เมื่อเราโปรแกรมค่าเหล่านี้ลงไปแล้ว Register จะทำการ set ค่า Blackground เป็นสีน้ำเงินและเลือก Palelte ในการควบคุมสีของ Pixel

ขณะนี้ colors ใน Medium resolution สามารถทำให้อยู่ในสีใดสีหนึ่งโดยการรวม Bit จาก Color Select Register กับอีก 2 Pixel bit คือ C_0 และ C_1 ที่ได้จาก display buffer ทำให้ได้รูปแบบ display buffer 4 Pixel ต่อ Byte ดังแสดงในรูป 2.5 โดยใช้ 2 บิทร่วมกันเราจะได้ค่าต่าง 4 แบบ ถ้าให้ค่าในตัวเลขฐาน 10 จะอยู่ในช่วงจาก 0 ถึง 3

เมื่อ C_1 และ C_0 เป็น 0 ทั้งคู่ Pixel จะไม่ทำงานและทำให้ได้สีของ Blackground เหมือนกับจอภาพ ต่อมา Adapter จะสแกนโดยตลอดทั้งหมดค่าสภาวะ C_0, C_1 ที่อยู่ภายใน Color Select Register เพื่อกำหนดสีของ Blackground

เมื่อ C_1 และ C_0 เป็นค่าตัวเลขที่นอกเหนือไปจากค่า 0 Pixel ก็อยู่ในสภาวะ turn on ต่อมา Adapter ก็จะ Scan โดยตลอด ค่าสภาวะ C_0, C_1 ที่อยู่ภายใน Color Select Register เพื่อกำหนดสภาวะของสี fore-ground จากค่า Palelte ที่กำหนดไว้สำหรับ Palette ที่จะให้ค่านี้จะต้อง

ให้ค่าตัวเลขคล้องจองเข้ากับโค๊ดสีใน Palelte ที่มีอยู่ โปรแกรมต่อไปนี้จะ Turn on Pixel 3 จุดที่มีสีที่แตกต่างบนจอภาพ

```
10 CLS : Screen
20 OUT &H3D9, 33 ; Set Background Color
30 DEF SEG = &HB80; And Select Palelte 1
40 Poke 0,0
50 Poke 80, 64
60 Poke 160, 128
70 Poke 240, 192
```

ใน Mode 640 x 200 display buffer เก็บค่ารายละเอียดที่เกี่ยวกับสภาวะของจุดแต่ละจุดคือ on หรือoff ตามที่กล่าวไว้แล้ว 1 Bit ที่อยู่ภายใน Memory Mapping จะควบคุมได้ 1 Pixelใน High Resolution Mode จึงไม่มีเนื้อที่ที่จะกำหนดสีลงใน Pixel เพราะฉะนั้นในล่าดับแรก adapter จะสแกนโดยตลอดสภาวะของแต่ละจุดบนจอภาพ ต่อมาก็ผ่านค่ารายละเอียดนี้ไปให้ Color Select Register เพื่อกำหนดสีของ foreground ของ Pixel ในตำแหน่งที่ on ในการโปรแกรมให้มีสีของตัวอักษรเป็นสีแดงไปยัง Register ใน Mode High Resolution graphic รูปแบบของ Bit Pattern ใน Color Select Register เป็นดังนี้

5	4	3	2	1	0	
0	0	1	1	0	0	12
		I	R	G	B	

ในการ Set ค่านี้ด้วยภาษา Basic จะเขียนโปรแกรมได้ดังนี้

```
10 Screen 2
2 OUT &H3D9, 12
```

Using the Color Statement

ใน graphic mode คำสั่ง color ในภาษา Basic จะมีรูปแบบของประโยคแตกต่างไปเล็กน้อย รูปแบบของประโยคคำสั่ง color ใน Mode

Medium Resolution จะมีลักษณะดังนี้

Color [background] [, Palelte]

เมื่อ

Back ground = ตัวเลขที่จะกำหนดสีของ Background โดยโค๊ดสีของ Background จะอยู่ในช่วง 0 ถึง 15 ดังที่แสดงในตาราง 1.8

Palelte = ตัวเลขที่แสดงอยู่ในช่วง 0 ถึง 1 เพื่อเลือก Palelte ของกลุ่มสีดังแสดงในตาราง 2.3

การเลือกสีของ Background ทำได้เหมือนกับการเลือกสี Palelte หนึ่ง ๆ การให้ค่า Palelte จะเป็นค่าที่เราใช้เลือกระหว่างสีกลุ่มใดกลุ่มหนึ่งใน 2 กลุ่มสีที่ทำงานด้วยคำสั่งทาง graphic เช่น Pset, PReset, line, Circle, Paint, Draw(คำสั่งเหล่านี้จะนำมากล่าวในตอนกลังต่อไป) ภายหลังที่มีการกำหนดค่าตัวเลข Palelte แล้วโดยคำสั่ง color เราก็สามารถที่จะเลือกสีหนึ่งสีใดใน 3 สีที่กำหนดไว้ใน Mode High นั้น ส่วนใน Mode High Resolution graphic จะไม่มีคำสั่ง color

2.3 IBM PC graphic basic command

ขณะนี้เราได้รู้วิธีการเข้าหาและควบคุม Screen Memory โดยการใช้คำสั่ง Poke และ Peek แล้ว โดยแน่นอนคำสั่ง Basic ทั้งสองนี้จะมีประโยชน์ในการเข้าหาข้อมูลภายในScreen Memory Map แต่คำสั่งทั้งสองนี้ไม่เหมาะสมในการสร้างภาพ graphic ที่มีความยุ่งยาก คำสั่งทาง graphic หลายคำสั่ง ที่มีอยู่บนเครื่อง IBM PC สามารถนำมาใช้ในการกำหนดภาพ graphic ที่ยุ่งยากได้โดยวิธีการง่าย เหมือนกับว่าคำสั่งเหล่านี้คือ กลุ่มคำสั่งย่อยของภาษา Basic ที่เขียนขึ้นด้วยโค๊ดระดับภาษาเครื่องและมีความเร็วอย่างเพียงพอที่จะนำมาใช้กับงานที่เกี่ยวกับ graphic ในส่วนนี้เราจะแนะนำโดยย่อเกี่ยวกับคำสั่งเหล่านี้พร้อมกับตัวอย่างที่อธิบายว่าคำสั่งเหล่านี้คืออะไร และเราจะใช้มันในการกำหนดภาพ graphic computer ได้อย่างไรในการเริ่มต้นเราจะกล่าวถึงวิธีการกำหนดโคออดิเนตใน Basic graphic

Graphic Coordinate

คำสั่งเกี่ยวกับ graphic ต้องการข้อมูลเกี่ยวกับตำแหน่งที่จะวาดบนจอภาพเราจะใช้ข้อมูลนี้ในรูปแบบของ Coordinate Coordinate จะอยู่ในรูปแบบ (x,y) เมื่อ x คือตำแหน่งทางแนวนอนและ y คือตำแหน่งทางแนวตั้ง โดยจะเรียกลักษณะดังกล่าวนี้ว่า Absolute form ซึ่งรูปแบบดังกล่าวนี้จะอ้างถึงตำแหน่งที่ปรากฏจริงของจุดบนจอภาพ โดยจะไม่พิจารณาไปยังตำแหน่งสุดท้าย ในการแสดงคำสั่งเกี่ยวกับ graphic ค่าของ (x,y) จะแทน Coordinate ของจุดเพียงจุดเดียว

ยังมีวิธีอื่นในการแสดง Coordinate อีกแบบคือ relative form ถ้าต้องการใช้รูปแบบดังกล่าวนี้ เราจะต้องบอกภาษาเบสิคว่าจุดไหนคือจุดสุดท้ายที่ถูกอ้างอิงถึง รูปแบบจะเป็นดังนี้

Step (x offset, y Offset)

เมื่อ x และ y คือค่า offset ที่แทนระยะทางในทางแนวนอนและแนวตั้งจากจุดสุดท้ายที่ถูกอ้างอิงถึง (โดยที่จุดสุดท้ายที่ถูกอ้างอิงถึงจะ set ค่าด้วยคำสั่งทาง graphic) เราจะให้เห็นตัวอย่างของวิธีการใช้ Coordinate ทั้งสอง โดยจะกล่าวให้เห็นในส่วนหลังต่อไป)

Color Convention

ใน Mode Medium Resolution Colorก็คือค่าตัวเลขหนึ่ง ๆ ที่อยู่ในระหว่าง 0 ถึง 3 Color 0 จะเลือกสี blackground ในขณะที่ color 1,2,3 แต่ละตัวเลขจะเลือกสีหนึ่งสีใดจากค่า Palelte ที่เลือกไว้ในคำสั่ง color แต่ถ้าไม่มีการกำหนดคำสั่งใน color แล้ว สี foreground จะถูกกำหนดให้มีค่าเท่ากับ 3 ของโค๊ดสีใน Palelte นั้น ใน Mode High Resolution ค่าตัวเลือก 0 จะเลือกสีที่เป็นสีดำและค่า 1 สีที่เลือกก็คือสีขาว ถ้าไม่มีการกำหนดโค๊ดสี สี foreground จะถูกสมมุติขึ้นและสีอื่นทั้งหมดจะเกี่ยวข้องกับ graphic ในคำสั่งทาง graphic จะตามด้วยรูปแบบ color เหล่านี้

2.3.2 Pset and Preset Statement

Pset คือคำสั่งทาง graphic แบบง่าย ๆ โดยที่คำสั่งนี้จะทำการปรับค่า Pixel หนึ่ง ๆ ให้มีสีตามที่เลือกไว้ รูปแบบของคำสั่งคือ

Pset (x,y) , [color]

Preset (x,y) ,[color]

ค่า color ที่อยู่ในสี่เหลี่ยมปีกกาจะแสดงถึงว่าสีที่ใช้ในวงเล็บอาจจะมีในคำสั่งหรือไม่มีก็ได้ และถ้าไม่มีการกำหนดค่าภายในสี่เหลี่ยมปีกกานี้จะทำการรักษาค่าแต่ก่อนที่กำหนดไว้ก่อนหน้านี้ ในคำสั่งนี้ color จะมีหรือไม่มีก็ได้

Preset จะมีลักษณะคล้ายกับ Pset ต่างกันก็เพียงว่าถ้าไม่มีการให้ค่า Parameter ของ color ไว้ในคำสั่ง Preset ค่าสี Back - ground จะมีค่าเป็น 0 คำสั่ง Preset (x,y) จะวาดจุดบนจอภาพในรูปแบบสี Blackground การกระทำแบบนี้จะทำให้เห็นว่าตำแหน่งดังกล่าวถูกลบทิ้งโดยผู้ใช้ เพราะฉะนั้นคำสั่ง Pset (x,y) 0 ก็จะมีลักษณะเหมือนกับคำสั่ง Preset (x,y)

ตัวอย่างโปรแกรมต่อมาจะแสดงให้เห็นว่าคำสั่งเหล่านี้สามารถนำมาใช้ในการ Plot จุดบนจอภาพได้อย่างไร

```
10 Screen 1,0 ; Set Medium Resolution in color
20 Color 1, 0 ; Set background blue, Palelte 0
30 Pset (3, 10), 2 ; PUT Dot in red
40 Pset step (7,-5) ; put another dot in red at (10,5)
50 Preset (3,10) ; Erase dot at (3,10)
```

หมายเหตุ Pset step ในบรรทัด 40 กำหนด coordinate ในรูปแบบของ Relative coordinate

funtion ทาง graphic เกือบจะทั้งหมดจะสามารถกระทำได้โดยใช้คำสั่ง Pset และ Preset เช่น lines,Circle หรือรูปแบบอื่นที่ประกอบขึ้นด้วยจุดที่ต่อกันเพื่อจะสร้างรูปแบบตามที่ต้องการ วิธีการที่เร็วและง่ายกว่าที่จะทำการวาดรูปสี่เหลี่ยมนั้นด้วยคำสั่งเพียงคำสั่งเดียวแทนที่จะวาดและกำหนด

ตำแหน่งของแต่ละ Pixel ที่ละ Pixel คำสั่ง line และ circle ก็จะเป็นคำสั่งที่มีความสะดวกกว่า

2.2.3 line statement

คำสั่ง line สามารถที่จะทำการต่อจุดระหว่างจุดทั้งสองให้เป็นเส้นตรง (เส้นทึบและเส้นประ) หรือวาดรูปสี่เหลี่ยมบนจอภาพ คำสั่ง line สามารถวาดรูปสี่เหลี่ยมได้รูปแบบของคำสั่ง line จะเป็นดังนี้

Line (x,y) - (x_2,y_2) ,(color) 9,B(F)) (,style)

ค่าที่อยู่ในวงเล็บปีกกาจะเป็นตัวแสดงให้ทราบว่าค่าที่อยู่ในวงเล็บนี้จะมีหรือไม่มีก็ได้และคู่ coordinate (x_1 ,y_1) และ (x_2,y_2) คือ coordinate ของจุด 2 จุดที่จะทำการต่อกันและการใช้ color หรือไม่ใช้ color ก็จะมีมีลักษณะเหมือนกับคำสั่ง Pset และตัวเลือก B วาดรูปสี่เหลี่ยมที่ไม่มีสี ในขณะที่ BF จะวาดสี่เหลี่ยมพร้อมสีที่เลือกไว้ตัวเลือกสุดท้ายคือ style คือตัวเลขขนาด 16 Bit ที่ใช้ในการวาดเส้นประตามช่วงตลอดจอภาพ คำสั่งนี้จะมีอยู่ในภาษา Basic Version 2 . หรือ Version ที่สูงกว่านี้ และตัวเลือก style ถูกเลือกใช้เป็นเส้นปกติและรูปสี่เหลี่ยมแต่ไม่สามารถถูกนำมาใช้ Lilled Boxes

รูปแบบของคำสั่ง line ที่เป็นตามนี้คือ

Line - (x_2,y_2)

ถ้าไม่มีจุดเริ่มต้น (x_1,y_1) , PC จะทำการวาดเส้นตรงจากจุดสุดท้ายที่กำหนดไว้ในคำสั่ง line ก่อนหน้านี้มายังจุด (x_2,y_2) ในรูปแบบสี foreground สมมุติว่าเราต้องการที่จะวาดเส้นที่ต่อเนื่องกันไป การเริ่มต้นของแต่ละเส้นจะอยู่ที่ตำแหน่งสุดท้ายของเส้นตรงเส้นที่แล้ว ในการทำดังกล่าวนี้สามารถใช้ลำดับขั้นตอนของคำสั่ง line ดังนี้

```
10 screen 1, 0                  ; Set medium Resolution in color
20 color 0,0                    ; Black Background, Palelte 0
30 Line (30,100) - (100,120); Draw atriangle
```

40 Line - (130, 100)

50 Line - (130, 100)

(ดูในรูป 2.7) ตัวอย่างต่อมาจะวาดรูปสี่เหลี่ยมให้มีสีตามที่เลือกไว้ในกรณีนี้กลุ่มทั้ง 2 ของ coordinate (x_1,y_1) และ (x_2,y_2) จะแทนด้วยจุดทะแยงที่อยู่ตรงข้ามกันของรูปสี่เหลี่ยมนั้น (ดูรูป 2.7)

```
10 screen 1, 0
20 Color 1, 1
30 Line (0,0) - (100,100), 2, BF
```

ส่วนลักษณะของเส้นที่แสดงผลโดยการกำหนด Bit ให้แก่ตัวเลือก style เพราะว่า style มีขนาดกว้าง 16 Bit และรูปแบบของ Bit ที่จะทำให้ได้ dot lineเป็นดังนี้

10 10 10 10 10 10 10 10

ค่า Binary รูปแบบนี้จะมีค่า = AAAA.. ฐานสิบหกเพราะว่า hex A= 1010 เพราะฉะนั้นเราสามารถสร้างรูปแบบของเส้นได้โดยการกำหนดรูปแบบของ 16 Bit นี้ ลำดับเส้นแต่ละเส้นถึง Lalter dashes จะถูกกำหนดในลักษณะดังนี้

1100 1100 1100 1100 (&HCCCC)

1110 1110 1110 1110 (&HEEEE)

1111 1000 1111 1000 (&HF8F8)

1111 1111 0000 0000 (&HFF00)

ในการวาดรูปสี่เหลี่ยมที่มีสี cyan ด้วยเส้น dashed เราจะเขียนโปรแกรมได้ตามนี้

```
10 screen 1, 6
20 Color 0, 1
30 Line (100,100) - (200,140) , 1, B, &HCCCC
```

ดูรูป (2.7 C) จุดสุดท้ายที่อ้างถึง ภายหลังคำสั่ง line เราสามารถนำมาใช้เป็นจุดในการวาดเส้นตรงอีกเส้นหนึ่งได้ดังตัวอย่าง

Line (100,100) - step (50,-30)

คำสั่งนี้จะวาดเส้นตรงจากจุด (100, 100) ไปยังจุด (150, 70) หมายเหตุ ในภาษา Basic release 2.0, จุดที่อยู่นอกเหนือไปจากย่านของ coordinate นั้นจะมองไม่เห็นบนพื้นผิวที่มองเห็น ลักษณะดังกล่าวนี้จะถูกเรียกว่า line clipping เพราะว่าภาพนั้นอยู่ภายนอกของย่าน coordinate จะถูก clip ที่ขอบเขตของพื้นผิวที่มองเห็น ลักษณะแบบนี้จะมีประโยชน์มากใน computer graphic ดังที่จะแสดงให้เห็นในบทที่ 3

2.3.4 Circle Statement

คำสั่ง Circle จะเป็นคำสั่งที่ใช้วาด O, เส้นโค้ง, และรูปวงรี และยังเป็นคำสั่งที่สามารถสร้างรูปแบบ curve ต่าง ๆ ได้ง่าย ๆ รูปแบบของข้อความคำสั่ง circle คือ

Circle (x,y), r ,[color [,start, end[,aspect]]]

โคออดิเนต (x,y) คือ จุดศูนย์กลางของ O และ R คือรัศมี (ระยะทางจากจุดศูนย์กลางไปยังขอบเขตด้านนอกของ O) color ก็คือสีของ O และจะมีหรือไม่มีในข้อความนี้ก็ได้ (ถ้าไม่ได้กำหนดก็ถือว่าให้มี color = 3) start และ end คือจุดปลายทั้งสองของส่วนโค้งที่ต้องการวาดและถูกกำหนดขึ้นในรูปแบบ radian (ดูในรูป 2.8) ถ้า Parameter ทั้ง 2 ไม่ได้ใช้คำสั่งนี้จะทำการวาด O ทั้งหมด และ Parameter ตัวสุดท้าย คือ Aspect คืออัตราส่วนระหว่างรัศมีตามแนวแกน y และรัศมีตามแนวแกน x (ความสูง/ความกว้าง) ซึ่งจะยอมให้เราได้ใช้ส่วนนี้ในการวาดรูปวงรี ในวงกลมจริงนี้ ค่า Aspect ratio นี้จะมีค่า 1 เสมอ สำหรับจอภาพ เช่น IBM PC ซึ่งความยาวของ Pixel มีขนาดไม่เท่ากันในทิศทางด้าน x และ y เช่น (320 x 200) เราต้องมีการปรับ Aspect Ratio เพื่อให้ได้วงกลมจริง ๆ ใน Mode Medium Resolution ค่า Aspect Ratio นี้จะมีค่า = 5/6 และถ้าค่าตามแกน

น้อยลงก็จะมีผลต่อการมองภาพ o นั้น ๆ ส่วนค่า Aspect Ratio ใน Mode High Resolution จะมีขนาดเป็น $\frac{1}{2}$ ของ Mode Medium Resolution ดังนั้นในการสร้างรูป o รูปเดียวกันนี้เราจะ set ค่า Aspect Ratio = $\frac{5}{12}$ (ดูรูป 2.9) ถ้าค่า Aspect Ratio 1 ค่า R ก็คือรัศมีที่วาดตามแนวแกน y แต่ถ้า Aspect Ratio 1 ค่า R ก็คือค่ารัศมีที่วัดตามแนวแกน y

คำสั่ง circle สามารถวาด wedge ได้อีกด้วย (หรือ Pie), wedge นี้ก็ไม่แตกต่างไปจาก ARC มากนักเพียงแต่ว่า wedge นี้จะมีเส้นตรงต่อระหว่างจุดปลายทั้งสองเข้ากับจุดศูนย์กลางของ o ด้วยคำสั่ง circle ถ้าจุดเริ่มต้นหรือจุดปลายจุดใดจุดหนึ่งเป็นค่าจำนวนลบจุดที่เป็นลบจะถูกต่อเข้ากับจุดศูนย์กลางของ o ถ้าจุดทั้ง 2 เป็นลบทั้งคู่ก็สามารถสร้างรูป wedge ได้ โปรแกรมต่อไปนี้จะแสดงให้เห็นการใช้คำสั่ง circle

```
10 CLS
20 screen 1, 0
30 color 0,1
40 Pi = 3.141593
50 circle (100,100), 30, draw a circle in White
60 circle (100,100), 30, 5/18 ; draw a Horizontal Ellipse
70 circle (100,100),30,2 ,1.2*Pi,1.7*Pi ; draw an ARC
80 circle (100,100),30,2,-1,3*Pi, -1,7*Pi; draw a wedge
```

หมายเหตุ ในบรรทัดที่ 60 แสดงให้เห็นว่า Parameter บางตัวจะถูกข้ามไปโดยอยู่ในระหว่างเครื่องหมายลูกน้ำ , , , ข้าม Parameter ไป 2 ตัว

จุดสุดท้ายที่อ้างอิงถึงภายหลัง o ที่วาดขึ้นคือ จุดศูนย์กลางของ o และถ้าจุดนี้อยู่นอกจอภาพจะไปวาดจุดทั้งหมดที่อยู่บน o ลงบนจอภาพ (ดูรูป 2.10)

2.3.5 Paint Statement

คำสั่ง Paint จะเติมสีลงบนจอภาพด้วยสีที่เลือกไว้ รูปแบบคำสั่งที่สมบูรณ์จะเป็นดังนี้

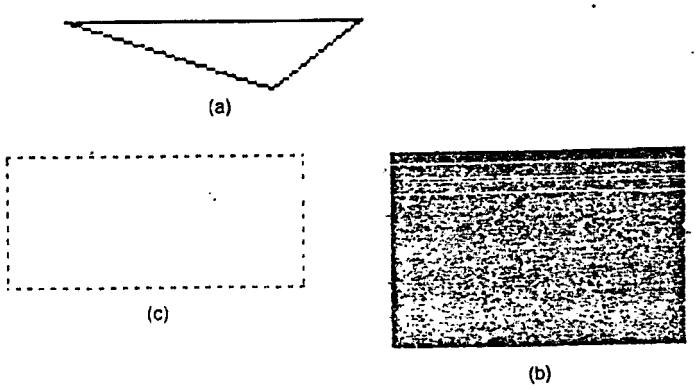


FIGURE 2.7 Line statements used to draw (a) a triangle; (b) a box filled in magenta, (c) a cyan box with dashes.

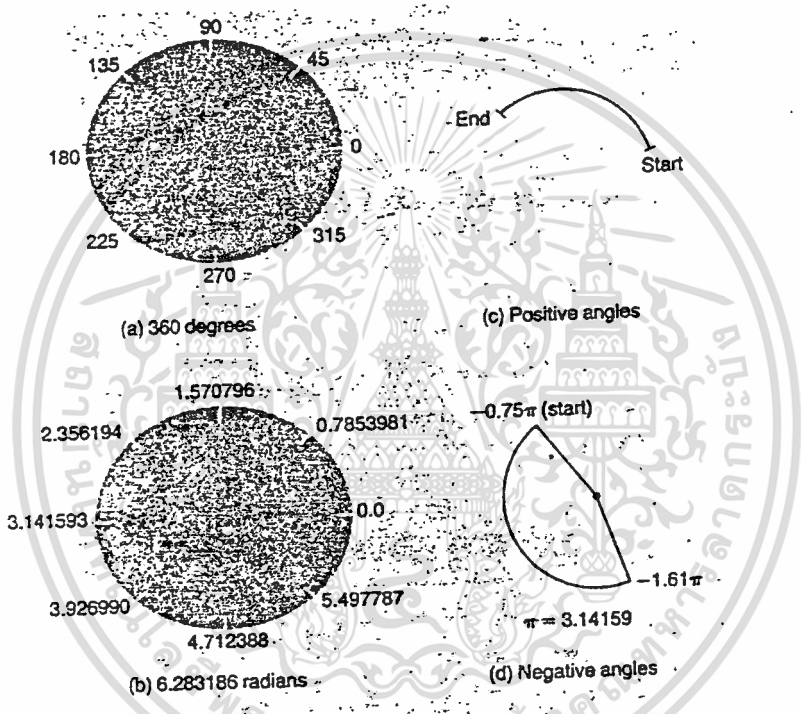


FIGURE 2.8 Specifying circle parameters

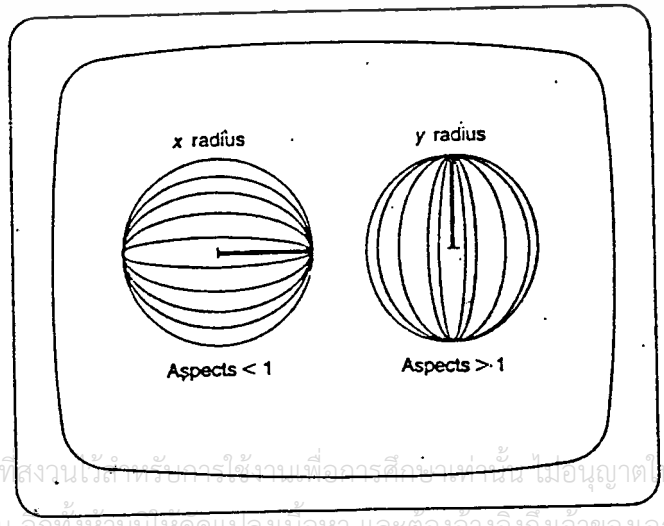


FIGURE 2.9 Effect of aspect ratio value on ellipses

Paint (x,y) [, Paint] [, boundary]

Coordinate (x,y) คือจุดที่อยู่ภายในเนื้อที่ที่ต้องเติมสี Paint สามารถเป็นได้ทั้งตัวเลขและ string ถ้าเป็นตัวหมายเลขมันจะมีความหมายถึงสีที่ต้องใส่ในเนื้อที่บนจอภาพ แต่ถ้าเป็น string จะเป็นการกำหนดรูปแบบของ Parameter และถูกนำไปใส่ใน Painted figure figure ควรที่จะวาดภายในขอบเขตสี (figure ที่ถูกเติมสีก็คือ figure ของขอบเขตสี)

Paint tiling

วิธีการออกแบบลักษณะที่แทน string โดยการใช้ tile Mask tile mask โดยปกติจะมีขนาดกว้าง 8Bit และต้องเขียนในรูปแบบ string ดังนี้

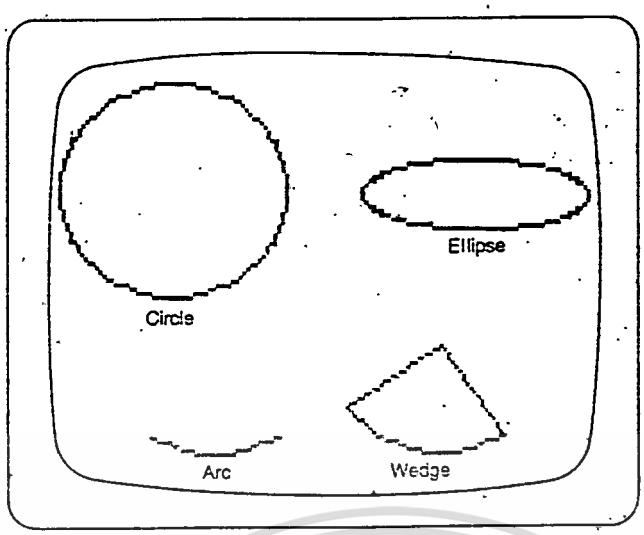
CHR (Hexadecimal numbers Represent Hz 8 Bit Pattern)

การแสดงด้วย stringนี้อาจประกอบกันได้มากถึง 64 ไบท์ โครงสร้างของตัวแทน string จะปรากฏในรูป 2.11 โดยรูปแบบของtile แล้วจะถูกกระทำซ้ำโดยตลอดจอภาพ เนื่องจากใน Mode High tesolution จะมีเพียง 1 Bit / pixel (ดูในส่วน 2.2.1) และจุด ๆ หนึ่งจะถูก plot ที่ทุก ๆ ตำแหน่งใน Bit Mask ที่มีค่าเป็น 1 สำหรับตัวอย่างถ้าเราออกแบบ tile Mask ด้วยหน่วยความจำ 3 ไบท์ที่มีรูปแบบ Bit Pattern ดังนี้

	7	6	5	4	3	2	1	1	ตำแหน่งบิท
Tile byte 0	1	1	0	0	0	1	0	1	CHR $ (&HC5),197
Tile byte 1	0	1	0	1	1	0	1	0	CHR $ (&H5A),90
Tile byte 2	0	1	1	1	1	1	1	0	CHR $ (&H7E),126

เราก็สามารถเปลี่ยนค่า Bit Paltern เหล่านี้อยู่ในรูปแบบของ string ได้ดังนี้

Tile $ = CHR $ (&HC5)+CHR $(&H5A)+CHR$(&H7E)



and wedge)

	7	6	5	4	3	2	1	0	Bit position
BYTE 0	x	x	x	x	x	x	x	x	
BYTE 1	x	x	x	x	x	x	x	x	
BYTE 2	x	x	x	x	x	x	x	x	
BYTE 3	x	x	x	x	x	x	x	x	
.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	
BYTE 62	x	x	x	x	x	x	x	x	
BYTE 63	x	x	x	x	x	x	x	x	

x = 0 or 1 binary bit.

FIGURE 2.11 The structure of tile pattern in PAINT (640 × 200 mode)

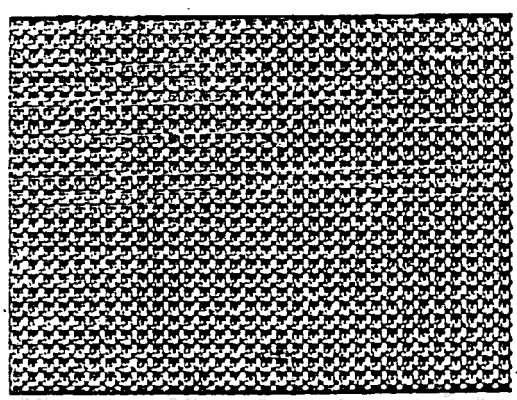


FIGURE 2.12 Effect of tiling in 640 × 200 graphics mode

โปรแกรมข้างล่างนี้จะแสดงให้เห็นวิธีการ tile ของรูป ด้วย tile mask ที่ออกแบบตามข้างบนนี้

```
10 CLS: screen 2 : Key OFF
20 Line (100,100) -(400,180), B
30 Tile $ : CHR$(&HC5) + CHR$(&H5A) +CHR$(&H7E)
40 Paint (300,140), Tile $
50 END
```

ในกรณีนี้จะใส่สีที่ coordinate (300, 140) และจนเริ่มต้นการ Plot Byte ที่ 1 รูปแบบดังกล่าวนี้จะปรากฏบนจอภาพดังรูป 2.12

วิธีออกแบบ Paltern ใน Mode Medium Resolution นั้นแตกต่างไปจากวิธีที่ใช้ใน Mode High Resolution โดยที่หนึ่ง Medium resolution Bit จะอธิบายได้ถึง 4 Pixel เพราะว่าใน Mode Medium resolution จะใช้ Bit 2 Bit ต่อ Pixel ลักษณะเท่านั้น (ดูในส่วน 2.2.1) เพราะฉะนั้นทุก ๆ 2 Bit ของ tile byte จะเป็นสีใดสีหนึ่งใน 4 สีสัมพันธ์กับจุดใดจุดหนึ่งใน 4 pixel ที่ถูก plot ตาราง 2.4 แสดงให้เห็นค่า Binary และ Hexadecimal ที่สัมพันธ์กับสีที่ได้

โปรแกรมต่อไปนี้จะแสดงให้เห็นวิธีการ tile พื้นที่ o วงหนึ่งใน Mode Medium Resolution ด้วยเส้นตรงเส้นที่เป็นสีแดง และเส้นตรงอีกเส้นที่เป็นสีเขียว (ดูรูป 2.13)

```
10 CLS : screen 1, 9 : keyoff
20 color
30 Tile$ =CHR$(&HAA) + CHR$(&HAA) + CHR$(&H55)
40 circle (160, 100), 50, 3
50 Paint (160,100,Tile,3)
60 END
```

TABLE 2.4 The tile patterns in 320 × 200 graphics mode

Color palette 0	Color palette 1	Color number in binary	Pattern to draw solid line in binary	Pattern to draw solid line in hexadecimal
Green	Cyan	01	01010101	&H55
Red	Magenta	10	10101010	&HAA
Brown	White	11	11111111	&HFF

Source: *IBM PC Basic Reference Manual*

FIGURE 2.13 Painting circle with tile pattern (320 × 200 mode)

TABLE 2.5 DRAW statement—graphics definition language

Control command	Format	Effect	Parameter
Move and draw	M *x*,*y*	To specified point (*x*,*y*)	*x*,*y*, = absolute coordinates
	M +*x*,+*y*	To specified point (*X* + *x*, *Y* + *y*)	*X*,*Y* = last referenced point +*x*,+*y* = relative coordinates
	U*n*	Up [↑]	*n* rows'
	D*n*	Down [↓]	*n* rows'
	R*n*	Right [→]	*n* columns'
	L*n*	Left [←]	*n* columns'
	E*n*	Diagonally up and right [↗]	*n* = diagonal distance
	F*n*	Diagonally down and right [↘]	*n* = diagonal distance
	G*n*	Diagonally down and left [↙]	*n* = diagonal distance
	H*n*	Diagonally up and left [↖]	*n* = diagonal distance
Prefix	B [Move and Draw]	Do not draw on the next move	
	N [Move and Draw]	Return to current point after next move	
Action	A *n*	Rotate all subsequent moving and drawing counterclockwise	*n* = rotation angle 0 = 0 2 = 180 1 = 90 3 = 270
	TA *n*	Turn angle *n*	*n* = rotation angle −360 ≤ *n* ≤ 360 *n* > 0 counterclockwise *n* < 0 clockwise
	S *n*	Scale subsequent distances	*n* = scale factor 1 ≤ *n* ≤ 255
	C *n*	Select color from given palette	*n* = 0 or 1 (640 × 200 mode) = 0, 1, 2, or 3 (320 × 200 mode)
	P paint, bound	Paint or fill in area	paint = paint color bound = boundary color
	X variable	Execute subcommands from another string	variable = a string variable that contains more subcommands

2.3.6 DRAW Statement

คำสั่ง draw จะทำการวาดวัตถุตามที่กำหนดด้วย string รูปแบบของข้อความเป็นดังนี้

draw string

คำสั่ง draw จะใช้ภาษา graphic definition languge (GDL) ในการทดสอบที่รวมกันของเส้นตรงและจุด คำสั่งภาษานี้จะประกอบด้วยตัวแทน string ซึ่ง string นี้จะกำหนดวัสดุหนึ่งที่ถูกวาดขึ้นเมื่อภาษา Basic นี้execute คำสั่ง draw ในระหว่างการ execution ภาษา Basic จะทำการตรวจสอบค่า string และแปลงความหมายของคำสั่งของตัวอักษรตัวเดียวจาก string รายละเอียดคำสั่งเหล่านี้จะอยู่ในตาราง 2.5

Move and draw command

คำสั่ง Move (m) มีการใช้งานเป็น 2 mode ที่แตกต่างกันคือ Relative และ Absolute Mode ทั้งสองนี้จะมีลักษณะคล้องจองและการทำงานเหมือนกับ Relative และ Absolute ของคำสั่ง Pset,line และ circle Mode Relative นี้จะแสดงโดยตัวเลขจำนวนเต็มที่คิดเครื่องหมาย และ Mode Absolute จะแสดงโดยตัวเลขจำนวนเต็มที่ไม่คิดเครื่องหมาย

Command string ประกอบไปด้วยคำสั่ง move เดียว ๆ หรือคำสั่งหลาย ๆ คำสั่งรวมเข้าด้วยกันภายใน string เดียวกันโดยที่จะใส่ semincolon ระหว่างคำสั่ง move และใส่ space ในที่ใด ๆ โดยตลอดที่มีคำสั่ง string ดังนั้นผลของคำสั่ง draw มากมายจะถูกสะสมเพิ่มขึ้น และก็ไม่ควรที่จะรวมคำสั่ง move ทั้งหมดนี้ไห้อยู่ในรูปแบบของ Common string เพียงคำสั่งเดียว

มีคำสั่ง move แบบพื้นฐาน 8 แบบคือ U, D, L, R, E, F, G และ H ในตารางที่ 2.5 แสดงให้เห็นลักษณะทิศทางการเคลื่อนที่ของแต่ละคำสั่ง move และในแต่ละคำสั่งmove ก็สามารถเขียนเป็นรูปแบบคำสั่งได้ 3 แบบ ตัวอักษรของมันเอง, ตัวอักษรตามด้วยค่าจำนวนเต็มไม่คิดเครื่องหมาย, ตัวอักษร

ตามด้วยเครื่องหมาย = และชื่อตัวแปรตัวอักษรของมันเอง แสดงว่ามีการเคลื่อนที่ครั้งหนึ่งไปเป็นระยะทาง 1 unit ในทิศทางที่กำหนดไว้ ส่วนตัวอักษรตามด้วยค่าคงที่ไม่ติดเครื่องหมาย หรือตามด้วยเครื่องหมาย = ชื่อตัวแปรจะยอมให้เราเคลื่อนที่ได้หลาย ๆ หน่วยในทิศทางนั้น ๆ

Prefix Command

B และ N Prefixed ถูกนำมาใช้โดยคำสั่งทั่ว ๆ ไป (global)(m) หรือคำสั่ง move เฉพาะที่ (loval) B Prefix จะทำให้คำสั่งที่ตามมาถูก execute เป็นการ move โดยไม่มีการ draw ในขณะที่ N Prefix ทำให้ตำแหน่งปัจจุบันยังคงอยู่ในตำแหน่งเดิมก่อนที่คำสั่งอื่นจะถูก execute

Active Command

มีคำสั่งเกี่ยวกับการกระทำอยู่ 6 แบบที่ยอมให้เราปรับค่าสีและdraw figure รูปแบบของคำสั่งเหล่านี้ถูกแสดงในตาราง 2.5 คำสั่ง C แทนสี และมันจะทำการ Set ค่าสีสำหรับคำสั่ง move ทั้งหมดจนกระทั่งถึงคำสั่ง C ตัวต่อไป คำสั่ง A จะ set ค่ามุมของคำสั่ง move ทั้งหมดจนกระทั่งถึงคำสั่ง A ตัวถัดไปและมีการเคลื่อนที่ได้ 4 แบบ คือ 0, 90, 180, 270และก็คำสั่งต่อมาคือคำสั่ง TA จะเป็นคำสั่งที่กลับรูปแบบเป็นมุมโดยตลอด 360 องศา แต่เป็นการเลื่อนทีละ 1 และถ้าค่ามุมเป็นบวกก็จะหมุนภาพไปในทิศทางทวนเข็มนาฬิกาและค่ามุมเป็นลบที่จะหมุนไปในทิศทางตามเข็มนาฬิกา TA ตามด้วย TA ไม่ใช่เป็นการ Commulative แต่ละ TA จะทำการ Reset มุมเพื่อให้ภาพเปลี่ยนไปทีละลำดับ เมื่อวัตถุถูกหมุนโดยค่ามุมที่กำหนดไว้วัตถุเหล่านี้จะถูกวาดด้วยค่า Aspect Ratic 4/3 หมายความว่า จะทำการวาดวัตถุที่หมุนในทิศทางใดทิศทางหนึ่งโดยที่จะป้องกันไม่ให้ภาพนั้นผิดรูปแบบหรือเพี้ยนไปในกรณีที่นำภาพดังกล่าวนั้นมาใช้ใน Monitor ทั่ว ๆ ไป

คำสั่ง S จะทำการ Set ค่า Scale factor เป็นคำสั่งที่ใช้ในการปรับขนาดวัตถุ ค่า scale factor ต้องอยู่ในช่วงระหว่าง 0 ถึง 255 คำสั่ง P จะเป็นคำสั่งที่ใช้ set สีของรูปภาพที่จะ paint และสี border ไปจนถึงขอบเขตสิ้นสุด

และสุดท้ายคือคำสั่ง X ทำหน้าที่คล้ายกับ Subroutine call มันจะทำให้เกิดชื่อคำสั่ง string ที่ถูก execute ในข้อความคำสั่งตัวอักษรจะถูกตามด้วย Semicolon, Semicolon จำเป็นต้องใส่เพื่อให้ภาษา Basic รู้ว่าชื่อของ string ได้ถึงสิ้นสุดแล้ว เราจะแสดงให้รู้วิธีการใช้คำสั่ง draw ต่าง ๆ ด้วยโปรแกรมสั้น 2 โปรแกรมข้างล่างนี้ โปรแกรมแรกจะทำการวาดรูปสามเหลี่ยมสีเขียวและขอบเขตสิ้นสุดสีแดงบนพื้นสีน้ำเงินจุดเริ่มต้นอยู่ที่จุดศูนย์กลางของภาพ (ดูรูป 2.14)

```
10 CLS
20 screen 1, 0 : color 1, 0
30 draw "C2 TA 30 U 49 TA 150"
40 draw "U 50 TA 270 U 50"
50 draw "TA 60 BU 25 PI, 2"
```

โปรแกรมต่อมาจะแสดงให้เห็นการใช้คำสั่งย่อย X หมุนรูปสี่เหลี่ยมรอบจุดศูนย์กลางเพื่อที่จะสร้าง Decorative Whell (ดูรูป 2.15)

```
10 CLS
20 screen 1, 0 : color 1, 1
30 box$ = " U 40 R 48 D 40 L48"
40 for I = 1TO 360 Step 20
50 draw "TA = 1, X box$; "
60 next I
```

Window Statement and clipping

คำสั่ง window เป็นคำสั่งที่ใช้ในการกำหนดค่า coordinate ของ screen ขึ้นใหม่และในความหมายแบบอื่นคือ window เป็นคำสั่งที่ใช้ในการวาดวัตถุในระบบ coordinate ใหม่ (ในระบบ world coordinate) ที่ไม่มีขอบเขตสิ้นสุดที่ coordinate ของจอภาพ graphic ที่ตามมาจะถูก set ขนาดให้เข้ากับ coordinate ใหม่ ข้อความที่สมบูรณ์ของคำสั่งนี้คือ

Window screen $(x_1,y_1) - (x_2,y_2)$

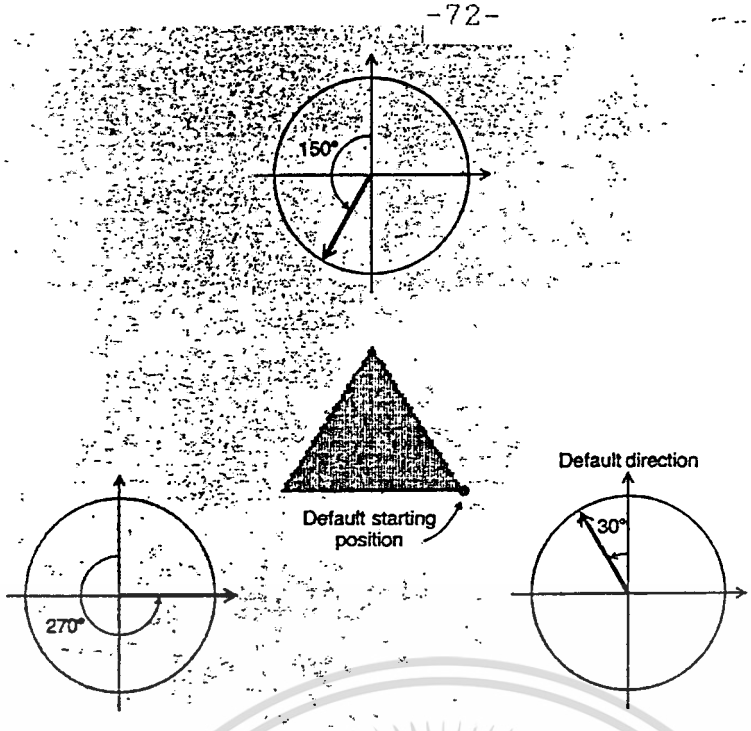


FIGURE 2.14 Triangle drawn by using DRAW statements. The default starting position is the center of the screen.

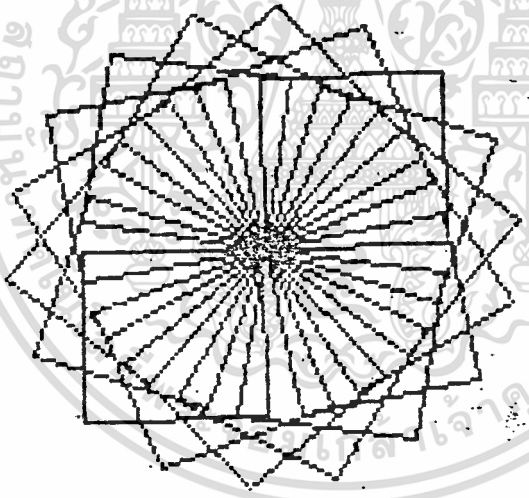


FIGURE 2.15 A decorative wheel created by using the X subcommand.

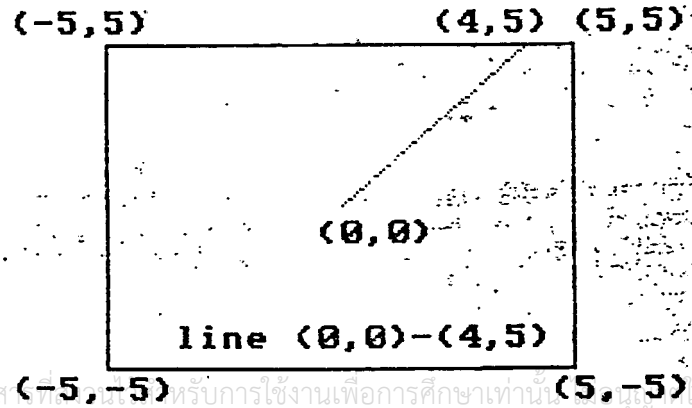


FIGURE 2.16 Effect of the WINDOW statement: The WINDOW statement converts the screen coordinates into Cartesian coordinates.

(x_1, y_1) และ (x_2, y_2) คือ coordinate ที่ผู้ใช้กำหนดขึ้นและจะถูกจะถูกเรียกว่า world coordinate พื้นที่สี่เหลี่ยมผืนผ้าในระบบ world coordinate จะถูกเรียกว่า window เมื่อมีการเรียกใช้คำสั่งนี้ ภาษา Basic จะทำการเปลี่ยนเป็นค่า world coordinate ให้เหมาะสมพอดีกับ coordinate การแสดงผลที่อยู่บนจอภาพ

ถ้าไม่มีคำสั่ง window coordinate ของ PC screen จะมีรูปแบบ coordinate ดังนี้คือ มุมบนด้านซ้ายมี coordinate(0,0) และมุมล่างขวามือมี coordinate (319,199) ใน Mode High Resolution graphic จะมีค่า (639, 199)

เมื่อมีการใช้คำสั่ง window เช่น คำสั่ง window (-10,-10) - (10, 10) และเมื่อถูก execute มุมบนด้านซ้ายของจอภาพจะมีcoordinate (-10, 10) และมุมล่างด้านขวาจะมี coordinate (10, -10) และเมื่อมีโปรแกรมตามหลังคำสั่งนี้ถูก execute ซึ่งโปรแกรมเป็นดังข้างล่างนี้ ผลลัพธ์ที่ได้อยู่ในรูป 2.16

```
10 CLS
20 screen 2, 0,: color 1, 0
30 window (-10,-10) - (10,10)
40 Line (-5,-5) - (5-5), , B
50 Line (0,0) - (4,5), 1
```

หมายเหตุ รูปแบบดังกลาวนี้เป็นรูปแบบธรรมดาที่ใช้ในระบบ Cartesion coodinate คือค่าของ X มีค่าเพิ่มขึ้นในทางขวามือ และค่าของ Y มีค่าเพิ่มขึ้นในทางด้านบน หมายเหตุ จุดศูนย์กลางของ screen (0,0) อยู่ในตำแหน่งตรงกลางของจอภาพ

เมื่อมีการรวม screen attribute ด้วยค่าของ coordinate ก็ยังคงไม่เปลี่ยนแปลง ในกรณีเช่นนี้คำสั่ง window เช่น window (-10, -10) - (10, 10) เป็นการกำหนดscreen ให้มีรูปแบบดังรูป 2.17 รูปแบบดังกลาวนี้สามารถถูกทดสอบโดยการ execute โปรแกรมต่อไปนี้

```
10 CLS
20 screen 1, 0 : color 1, 0
30 window screen (-10,-10) -(10,10)
40 Line (-5,-5) - (5,5) , , B
50 Line (0,0) - (4,5) , 1
```

คำสั่ง window ยังมีการใช้ line clipping ซึ่งเป็นกรรมวิธีหนึ่งที่จะตัดจุดทั้งหลายที่อยู่นอกขอบเขตสิ้นสุด window และทำให้จุดทั้งหลายนี้มองไม่เห็นในเนื้อที่ window เพราะฉะนั้นถ้ามีวัตถุรูปหนึ่งรูปใดอยู่ภายใน window เป็นบางส่วนและอยู่ภายนอก window เป็นบางส่วนส่วนที่อยู่ข้างนอก window จะต้องถูกตัดออกเพื่อว่าส่วนที่อยู่ภายใน window จะเป็นส่วนที่มองเห็นด้วยสายตาของผู้ใช้เพียงส่วนเดียว (รูปแบบการ clipping นี้มีความสำคัญมากเพื่อนำมาใช้เกี่ยวกับ computer graphic และจะกล่าวถึงในบทที่ 3 ต่อไป)

Window ยังเป็นคำสั่งที่ใช้ในการ "Zoom" และ "Pon" ถ้า window ที่ใช้มีขนาด coordinate ใหญ่กว่าภาพมากและภาพที่จะปรากฏบนจอจะมีขนาดเล็กลง แต่ถ้าเลือก window coordinate ที่เล็กกว่าขนาดของภาพจะก่อให้เกิดการ clip เกิดขึ้น และทำให้ส่วนของภาพขยายใหญ่ขึ้น โดยการเปลี่ยนแปลงขนาด window เราสามารถ zoom ได้จนกระทั่งวัตถุครอบคลุมเนื้อที่ตลอดทั้งภาพ หรือ Panจนกระทั่งไม่ปรากฏภาพ แต่จะปรากฏเป็นจุดหนึ่ง ๆ บนจอภาพ โปรแกรมต่อไปนี้แสดงให้เห็นลักษณะการ clipping และ zooming ของคำสั่ง window ผลลัพธ์ที่ได้ถูกแสดง ดังรูป 2.18

```
10 CLS : screen 2 : keyoff
20 for I = 1 TO 10
30 X = 200-5* I : Y = 200 -5 * I
40 window (0,0) - (X,Y)
50 Gosub 80
60 Next I
70 END
```

2.3.8 View Statement and Viewport Planning

คำสั่ง view เป็นคำสั่งที่กำหนดส่วนย่อยของพื้นที่การมอง เรียกว่า viewport โดยที่ข้อมูลของรูปภาพในส่วนของภาพนั้นถูกถ่ายโอนมายังพื้นที่การมองนี้
ข้อความของคำสั่งที่สมบูรณ์คือ

View [[screen][(X_1,Y_1) -(X_2,Y_2)[,[color][, [border]]]]]

Parameter ที่เป็นตัวเลือกว่าจะมีหรือไม่มีก็ได้ในคำสั่งนี้จะเป็นส่วนที่ยอมให้ view port ถูกเติมสีและขอบเขตสิ้นสุดด้วยสีที่กำหนดไว้ และคู่ coordinate (X_1,Y_1) (X_2,Y_2) โดยที่ (X_1,Y_1) อยู่ทางด้านบนซ้ายมือและ (X_2,Y_2) อยู่ที่ด้านล่างขวามือของ viewport ที่กำหนดบน screen ค่าเหล่านี้ต้องถูกกำหนดในช่วงจำกัดขอบเขตสิ้นสุดที่แท้จริงบนจอภาพ ส่วน Parameter color นั้นจะเติมสีลงใน viewport ที่กำหนดไว้และ Parameter " boundary" (เป็นสีหนึ่งในช่วง 0 ถึง 3) เป็นส่วนที่วาดเส้นขอบเขตสิ้นสุดของ viewport ถ้าในส่วนที่ Parameter screen ไม่ได้เขียนในรูปแบบนี้จุดทั้งหมดที่ plot ลงบนจอภาพจะสัมพันธ์กับ "viewport" นั่นคือ X_1,Y_1 ต้องบวกเข้ากับ coordinate X,Y ก่อนที่จะทำการ plot ลงบนจอภาพ ถ้ามีการรวม Parameter screen เข้าไปในข้อความแล้วทุกจุดจะ plot เป็นแบบ Absolute และอาจจะอยู่ภายในหรือภายนอกขอบเขตสิ้นสุดจอภาพ เราสามารถกำหนด viewport ได้หลายวันในเวลาเดียวกัน แต่จะมี viewport อันเดียวเท่านั้นที่ทำงานได้ในเวลาหนึ่ง ๆ คำสั่ง RUN, คำสั่ง screen จะเป็นตัวบังคับไม่ให้ viewport ทำงาน

View ยังสามารถขยายหรือลดขนาดของวัตถุโดยการนำวัตถุไปใส่ใน viewport ที่มีขนาดใหญ่และเล็กต่าง ๆ กัน coordinate ของ viewport จะถูกกำหนดโดยการ execute คำสั่ง window ครั้งสุดท้ายและวัตถุที่ตามมาก็จะถูกปรับค่า scale ให้เหมาะสมกับ viewport ต่อไปนี้เป็นตัวอย่างของโปรแกรม

```
10 CLS : screen 1, 0 :color 1, 0
20 window (50,50) - (350,200) ;starting viewport
30 Line (50,50) - (350,200), B
```

```
40 Gosub 100
50 View (20,20)-(150,100), 1,3,; viewport 1
60 Gosub 100
70 view (200,150)-(300,190) ,1,2 ; viewport 2
80 Gosub 100
90,END
100 circle (240,150), 40,2, ,5/18
110 circle (240,150), 40 ,2, ,1
120 Return
```

โปรแกรมนี้จะกำหนด coordinate (0,0) เป็นจุดทางด้านล่างซ้ายมือ และจะทำการวาดวงกลม 2 วงในรูปแบบสีแดง และกำหนดส่วนของจอภาพเป็น Viewport สีเขียวพร้อมทั้งมีขอบเขตสิ้นสุดสีน้ำตาลและจะทำการวาด 0 2 วงเดียวกันนี้แต่อยู่ใน viewport ที่สอง ผลลัพธ์ที่ได้แสดงให้เห็นในรูป 2.19 หมายเหตุ 0 เหล่านี้ได้ถูก scale ใน viewport ที่แตกต่างกัน

2.3.9 Get and PUT Statement

คำสั่ง Get และ Put จะนำมาใช้กับวัตถุที่มีการเคลื่อนที่ด้วยความเร็วสูงและมีประโยชน์อย่างมากในเรื่องเกี่ยวกับการเคลื่อนที่ของคอมพิวเตอร์ (computer Animation) คำสั่ง Get จะอ่านค่าสีของจุดทุกจุดภายในพื้นที่สี่เหลี่ยมเข้าไปใน Array และคำสั่ง Put จะทำการเขียนค่าสีลงบนเนื้อที่ของจอภาพที่กำหนดไว้ รูปแบบของข้อความคำสั่งทั้งสองคือ

Get $(X_1, Y_1) - (X_2, Y_2)$, Arrayname

Put (X_1, Y_1) , Array name [, Action]

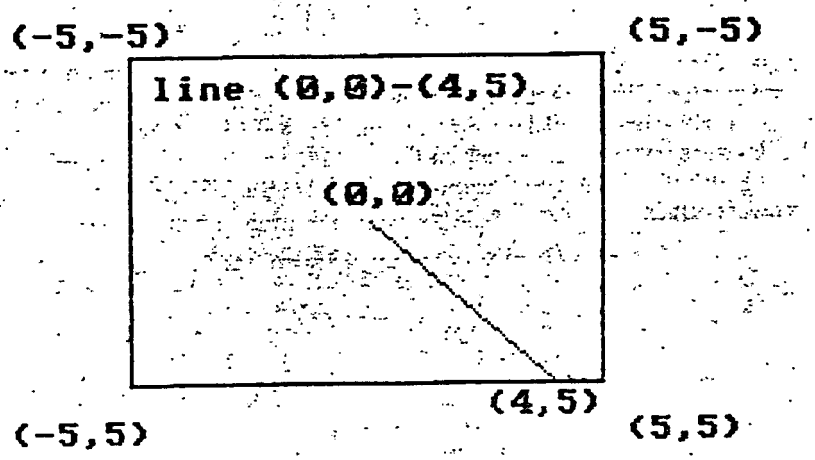


FIGURE 2.17 Effect of the WINDOW SCREEN statement: The WINDOW SCREEN statement converts the Cartesian coordinates into the IBM PC device coordinates.

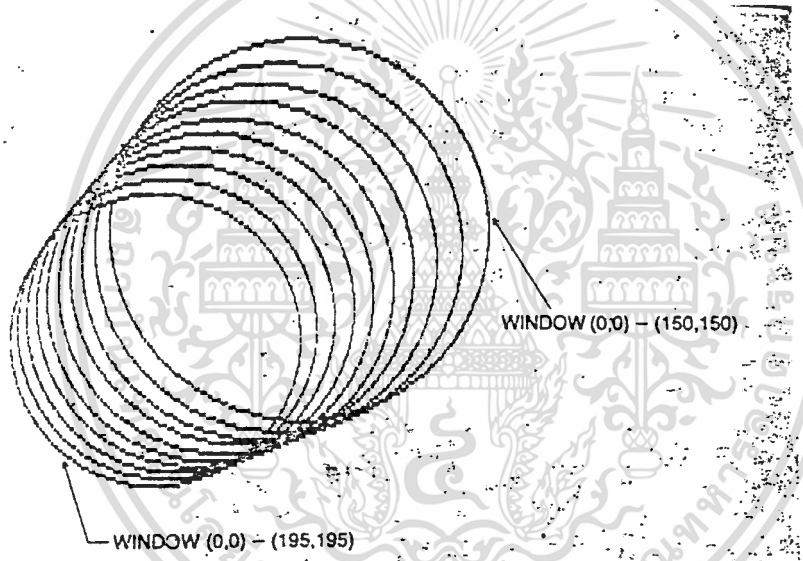
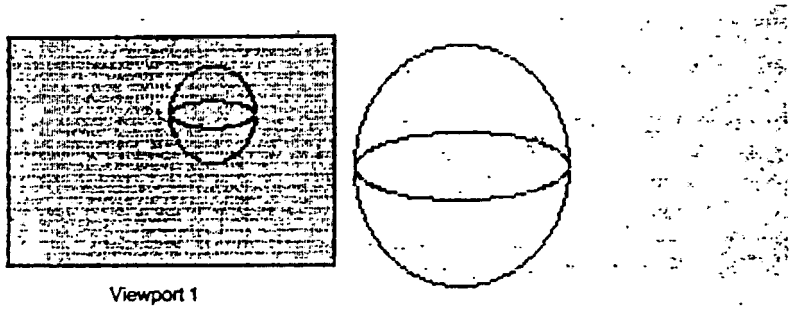


FIGURE 2.18 A series of circles drawn by varying the parameters of the WINDOW statement.



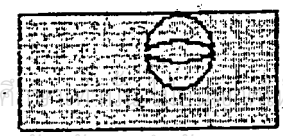


FIGURE 2.19 Effect of VIEW statement (setting multiple viewports).

Get statement

ลำดับเราจะกล่าวถึงคำสั่ง get พื้นที่สี่เหลี่ยมที่กำหนดไว้จะมีมุม (X_1,Y_1) และ(X_2,Y_2) อยู่ตรงข้ามกัน Array จะถูกนำมาใช้ในการเก็บข้อมูลภาพโดยที่จะต้องเป็นตัวเลขและมีค่าที่แน่นอน Array จะต้องมีขนาดและได้กำหนดไว้ก่อนแล้วก่อนที่จะมีการใช้คำสั่ง get

พื้นที่สี่เหลี่ยมของจอภาพที่คำสั่ง get จะทำการเก็บข้อมูลจะต้องมีขนาดและตำแหน่งที่อยู่ อย่างไรก็ตามเราต้องกำหนด Array ที่มีค่ามากเพียงพอที่เก็บข้อมูล เมื่อมีการใช้คำสั่ง get เราสามารถใช้สูตรต่อไปนี้ในการคำนวณหาขนาด Array ที่ต้องการ

จำนวนไบท์ = 4 + INT (X.B + 7)/8 (2.3)

เมื่อ B = Bit Pixel (2 Bit Mode Medium Resolution 1Bit Mode High Resolution)

X = ความยาวทางด้านแนวนอนของสี่เหลี่ยม

Y = ความยาวทางด้านแนวตั้งของสี่เหลี่ยม

สำหรับตัวอย่าง ถ้าเราต้องการ execute คำสั่ง get (10,10)-(19,19),Array ใน Mode Medium Resolution Arrayจะมีขนาดที่จะเก็บข้อมูลได้ขนาด 34 ไบท์ซึ่งได้มาจาก

4 + INT (10*2+7)/8 x 10 = 34 ไบท์

จำนวนไบท์ต่อ element ของ Array คือ 2 ไบท์ต่อ Integer และ 4 ไบท์ต่อ Single Precicion และ 8 ไบท์ต่อ double Precision โดยเหตุนี้เราสามารถใช้ Array เต็มที่มีขนาดอย่างน้อยที่สุด 17 ไบท์ ดังนั้น เราจึงกำหนดรูปแบบของ Array เช่น Array (H)

Put Statement

coordinate 2 (X_3,Y_3) กำหนดให้เป็นมุมบนซ้ายของภาพที่วางอยู่ Array Name คือ Arrayที่ข้อมูลภาพและการกระทำที่เกี่ยวกับภาพนั้นได้นำมาเก็บไว้ในชื่อ Array Parameter ที่มีหรือไม่มีก็ได้ในคำสั่งนี้เลือกได้ 1 ใน 5 วิธีที่ภาพจะถูกนำมาใส่บนจอภาพ ส่วนค่าที่ใช้ในการกระทำคือ Pset, Preset, XOR, AND, OR และจะ set ค่าเป็น XOR เมื่อไม่มีค่าหนึ่งค่าใดใส่ให้

Parameter ตัวนี้ จุดอ้างอิงสุดท้ายคือจุด X_3, Y_3

Put กับ Pset Action จะแสดงส่วนของภาพที่เก็บไว้ภายในด้วยคำสั่ง get และลบข้อมูลที่มีอยู่เดิมออก

Preset Action มีลักษณะคล้ายกับ Pset Action แต่จะทำการใส่ค่าลบของ Image ลงบนจอภาพ ใน Mode Medium Resolution เช่น Color 0 และ 3 จะมีค่าเป็นลบซึ่งกันและกันเช่นเดียวกับ Color 1 และ 2 ใน Mode High Resolution color 0 ,1 จะมีค่าลบซึ่งกันและกัน

การกระทำ AND, XOR, OR ในแต่ละ Array และ Screen color จะเป็นไปตามกฎที่ตั้งไว้ ซึ่งผลลัพธ์การกระทำได้แสดงให้เห็นในตาราง 2.6 สำหรับตัวอย่าง AND Action ระหว่างจุด cyan (color 1) กับจุดสีขาว (color 3) จะได้ผลลัพธ์คือจุด cyan (color 1) or Action ระหว่างจุด cyan (color 1) กับสีขาว (color 3) จะได้เป็นสีขาว (color 3) ส่วน ใน XOR Action นั้นจะให้ค่าสีเป็นสีน้ำเงิน เมื่อสี cyan และสีขาวกระทำต่อกัน

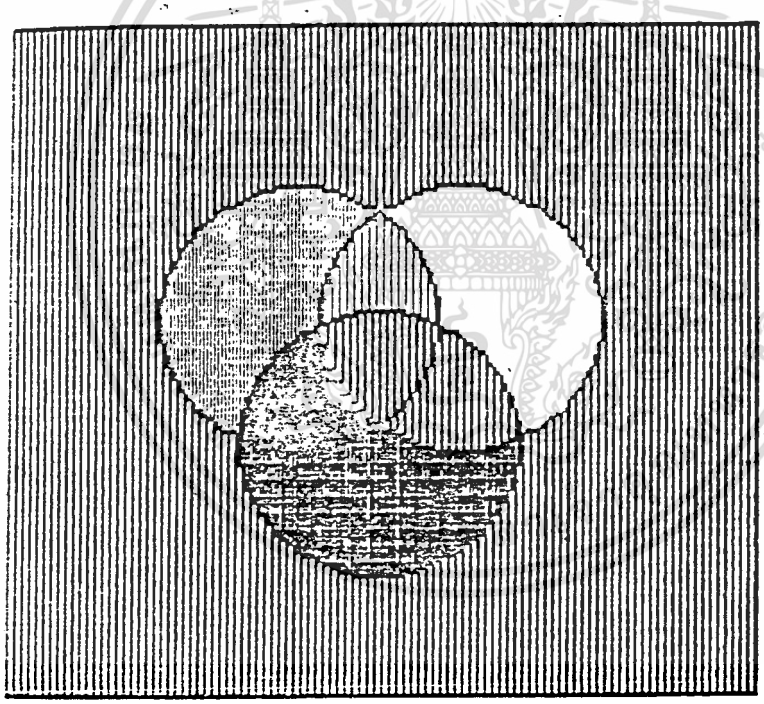
ผลของคำสั่ง Put ที่มีต่อการกระทำ XOR กับ background color ก็คือภาพที่ได้บนจอภาพ และผลของ Put ที่มีต่อการกระทำ XOR กับภาพเดียวกันนี้จะทำให้ภาพในตำแหน่งนี้ถูกลบทิ้งไปโดยเหตุนี้เอง XOR Action สามารถใช้ในการลบภาพได้เช่นเดียวกับการเขียนภาพ ตัวอย่างต่อมาจะแสดงให้เห็นลักษณะบางอย่างในการใช้ get และ put ตามที่กล่าวข้างบน (ดูรูป 2.20 ซึ่งเป็น out put ที่ได้ของ Program)

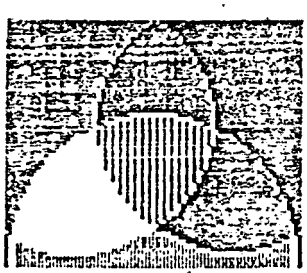
```
10 CLS : screen 1, 0
20 color 1, 0
30 window (-4,-4) - (4,4)
40 view (60,15) - (260,175), , 3
50 circle (0,-1), 1, 5  ; creat Venn diagram
60 circle (-87,5) ,1.5
70 circle (87,.5) , 1.5
```

TABLE 2.6 PUT statement color blending suffixes

Array color	PSET screen color 0 1 2 3	PRESET screen color 0 1 2 3	AND screen color 0 1 2 3	OR screen color 0 1 2 3	XOR screen color 0 1 2 3
0	0 0 0 0	3 3 3 3	0 0 0 0	0 1 2 3	0 1 2 3
1	1 1 1 1	2 2 2 2	0 1 0 1	1 1 3 3	1 0 3 2
2	2 2 2 2	1 1 1 1	0 0 2 2	2 3 2 3	2 3 0 1
3	3 3 3 3	0 0 0 0	0 1 2 3	3 3 3 3	3 2 1 0

Note: PSET plots exactly as captured. PRESET plots a negative image (colors 0 and 3 are negatives of each other, so are 1 and 2).





PUT (10,10), ARRAY, PRESET

Negative image

FIGURE 2.20 Effect of GET and PUT statements

```
 80 paint (0,2), CHR$ (&H33), 3; Fill inthis Section
 90 paint (0.-1), CHR$ (&HAA), 3
100 paint (-.87,.5), CHR$ (&H55), 3
110 paint (0,1), CHR$ (&H11),3
120 paint (-.87,-.5), CHR$ (&HFF),3
130 paint (.87,-.5), CHR$ (&H22), 3
140 paint (0,0), CHR$ (&HCC), 3
150 window : view ; Restors window and viewport
160 DIM Array (1204) ; See Eq. (2.3)
170 Get (120,60)-(200,120), Array; Curve out an area
180 CLS
190 Pot (10,10), Array, Preset; Put the Image in nagative
200 END
```

2.3 Bsave and Bload command

คำสั่ง Bsave จะทำการ save ส่วนของหน่วยความจำคอมพิวเตอร์ลงบนอุปกรณ์ที่กำหนดไว้ ส่วนคำสั่ง Bload จะทำการ load Memory Image file ลงไปในหน่วยความจำ computer Bsave และ Bload มีประโยชน์ในการและ save โปรแกรมภาษาเครื่อง (ซึ่งสามารถถูกเรียกใช้ โดยคำสั่ง call) ใน computer graphic Bsave Bload สามารถนำมาใช้ในการ save และ Restore ภาพของจอภาพโดยการเคลื่อนที่ข้อมูลภาพเข้าไปใน Disk หรือเคลื่อนข้อมูลภาพออกจาก disk ข้อความของคำสั่งทั้งสองเป็นดังนี้

```
Bsave Silename, offset, length
Bload file space [,offset]
```

เมื่อ file space ก็คือ file name, offset คือค่า Address ที่จะทำการ save และ load screen Image file ที่เริ่มต้นในตำแหน่งนี้, length คือความยาวของ screen Image ที่ต้องการ save

ตามอธิบายไว้ในส่วน 2.1.3 หน่วยความ 16 ของ screen Memory ที่ใช้ใน Color / graphic Adapter จะอยู่ที่ Address &B8000

และเราจะใช้คำสั่ง DEF SEG ในการสร้าง Segment Adapter ในตำแหน่งเริ่มต้นของ screen buffer และค่า offset0และ length &H4000 เป็นตัวกำหนดตำแหน่งเนื้อที่ 16K byte โดยตลอด screen buffer ที่นำมาใช้ในการ save ข้อมูลในเนื้อที่ดังกล่าวลง disk

```
10 CLS : screen 2
20 DEF SEG = &HB800
30 circle (320,100), 50
40 paint (320,100)
50 Bsave "picture", 0, &H4000
60 END
```

ในทันทีที่ file ได้สร้างขึ้นแล้วด้วยคำสั่ง Bsave Screen สามารถถูกสร้างขึ้นใหม่ในครั้งที่สองโดยใช้คำสั่ง Bload ตัวอย่างต่อไปจะทำการ load Image file ที่ชื่อ Picture จาก disk ลงบนหน่วยความจำ computer

```
Bload "Picture", 0
```

2.4 Advance graphic programming

ตามที่เห็นมาแล้วว่า ภาษา Basic นั้นก็คือรูปแบบของภาษาอังกฤษที่เป็นภาษาระดับสูง ซึ่งสามารถเขียนโปรแกรมได้ง่าย เข้าใจและก็แก้ไขได้ง่ายอีกด้วย แต่มันมีประสิทธิภาพไม่ค่อยดีและทำงานได้ช้ามาก ๆ ในการประยุกต์ใช้งาน ในเรื่องของความเร็วนั้นเป็นสิ่งสำคัญมากใน graphic computer ในกรณีดังกล่าวนี้ ถ้าเป็นไปได้ผู้ใช้ก็ควรที่จะควบคุมด้วย Processor โดยตรงเพื่อที่จะเกิดประโยชน์อย่างเต็มที่และความสามารถดังกล่าวนี้จะมีทางเป็นได้ ถ้าผู้ใช้มีความเข้าใจในรูปแบบของระบบ ผู้ใช้ก็สามารถที่ระทำงานเกี่ยวข้องกับ software และก็ Hardware ของเครื่องนั้น

Machine language จะเป็นส่วนที่สร้างโปรแกรมที่มีประสิทธิภาพเพราะว่ามันสามารถควบคุมหน่วยความจำได้โดยตรงแต่ภาษานี้มีรูปแบบการใช้ที่ค่อนข้างยาก และเราไม่ค่อยเขียนโปรแกรมได้โดยตรงโดยใช้ภาษาเครื่องบนเครื่อง PCแต่เราใช้ภาษา Assembly ซึ่งเป็นภาษาสัญญลักษณ์ต่าง ๆ ของภาษา

เครื่อง คำสั่งของมันประกอบด้วยการรวมอักษรต่าง ๆ เพื่อให้ง่ายแก่การจำ และตัวอักษรในแต่ละกลุ่มนี้จะแปลเป็นภาษาเครื่องเพียงคำสั่งเดียว เพราะฉะนั้นความเร็วในการ execute จะมีความรวดเร็วมากขึ้นเมื่อเทียบกับภาษา Basic

PC Basic จะสามารถให้การทำงานในระบบทาง world ไกดี คือเราสามารถใช้ภาษา Basic เป็นส่วนของโปรแกรมที่ไม่มีผลต่อความเร็วมากนักและต่อมาใช้คำสั่ง call หรือ USR funtion ใน Program ภาษา Basic เพื่อที่จะกระโดดไปทำงานในตำแหน่งโปรแกรมภาษาเครื่องในส่วนของ Segment ที่ต้องการความเร็วสูงในหัวข้อนี้เราจะแสดงให้เห็นวิธีการเชื่อมโยง Assembly language เข้ากับภาษาBasic การเชื่อมโยงจะมีความสำคัญมาก เมื่อโปรแกรมที่สมบูรณ์ขึ้นด้วยวิธีการเชื่อมโยงเป็นโปรแกรมที่มีประสิทธิภาพเพราะว่าเป็นการรวมกันโดยรูปแบบที่ง่ายและสะดวกของภาษา Basic เข้ากับความเร็วในการทำงานของภาษาAssembly ตัวอย่างการประยุกต์ใช้งาน คือ การวาด Processor อย่างง่าย โดยใช้เทคนิคดังกล่าวนี้

2.4.1 Program Design Drawing Processor

สมมุติว่าเราต้องการ Graphic Program ที่จะให้ผู้ใช้วาดรูปภาพอย่างง่ายโดยไม่มีการเขียน Graphic Program ส่วนอื่นเพิ่มเติม

จุดประสงค์ของเราก็คือ การเขียน Program graphic ด้วยภาษา Basic ที่เชื่อมโยงเข้ากับ Subroutine ภาษาเครื่องและเราจะเรียกโปรแกรม graphic นี้ว่า Drawing Processor Drawing Processor มีส่วนประกอบสำคัญ 2 ส่วนคือ ส่วน Main Program นั้นเขียนโปรแกรมด้วยภาษา Basic และส่วนของ Subroutine ที่เขียนด้วยภาษา Assembly เราจะเรียกโปรแกรม Main นี้ว่า Sdraw และ Subroutine ของภาษา Assembty ว่าKBdraw และ Sdraw Program จะทำหน้าที่เกี่ยวกับ Input/output ของ Drawing Processor และผ่านคำ Key board I/O ไปยัง Kbdraw Program ที่ทำหน้าที่ควบคุมขั้นตอนในการวาด

Sdraw นั้นจะทำหน้าที่เป็นตัวชี้แนะ Program Drawing Processor

ตามที่แสดงไว้ข้างล่างนี้ คือมันจะแสดงเป็น menu พร้อมกับตัวเลือก 4 ตัว เมื่อมีการเลือกเกิดขึ้นก็จะมีการ call ไปยังโปรแกรมย่อย Kbdraw ที่เขียนด้วยภาษา Assembly และจะทำการควบคุมตามตำแหน่ง Key board ที่ผู้ใช้กด ทั้งที่เป็นรูปแบบที่สร้างขึ้นใหม่ (ตัวเลือกที่ 1) หรือดัดแปลงรูปแบบที่มีอยู่แล้ว (ตัวเลือกที่ 2)

Kbdraw Subroutine ยังได้เตรียม funtion Key 9 funtion ที่จะทำการ Seton Pixel บนจอภาพ, เคลื่อนที่ Pixel, Polt จุดลงในทิศทางที่กำหนดไว้, เปลี่ยนแปลงสี foreground และ Backgrouns รายละเอียดที่สมบูรณ์ของ funtion จะให้ไว้แล้วในตาราง 2.7 ส่วน flow chart ของโปรแกรมนี้อยู่ในรูป 2.21 และรายละเอียดของ codeจะแสดงให้เห็นในตอนท้ายของส่วนนี้ ในส่วนถัดไปเราจะแสดงขั้นตอนที่เกี่ยวข้องในการสร้าง Kbdraw และเชื่อมโยง Kbdraw เข้ากับ Sdraw เราจะอธิบายให้เห็นถึงรายละเอียดในหัวข้อเหล่านี้

- วิธีการแบ่งหน่วยความจำของ Subroutine
- วิธีการใส่ Subroutine ภาษา Basic (Sdraw) เข้าไปใน Memory
- วิธีการเรียก Subroutine จากภาษา Basic (Sdraw) และผ่านค่า Parameter ไปยังโปรแกรมย่อย (Kbdraw)

2.4.2 The Step in Developing an Assembly Language Subroutine

ในการเขียนโปรดแกรมภาษา Assembly ของเครื่อง IBM PC ผู้ใช้จำเป็นต้องรู้ต้องเข้าใจในสิ่งเหล่านี้

1. การทำงานของ 8088 และกลุ่มคำสั่ง
2. Input และ output Programming โดยเฉพาะการจัดการที่เกี่ยวกับการ Interrupt
3. IBM macro Assembly
4. DOS

TABLE 2.7 Function keys for DRAWING PROCESSOR

Key	Function	Instructions
[F1]	To select background color. Default: black	After pressing key [F1], a digit key between 0 and 9.
[F2]	To select color palette Default: palette 0	After pressing key [F2], press a digit key, either 0 or 1.
[F3]	To select dot color. Default: 1 = green 2 = red 3 = yellow	After pressing key [F3], press a digit key, either 1, 2, or 3.
[F4]	To locate a pixel on the screen. Default: upper left corner pixel (0, 0)	After pressing key [F4], enter (1) the x-coordinates (000–319) by pressing three digits in sequence, then immediately (2) the y-coordinates (000–199) by pressing three digits in sequence.
[F5]	To select the x and y increments (in pixels). Defaults: x increment = 1 y increment = 1	After pressing key [F5], press 1) a digit between 0 and 9 to set the x increment, 2) a digit between 0 and 9 to set the y increment.
[F6]	To draw	Use the cursor keys in the numeric pad (to the right of the keyboard) to move the dot in the direction indicated by the arrow.
[F7]	To move the dot without drawing.	Same as [F6].
[F8]	Not used.	—
[F9]	To move the dot to the home position (upper left corner).	Press key [F9].
[F10]	To terminate the drawing and save the picture under the filename specified before. (Control returns to the BASIC program.)	Press key [F10].

Note: Unless specified, the numeric pad cannot be used.

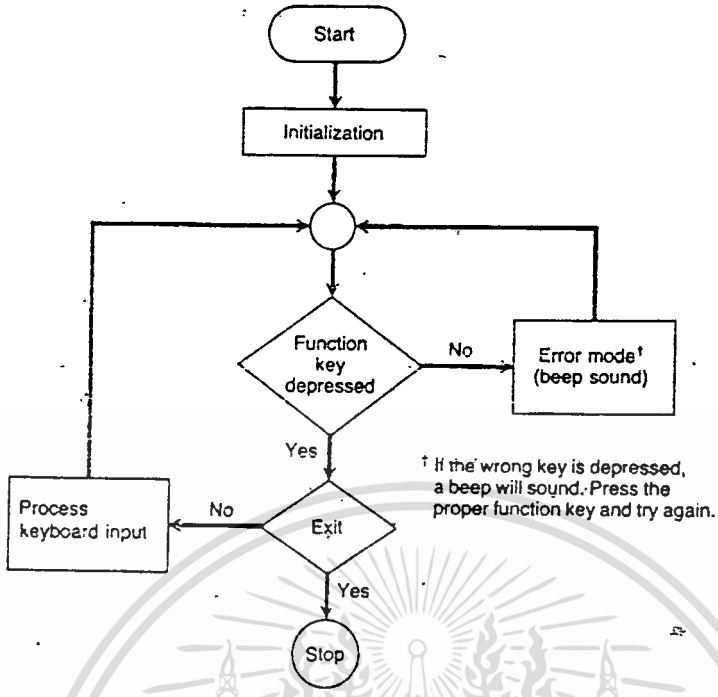


FIGURE 2.21 Programming logic for DRAWING PROCESSOR

TABLE 2.8 Logging procedure: DEBUG

```
B>DEBUG BASIC.COM
-R
AX=0000  BX=0000  CX=2C80  DX=0000  SP=FFF0  BP=0000  SI=0000  DI=0000
DS=04B5  ES=04B5  SS=04B5  CS=04B5  IP=0100   NV UP DI PL NZ NA PO NC
04B5:0100 E9032A        JMP     2B06
-N B:KBDRAW.EXE
-L
-R
AX=0000  BX=0000  CX=0980  DX=0000  SP=0800  BP=0000  SI=0000  DI=0000
DS=04B5  ES=04B5  SS=1F80  CS=1F68  IP=003B   NV UP DI PL NZ NA PO NC
1F68:003B 2E            SEG     CS
1F68.003C C606010001    MOV     B,[0001],01                 CS:0001=01
-R CX
CX 0980
:2C80
-R SP
SP 0800
:FFF0
-R SS
SS 1F80
:04B5
-R CS
CS 1F68
:04B5
-R IP
IP 003B
:0100
-R
AX=0000  BX=0000  CX=2C80  DX=0000  SP=FFF0  BP=0000  SI=0000  DI=0000
DS=04B5  ES=04B5  SS=04B5  CS=04B5  IP=0100   NV UP DI PL NZ NA PO NC
04B5:0100 E9032A        JMP     2B06
-G
DIRECT STATEMENT IN FILE
OK
DEF SEG=&H1F68
OK
BSAVE "B:KBDRAW.EXE",0,&H174
OK
SYSTEM
PROGRAM TERMINATED NORMALLY
-Q
```

Notes: (1) Prefix - indicates the current cursor position.
(2) Either lowercase or uppercase characters can be used in programming.

5. Software Monitor ที่ได้เตรียม code ที่จำเป็นต่อการ Interface เข้ากับระบบ Hareware ส่วนขั้นตอนที่จำเป็น 3 อย่าง ในการออกแบบสร้างโปรแกรมภาษา Assembly คือ

1. เขียน Soure Program โดยการใช้ Text Editor, ... Edlin

2. Assembling Source โดยการใช้ Macro Assembler เพื่อสร้าง Object code

3. Link object code เพื่อสร้างส่วนที่ทำการload executable ที่สามารถนำมาใช้ได้โดยการใส่ชื่อของ object code นี้เข้าไปในระบบ

จุดประสงค์ของเราก็คือ วิธีการ link Subroutine ที่เขียนด้วยภาษา Assembly เข้ากับ Main Program ที่เขียนด้วยภาษา Basic เราจะเรียกโปรแกรมทั้งสองนี้ว่า Kbdrawm Sdraw ตามลำดับในการกระทำดังกล่าวนี้จะต้องทำตามขั้นตอนทั้ง 3 ส่วนที่กล่าวมาแล้ว และต้องจำไว้ว่า load Module ควรที่จะทำการ load ที่หน่วยความจำระดับสูง เพราะว่าจะเป็นการป้องกันการทับกันของข้อมูลมิฉะนั้นแล้วเหตุการณ์บางอย่างที่ไม่ดีอาจเกิดขึ้น ส่วนต่าง ๆ ที่ตามมาจะช่วยชี้แจงให้เข้าใจขั้นตอนเหล่านี้

Editing the source program

การใช้ IBM PC Text editor เช่น Edlin จะต้องร้องขอต่อ Dos โดยการใส่คำสั่ง Edlin (ภายหลัง Dos Pront เกิดขึ้นแล้ว) แล้วกด Enter Key (แสดงโดยเครื่องหมาย ←┘) text Edlin นี้จะยอมให้ผู้ใช้เขียน ASC II source program ของ code ภาษา Assembly และเก็บโปรแกรมลงในแผน disk ในกรณีของเรานั้น sourge program นี้ถูกเขียนในชื่อของ Kdraw , ASM

สมมุติอยู่ในสภาวะการควบคุมของ Dos และใน drive b มี Diskette ที่มีชื่อ Kbdraw ASM เก็บอยู่ ภายหลัง Dos Pront เกิดขึ้นแล้วให้ใส่คำสั่ง edlin ตามด้วย filename (ในที่นี้คือ Kbdraw ASM) และกด Dos จะบอกให้เราทราบว่าอยู่ใน Edit Mode โดยที่หน้าจอจะมีข้อความดังนี้

New file

New file หมายถึง file ใหม่ที่เราต้องการเขียน และเครื่องหมาย * คือ Edlin promt และ Undeslin คือตำแหน่ง cursor บนจอภาพ (ถ้าท่านกำลังเขียน file ที่มีอยู่แล้ว, ขอความ "new file" จะถูกแทนที่โดยขอความ "End of input file"

Edlin จะมีคำสั่งให้เลือกใช้หลายคำสั่ง เช่น Insert (I), List (L), Append (A), write (W), delete (D),Replace Text(R) และ search Text (S) จนการเขียนและต้องการเก็บโปรแกรม (E) และออกจาก Mode Edit โดยไม่มีการ save โปรแกรมที่เขียนไว้ (Q) (สำหรับรายละเอียดของคำสั่งเหล่านี้จะมีอยู่ใน Dos commanp)

การเขียนโปรแกรม Kbdraw ของเราจะทำได้โดยการใช้คำสั่งเหล่านี้ (รายละเอียดที่สมบูรณ์ของ program จะแสดงให้เห็นในส่วนท้ายของส่วนนี้

Assembling the Source Program

ในขั้นตอน Assembly Object code ของ Source จะถูกสร้างขึ้น ใน code เหล่านี้คำสั่งต่าง ๆ จะถูกแปลงเป็นตัวเลขฐานสองที่แทนแต่ละคำสั่งและจะมีผลเกี่ยวข้องเมื่ออยู่ในขั้นตอนการ link ในการ Assemble kbdraw asm program ต้องใช้กำลังดังนี้

Masm kbdraw, ,con :;

คำสั่งที่นำมาใช้นี้จะทำการสร้าง file kbdraw obj. พร้อมกับรายชื่อของ Source โปรแกรม (และค่า Object code ของ Source Program) และก็รายชื่อ Object code เปรียบเทียบ Listing เหล่านี้ถูกแสดงให้เห็นทางจอภาพและอาจจะสะท้อนให้เห็นโดยการแสดงออกทาง Printer ก็ได้ (โดยการกด CNTR, กับ PRTSC) ถ้ามีความผิดพลาดบางอย่างใน Assembling ก็จะแสดงผลให้ทราบโดยทันทีในกรณีนี้ทั้ง Source และ object code จะถูกเปลี่ยนแปลงค่าใหม่ใน Debugging สำหรับรายละเอียดเกี่ยวกับ MASM จะมีใน IBM MACRO Assembler Manval

Linking the object code

ทันทีที่ได้ object code แล้วและ object code นี้จะต้องไม่มี การ error มันก็สามารถนำมาเชื่อมโยงเข้ากับ Subroutine อื่นได้โดย การใช้ Link Program ได้ดังนี้

.Link kbdraw, , con :/H ;

คำสั่งนี้จะเป็นการสร้างส่วน Load Module ในชื่อของ Kbdraw EXE และแสดงผลการ Map ของ load Module การMap นี้จะทำได้ อย่างมีประโยชน์เมื่อมีการ link เข้ากับ program ภาษา Basic หมายเหตุ ในที่นี้จะมี switch H. ซึ่ง นี้เป็นตัวชี้ให้ Dos ทำการ load ส่วนของ load Module ในหน่วยความจำตำแหน่งสูงเพื่อที่จะกันการทับกันของข้อมูล ถ้า Parameter นี้ข้ามเลยไปอาจมีผลที่ไม่ต้องการเกิดขึ้นได้

การ Map ของ kbdraw EXE แสดงให้เห็นว่า code program มีความยาว 074H byte และ Address ที่จะใส่คำสั่งแรกของโปรแกรม นี้อยู่ที่ 0000–003B ความยาวขนาด 0174 H byte จะต้องบันทึกไว้เพื่อนำมา ใช้ในครั้งต่อไป ตัวอย่าง

A >link kbdraw , , con : /h :

ภายหลังกด RETURN ก็จะปรากฏรายละเอียดต่างๆ ดังแสดงไว้ในรายงานท้าย

2.4.3 Linking an Assembly Subroutine to a Basic Program

ในการ Link kbdraw เข้ากับ sdraw จะต้องใช้ระบบ Program Debug ตามที่แสดงไว้ในตาราง 2.8 ในลำดับแรกนั้น debug จะต้อง ถูก execute ภายใต้ Dos ก่อน โดยการใส่คำสั่งดังนี้

Debug basic com

ภายหลังได้รับคำสั่งควบคุมแล้ว debug จะโหลดโปรแกรม basic ที่จุด CS : IP เมื่อ CS ต่อ Register และ IP คือ Pointer ในที่นี้คือ (04B5 :0100) และเมื่อเพิ่มแล้วจะมีเครื่องหมาย Pront ขึ้น โดยการใส่ command R เพื่อแสดงผลข้อมูลที่อยู่ภายใน Register ทั้งหมดและ flag เงื่อนไข, บันทึกค่าเหล่านี้ไว้เพราะจะต้องนำมาใช้ในภายหลังต่อไป เราต้อง

load หน่วยความจำของ load Module ของ Application Program (file kbdraw EXEสมมุติว่าอยู่ใน drive b) โดยการใส่คำสั่ง 2 คำสั่งดังนี้

```
N B : Kbdraw.EXE
L
```

การทำดังกล่าวนี้จะทำการ load kbdraw EXE high ในหน่วยความจำตำแหน่ง CS :IP ซึ่งสามารถหาได้จากการแสดง Register ทั้งสอง โดยใช้คำสั่ง R ในกรณีนี้ Kbdraw EXE จะถูก load เข้ามาที่ตำแหน่ง CS:IP : 1F68 : 003B และต้องบันทึกค่านี้ไว้เพราะว่ามันคือตำแหน่ง Address ของ kbdraw ที่ถูกเรียกโดย sdraw ของโปรแกรม basic ในขณะที่เรากำลังทำอยู่ให้ดูที่ข้อมูลในส่วนนี้และต้องแน่ใจว่ามันเป็นคำสั่งแรกของ Kbdraw มาถึงจุดนี้ทั้ง basic และ kbdraw ถูกโหลดเข้าไปในหน่วยความจำเรียบร้อยแล้วและพร้อมที่จะถูก link เข้าด้วยกัน โดยลำดับแรกการควบคุมต้องถูกส่งกลับมายัง basic โดยการเก็บค่า Register และข้อมูลทั้งหมดที่อยู่ภายใน Register เหล่านี้เมื่อภาษาbasic ถูก load เข้ามาครั้งแรก โดยการใช้คำสั่ง R เพื่อเก็บค่าเหล่านี้และต่อมาก็ตรวจเช็คว่าค่าทั้งหลายเหล่านี้ได้ถูกเก็บเรียบร้อยแล้ว

ในขั้นตอนที่ 2 ใส่คำสั่ง 6 เพื่อให้ระบบอยู่ภายใต้การควบคุมของ basic โดยที่ basic จะบอกให้ทราบด้วยคำว่า OK ใน Direct Mode ให้ใส่คำสั่งตามนี้

```
DEF SEG = &H1F68
OK (Basic Promt)
```

และ Bsave "B : kbdraw.EXE" , 0 , &H174

```
OK (Baisc Promt)
```

ในคำสั่งแรก 1F68 คือส่วน Address ที่ kbdraw EXE ตั้งอยู่ในหน่วยความจำก่อนหน้านั้น ในคำสั่งที่ 2 kbdraw EXE จะถูก save ลงบนแผ่น disk ใน drive b โดยเริ่มต้นที่ offset 0 มีความยาว 174H byte (โดยความยาวนี้หาได้ในช่วงการ load ก่อนหน้านี้)

ต่อมาให้การส่งการ control กลับไปยัง debug ทำได้โดยการใส่คำสั่ง basic คือ system ถ้าการส่งกลับมาเป็นปกติ debug จะแสดงให้เราเห็นด้วยข้อความต่อไปนี้

Program terminate normally

ที่จุดนี้ กรรมวิธีการ link เสร็จสมบูรณ์แล้วและ debug สามารถถูกทำให้สิ้นสุดลงโดยการใช้คำสั่ง Q ซึ่งจะส่งการควบคุมคืนสู่ Dos

ภายหลังข้อมูล kbdraw EXE ได้ถูก save แล้ว Assembly language Subroutine kbdraw สามารถถูกเรียกใช้จาก basic program sdraw ที่เวลาใดก็ได้ตามต้องการโดยการรวมข้อความ 3 ประโยคต่อไปในโปรแกรม (ดูโปรแกรม 2.1)

```
520 DEF SEG = &H 1F68        ; Segment kbdraw
530 kbdraw = &H3B    ; Load Point for kbsraw
540 call kbdraw
```

อย่างไรก็ตามก่อนที่จะทำการเรียกข้อมูล kbdraw EXE ต้องถูกโหลดเข้ามาในหน่วยความจำก่อน รูปแบบนี้สามารถกระทำได้โดยใช้ภาษา basic ดังนี้

```
230 DEF SEG = &H1F68
240 BLoad "B : kbdraw EXE", 0
```

ในการเก็บค่า Code Segment Register(CS) ไปให้ basic ต้องแน่ใจว่า 2 ข้อความนี้ต้องตามด้วย

```
150 DEF SEG
```

ส่วนตัวอย่างโปรแกรมภาษา Basic เป็นดังโปรแกรม 2.1

2.4.4 Application Program

Draw Processor ที่แสดงให้เห็นในส่วนนี้ไม่ค่อยจะสมบูรณ์จะต้องมีการแก้ไขบางส่วน มันถูกนำมาใช้เพื่อแสดงให้เห็นการรวมภาษา basic เข้ากับภาษา Assembly ที่ทำให้ผู้ใช้สามารถนำไปประยุกต์ใช้งานได้รวมกับภาษาเครื่อง

รูป 2.2.2 เป็นการวาดโดยใช้ kbdraw สามารถสร้างโดยการเรียกผ่าน Basic และจุดประสงค์ของเราคือ วิธีการเชื่อมโยงภาษา Basic เข้ากับภาษาเครื่องดีไดนามากล่าวไว้แล้วในที่นี้

ผู้อ่านควรที่จะหาวิธีการเพิ่มเติมในวิธีการบางอย่างของ edit, Assembling, link โปรแกรมประยุกต์ใช้งาน (โปรแกรม 2.1) และการเพิ่มเติมในส่วนต่าง ๆ เหล่านี้เท่าที่จะเป็นได้ ความรู้สึกนักคิดของผู้ใช้ จะตัวบังคับให้สิ่งเหล่านี้สามารถที่จะทำได้

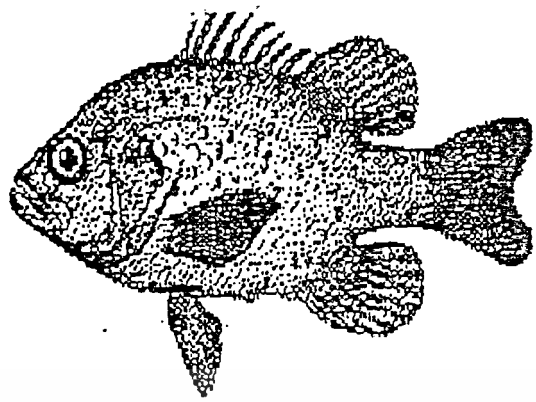


FIGURE 2.22 Fish drawn by the DRAWING PROCESSOR program

```
100 ' Program 2.1: Drawing Processor
110 KEY OFF:SCREEN 0,1:WIDTH 80:COLOR 11,0:CLS
120 DFLT=0
130 DEF SEG=&H1F68
140 BLOAD "b:kbdraw.exe",0
150 DEF SEG:
160 MSG1$="Screen Drawing":MSG2$="Draw a New Picture":MSG3$="Modify a Picture"
170 MSG4$="Display a Picture"
180 GOSUB 350:ON NUM GOTO 190,260,290,660
190 CLS:MSG$=MSG1$:GOSUB 650:PRINT:MSG$=MSG2$:GOSUB 650
200 LBLANK=5:GOSUB 470
210 GOSUB 480
220 SCREEN 1,0:COLOR 0,1:CLS
230 GOSUB 510
240 GOSUB 570
250 GOTO 180
260 CLS:MSG$=MSG3$:GOSUB 590
270 GOSUB 550:GOSUB 520:GOSUB 570
280 GOTO 180
290 CLS:MSG$=MSG4$:GOSUB 590
300 GOSUB 550
310 DEF SEG=0:POKE 1050,PEEK(1052)
320 V$=INKEY$:IF V$="" THEN 320
330 WIDTH 80:SCREEN 0,1:COLOR 11,0:CLS
340 GOTO 180
350 LOCATE 2,1
360 LOCATE 2,1:FOR I=1 TO 80:PRINT"a";:NEXT
370 LOCATE 25,1:FOR J=1 TO 80:PRINT"a";:NEXT
380 A$="Screen Drawing":LOCATE 5,(80-LEN(A$))/2,1:PRINT A$;
390 A$="Primary Option Menu":LOCATE 6,(80-LEN(A$))/2,1:PRINT A$;
400 PRINT:PRINT:PRINT:PRINT:PRINT:LOCATE ,10:PRINT"Options available :":PRINT
410 LOCATE ,29:COLOR 14:PRINT"- draw a new picture...1"
420 LOCATE ,29:PRINT"- Modify a picture.....2"
430 LOCATE ,29:PRINT"- Display a picture....3"
440 LOCATE ,29:PRINT"- exit.....4"
450 COLOR 11:PRINT:PRINT:PRINT:LOCATE ,10:INPUT "Enter option number ";NUM
460 RETURN
470 FOR I=1 TO LBLANK:PRINT:NEXT:RETURN
```

PROGRAM 2.1 Drawing processor

```
480 LOCATE ,10:COLOR 14 :INPUT "Enter Filename(8 char. max.)";FILE$:PRINT
490 LOCATE ,10:INPUT "Enter Extension(3 char. max.)";EXT$:PRINT
500 LOCATE ,10:INPUT "Enter Drive <a or b>";DR$:RETURN
510 SCREEN 1,0:COLOR 0,0:CLS
520 DEF SEG=&H1F68
530 KBDRAW=&H3B
540 CALL KBDRAW:RETURN
550 SCREEN 1,0:COLOR 0,0:CLS:DEF SEG=&HB800
560 BLOAD DR$+":"+FILE$+"."+EXT$,0:DFLT=1:RETURN
570 DEF SEG=&HB800:BSAVE DR$+":"+FILE$+"."+EXT$,0,&H4000
580 DFLT=1:WIDTH 80:SCREEN 0,1:COLOR 11,0:CLS:RETURN
590 CLS:GOSUB 650
600 LBLANK=5:GOSUB 470
610 IF DFLT=0 THEN GOTO 480
620 LOCATE ,10:COLOR 14:INPUT  "current filename <y or n>";ANS$
630 IF ANS$<>"y" THEN GOTO 480
640 RETURN
650 LOCATE 5,(80-LEN(MSG$))/2,1:PRINT MSG$:RETURN
660 COLOR 7:CLS:PRINT"End of Session,BYE"
    End of input file
```

```
1:*         Subroutine: KBDRAW
2: comment *
3:          Set_bgnd:   to select the background color from 8 different colors
4:          set_pal:    to select one of 2 palettes.palette 0 or 1 to allow
5:                      different colors.
6:          dot_col:    to select the pixel color.
7:          locate:     to set a pixel on the screen.
8:          inc:        select the increments (in pixels) in the x's and y's
9:                      directions.
10:         mov_d_on:   to set a pixel while moving.
11:         mov_d_off:  to move to a given point without setting the pixel
12: *
13: d
14: stack   segment para    'stack 'stack'    ;setting the stack
15:         db          256     dup('stack    ')
16: stack   ends
17: cseg    segment para    public 'code'
18:         assume  cs:cseg,ss:stack
19:         c_byte  db      10                  ;initialize the constants
20:         flag    db      1
21:         b_color db      0
22:         f_color db      1
23:         d_color db      1
24:         x_delta dw      1
25:         y_delta dw      1
26: al      label   word
27:         dw      offset  set_bgnd            ;setting pointer table
28:         dw      offset  set_pal
29:         dw      offset  dot_col
30:         dw      offset  locate
31:         dw      offset  inc
32:         dw      offset  mov_d_on
33:         dw      offset  mov_d_off
34:         dw      offset  a2
35:         dw      offset  start
36:         dw      offset  exit
37:         dw      offset  a2
38:         dw      offset  a2
39:         dw      offset  north_west
40:         dw      offset  north
41:         dw      offset  north_east
42:         dw      offset  to_err
43:         dw      offset  west
44:         dw      offset  to_err
45:         dw      offset  east
46:         dw      offset  to_err
47:         dw      offset  south_west
48:         dw      offset  south
49:         dw      offset  south_east
50:         dw      offset  to_err
51:         dw      offset  to_err
52: all     equ     $-al
53:         public  kbdraw
54: kbdraw  proc    far                         ;main procedure
55: start:
56:         mov     flag,1
```

PROGRAM 2.1 (continued)

```
57:             mov     cx,00                   ;setting pointer table
58:             mov     dx,00
59: a2:         cmp     flag,1                  ;dot in turned mode off mode ?
60:             mov     al,0
61:             je      c1                      ;yes,dot at the background color
62:             mov     al,f_color              ;no,set the foreground color
63: c1:         mov     ah,0ch
64:             int     10h                     ;set color graphics mode
65:             mov     ah,1
66:             int     16h                     ;polling keyboard for character
67:             pushf
68:             pop     ax
69:             test    ax,0060h                ;character available ?
70:             jz      dispach                 ;yes
71:             cmp     flag,1                  ;no
72:             mov     al,f_color
73:             je      c2
74:             mov     al,0
75: c2:         mov     ah,0ch
76:             int     10h
77:             push    cx
78:             mov     cx,2000
79: delay:      loop    delay                   ;delay for dot blinking
80:             pop     cx
81:             jmp     a2
82: dispach:    mov     ah,0                    ;a char. is available in buffer
83:             int     16h                     ;read its scan code
84:             sub     ah,59                   ;key f1 through f10 depressed ?
85:             jc      to_err                  ;no, error!
86:             mov     al,ah                   ;yes,set up its pointer
87:             xor     ah,ah
88:             sal     ax,1
89:             cmp     ax,18                   ;function key f10 is pressed ?
90:             je      exit                    ;yes,terminate the session.
91:             mov     si,ax
92:             jmp     word ptr cs:[si+offset a1]      ;jump to the desired
93:                                             ;function entry
94: to_err:     call    error
95:             jmp     a2
96: exit:
97:             ret
98: set_bgnd:           mov     bh,0            ;to set background color
99:             call    a3
100:            mov     b_color,bl
101:            jmp     a2
102: set_pal:           mov     bh,1            ;function to set palette
103:            call    a3
104:            jmp     a2
105: locate:                                    ;function to locate a pixel
106:            mov     si,2
107: rep:       mov     cx,3
108:            xor     bx,bx
109: rep1:      mov     ah,0
110:            int     16h
111:            sub     al,30h
112:            cbw
113:            add     ax,bx
114:            cmp     cx,1
115:            je      around
116:            mul     c_byte
117: around:    mov     bx,ax
118:            loop    rep1
119:            push    bx
120:            dec     si
121:            cmp     si,0
122:            jne     rep
123:            pop     dx
124:            pop     cx
125:            mov     flag,1
126:            mov     al,f_color
127:            mov     ah,0ch
128:            int     10h
129:            jmp     a2
130: inc:       mov     ah,0                    ;function to increment x or y
131:            int     16h
132:            sub     al,30h
133:            xor     ah,ah
134:            mov     x_delta,ax
135:            mov     ah,0
136:            int     16h
137:            sub     al,30h
```

PROGRAM 2.1 (continued)

```
138:            xor     ah,ah
139:            mov     y_delta,ax
140:            jmp     a2
141: mov_d_on:  mov     flag,0          ;function to turn on pixel
142:            jmp     a2
143: mov_d_off: mov     flag,1          ;function to turn off pixel
144:            jmp     a2
145: north_west:sub     cx,x_delta      ;move dot to north_west
146: north:     sub     dx,y_delta      ;move dot in north direction
147:            jmp     a2
148: north_east:sub     dx,y_delta      ;move dot to north_east
149: east:      add     cx,x_delta      ;move dot in east direction
150:            jmp     a2
151: south_west:add     dx,y_delta      ;move dot to south_west
152: west:      sub     cx,x_delta      ;move dot in west direction
153:            jmp     a2
154: south_east:add     cx,x_delta      ;move dot to south_east
155: south:     add     dx,y_delta      ;move dot in south direction
156:            jmp     a2
157: dot_col:   mov     ah,0            ;set dot color
158:            int     16h
159:            sub     al,30h
160:            mov     f_color,al
161:            mov     d_color,al
162:            jmp     a2
163: a3         proc    near            ;procedure to select palette
164:            mov     ah,0
165:            int     16h
166:            sub     al,30h
167:            mov     bl,al
168:            mov     ah,0bh
169:            int     10h
170:            ret
171: a3         endp
172: error      proc    near            ;error procedure,sound beep
173:            push    dx
174:            mov     dl,07h
175:            mov     ah,2
176:            int     21h
177:            pop     dx
178:            ret
179: error      endp
180: kbdraw     endp
181: cseg       ends
182:            end     kbdraw
```

PROGRAM 2.1 Drawing processor

```
A>link kbdraw, , con: /h;

IBM Personal Computer Linker
Version 1.10 (C)Copyright IBM Corp 1982


Start     Stop      Length    Name                    CLASS
00000H 00173H 0174H   CSEG                    CODE
00180H 0097FH 0800H   STACK                   STACK

Program entry point at 0000:003B
```

บทที่ 3

Mathematical element in 2D computer graphic

ขณะนี้เราได้รู้เรื่องเกี่ยวกับ computer graphic ในรูปแบบทั่ว ๆ ไปแล้ว ต่อไปเราจะสำรวจวิธีการในการฉายภาพลงบนจอภาพ และจะอธิบายการกระทำต่าง ๆ กับรูปภาพ เช่นว่า เราจะขยายขนาดของรูปภาพให้มีขนาดใหญ่ขึ้น เพื่อที่จะได้ภาพที่มีความชัดเจนหรือจะมีการตัดทอนบางส่วนของภาพ.. ถ้ามีส่วนของภาพที่เห็นมากเกินไป หรือเราอาจจะต้องการหมุนรูปภาพให้ไปอยู่ในตำแหน่งมุมต่าง ๆ เพื่อให้การมองภาพของวัตถุดียิ่งขึ้น ส่วนต่างที่จะกระทำต่อรูปภาพเหล่านี้เรียกว่า Transformation ซึ่งเป็นพื้นฐานของ computer graphic และสามารถทำได้โดยใช้วิธี geometry Transformation ในบทนี้เราจะมีการแนะนำพื้นฐานทางคณิตศาสตร์ที่จำเป็นต่อ computer graphic เช่น การ Transform จุด, เส้น, วัตถุในระนาบ 2 มิติ และยังจะใช้วิธีการเหล่านี้ในการแทนวัตถุ 2 มิติลงบนจอภาพด้วย

3.1 Teansformation of Point

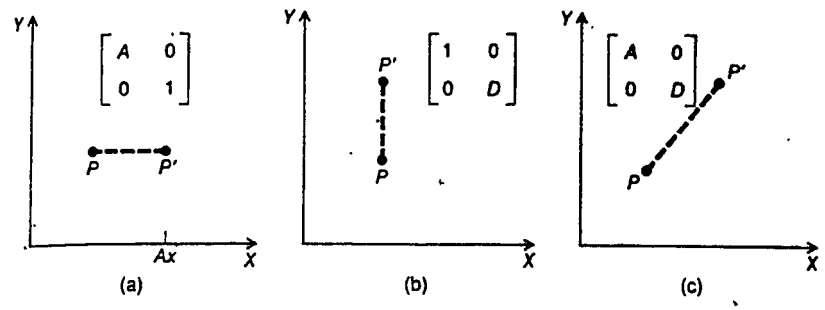
เมื่อเราจะแทนจุด P ลงบนจอภาพ 2 มิติด้วย coordinate (X,Y) เราต้องมีการ Transform ให้ได้จุด P'=(X',Y') ซึ่งได้มาจากผลคูณของจุด P กับ Transformation Matrix ขนาด 2x 2 โดยที่ Transformation Matrix จะมีค่าดังนี้คือ

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

เพื่อให้เห็นได้อย่างชัดเจน จุด P'จะมีค่าดังนี้

$$P' = PT$$

$$= (X,Y)\begin{bmatrix} A & B \\ C & D \end{bmatrix} = (AX+CY), (BX+DY) \qquad (3.1)$$



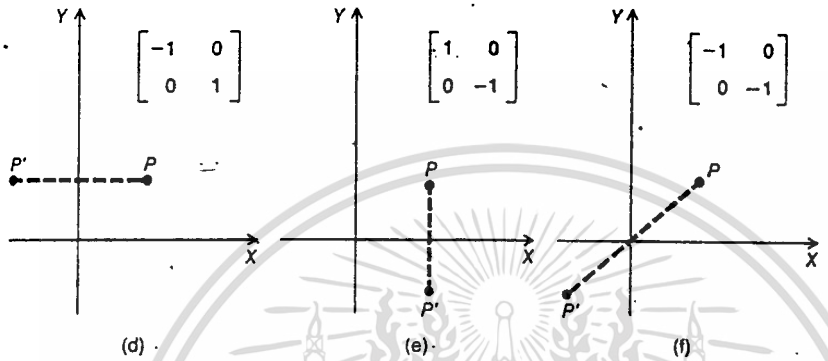


FIGURE 3.1 Transformation of points

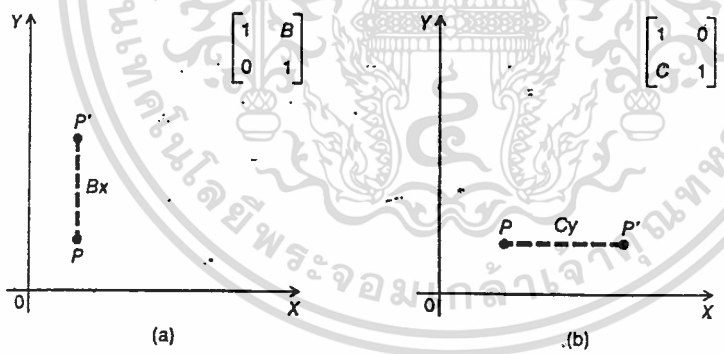
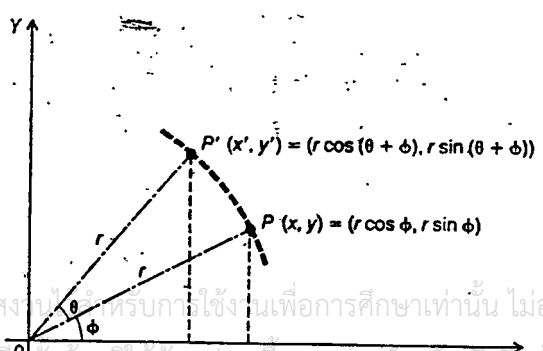


FIGURE 3.2 Shearing effects on points



โดยความหมายทางคณิตศาสตร์แล้ว ผลของ Transformation นี้จะถูกกำหนดโดยค่าที่เหมาะสมของ Parameter A, B, C,D ที่อยู่ภายใน Transformation Matrix ต่อไปเราจะตรวจสอบดูสภาพทั่ว ๆ ไปของค่า A, B, C,D นี้มีผลต่อตำแหน่งของจุด P' อย่างไร

3.1.1 Identity

เมื่อให้ B=C=0 และ A = D= 1 Transformation Matrix จะเป็นแบบ Identity Matrix (เอกลักษณ์)

$$(X',Y') = (X,Y)\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} = (X,Y) \qquad (3.2)$$

จะเห็นได้ว่า Identity Matrix จะไม่ทำให้ coordinate ของจุด P เปลี่ยนไป

3.1.2 Scaling

ในตอนนี้เราจะพิจารณาให้ D = 1 และ B = C = 0 เหมือนตัวอย่างก่อน

$$(X',Y') = (X,Y)\begin{bmatrix}A & 0\\ 0 & 1\end{bmatrix} = (AX,Y) \qquad (3.3)$$

การ Transform ในสมการ 3.3 นี้จะทำการยืดค่าตำแหน่งเดิมของ coordinate X ไปในทิศทาง X เป็นระยะ A เท่า (ดูในรูป 3.1a) และในทำนองเดียวกัน ถ้า Transformation Matrix มีค่า A = 1 และ B = C = 0 จะทำให้เกิดการเปลี่ยนแปลงขนาดในเฉพาะทิศทาง Y เท่านั้น และ Transformation Matrix จะมีรูปแบบดังนี้

$$(X',Y') = (X,Y)\begin{bmatrix}1 & 0\\ 0 & D\end{bmatrix} = (X,DY) \qquad (3.4)$$

ผลที่ได้ของการ Transform แสดงให้เห็นดังรูป 3.1

และเมื่อให้ B = C = 0, Transformation Matrix และผลของการ Transform coordinate จะถูกแสดงในรูปแบบดังนี้

$$(X',Y') = (X,Y)\begin{bmatrix}A & 0\\ 0 & D\end{bmatrix} = (AX,DY) \qquad (3.5)$$

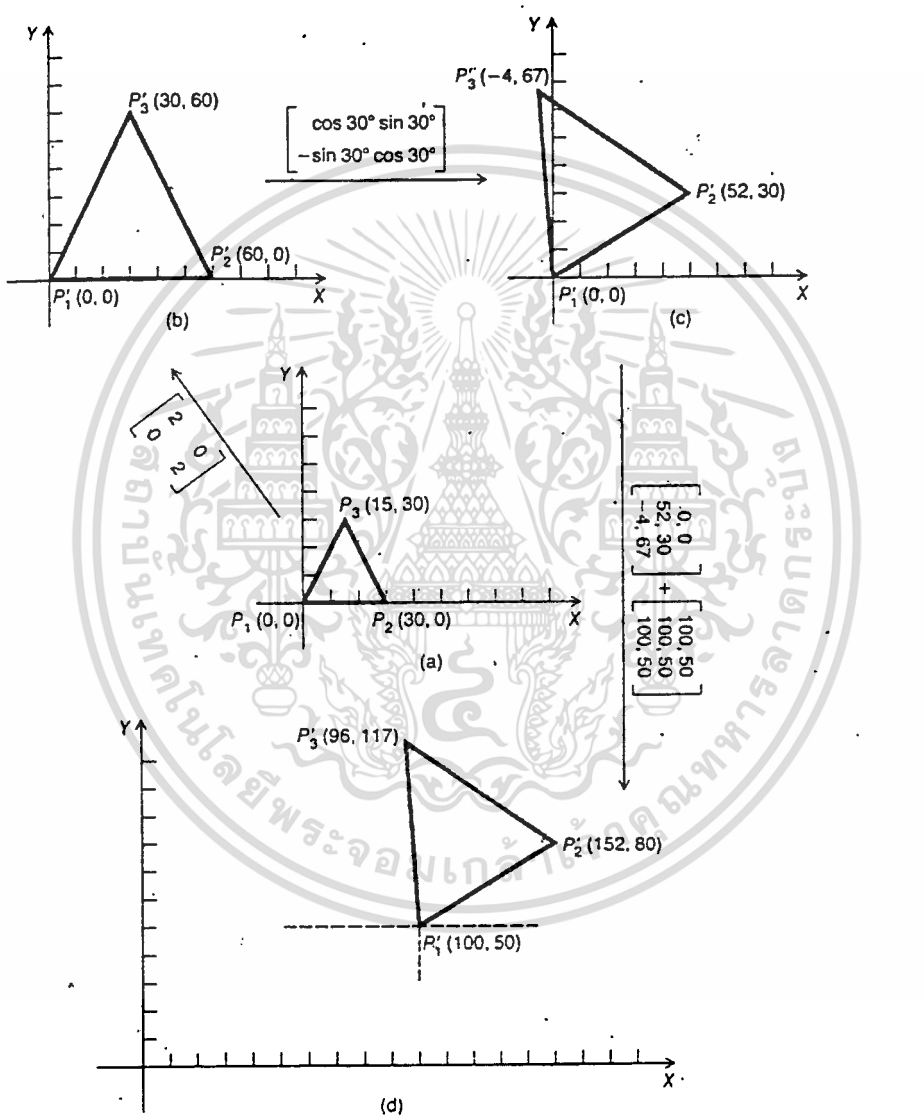


FIGURE 3.4 Transformation of triangles in 2-D space: (a) initial position, (b) scaling, (c) rotation, and (d) translation

ซึ่งลักษณะของ Transformation Matric ในสมการ 3.5 จะก่อให้เกิดการเปลี่ยนแปลงขนาดในทิศทางทั้ง 2 คือ ทิศทาง X และทิศทาง Y ดังแสดงในรูป 3.1 (c) ถ้า $A \neq D$ ก็จะทำให้ได้รูปแบบที่ผิดเพี้ยนไปจากความเป็นจริงและถ้า $A = D$ การขยายขนาดก็จะมีรูปแบบที่สมส่วน

3.1.3 Reflection

การ Reflect จะเกิดขึ้นเมื่อ A หรือ D, A และ D มีค่าเป็นลบใน Transformation Matric ดังแสดงให้เห็นในรูป 3.1 (d) และ 3.1 (1) การ Reflect จุดหนึ่งไปตามแกน X หรือแกน Y นั้นจะเกี่ยวข้องกับการกำเนิดภาพสะท้อนของจุดนั้น ๆ ไปยังด้านตรงข้ามของแกนนั้นเพื่อให้เห็นถึงการ Transform เราจะพิจารณาให้ $B = C = 0$, $A = -1$ และ $D = 1$

$$(X',Y') = (X,Y)\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = (-X,Y) \qquad (3.6)$$

ในกรณีนี้การ Reflect จะเกิดขึ้นตามแกน Y และในทำนองเดียวกัน ถ้า $B=C=0$, $A = 1$ และ $D = -1$ การ Reflect รอบแกน X จะเกิดขึ้น ถ้า $A=D=1$, $B=C=0$ จะได้การ Reflect รอบจุดกำเนิด ส่วนผลที่ได้จากการ Transform เหล่านี้เป็นดังรูป 3.1 (d)และ 3.1(1) และ 3.1 (f)ตามลำดับ หมายเหตุ การ scaling, Reflection ของ coordinate นั้นจะเกี่ยวข้องกับเทอมทะแยงของ Transformation Matric เท่านั้น

3.1.4 Shear

สมมุติว่ามี Transformation Matric ซึ่งมีค่า $A=D=1$ และ $C = 0$ จะได้ $X'Y'$ มีค่าดังนี้

$$(X',Y') = (X,Y)\begin{bmatrix} 1 & B \\ 0 & 1 \end{bmatrix} = (X, BX+Y) \qquad (3.7)$$

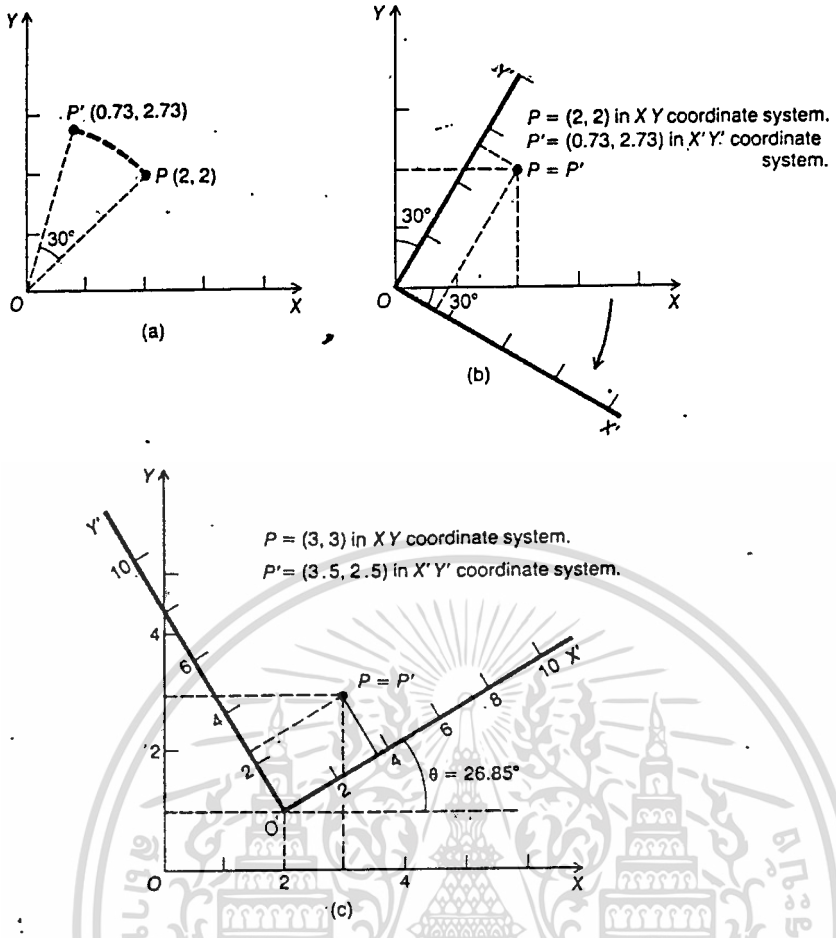


FIGURE 3.5 Relationship between (a) transforming a point, (b) transforming a coordinate system, and (c) point P and two coordinate systems

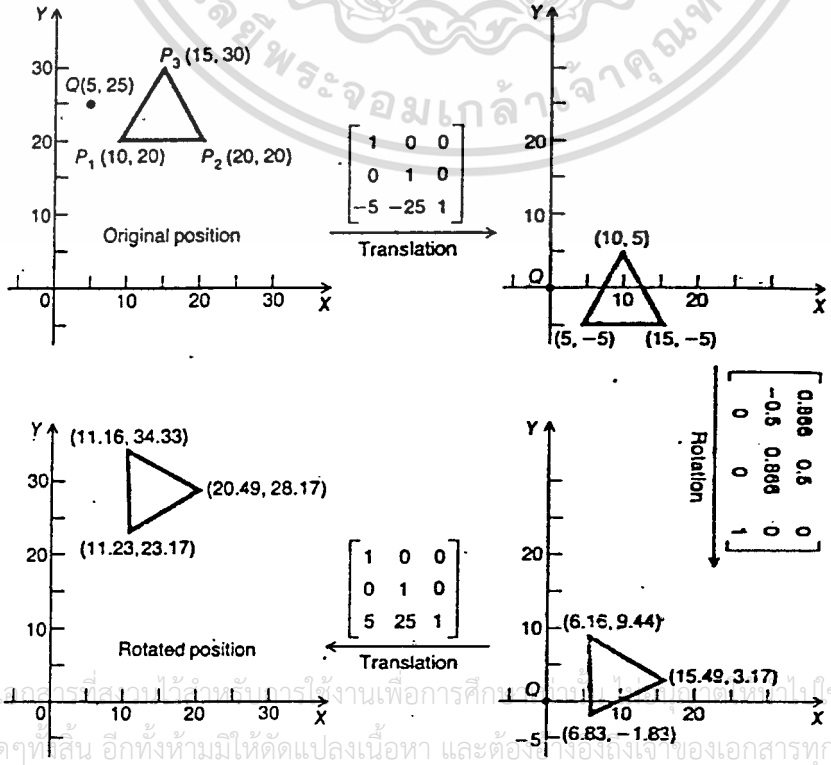


FIGURE 3.6 Two-D rotation about an arbitrary point

การ Transform ในรูปแบบนี้ค่า coordinate X ของจุด P ยังคงไม่เปลี่ยนแปลงในขณะที่ค่า Y จะเป็นค่า funtion เชิงเส้นของ X และ Y ซึ่งลักษณะการ Transform นี้เราเรียกว่า shear ตามทิศทาง Y และถ้าต้องการให้มีการ shear ตามทิศทาง X จะแทนด้วย Transformation Matric ดังนี้

$$(X',Y') = (X,Y)\begin{bmatrix}1 & 0\\ C & 1\end{bmatrix} = (X+CY,Y) \qquad (3.8)$$

การ shear ทั้ง 2 รูปแบบนี้จะมีผลดังรูป 3.2 (a) และ 3.2 (b) (ผลของการ shear จะเห็นได้อย่างชัดเจนเมื่อเรานำมากระทำกับเส้นหรือวัตถุ) สังเกตุได้ว่าเทอมทะแยงมุมทางซ้าย จะทำให้เกิดผลการ shear บน coordinate ของจุด P

3.1.5 Rotation

จุดใด ๆ ก็ตามที่ถูกหมุนเป็นมุม ϕ รอบจุด Origin ในรูปแบบของ Matric เราจะแทนการหมุนนี้โดยค่าดังนี้

$$(X',Y') = (X,Y)\begin{bmatrix}\cos\phi & \sin\phi\\ -\sin\phi & \cos\phi\end{bmatrix}$$
$$= (X\cos\phi - Y\sin\phi,\ X\sin\phi + Y\cos\phi) \qquad (3.9)$$

ค่ามุมบวกจะถูกวัดในทิศทางทวนเข็มนาฬิกาจากค่าแหน่ง X ไปยัง Y เมื่อจุดใดจุดหนึ่งเคลื่อนที่ไปในทิศทางตามเข็มนาฬิกา (จะได้ค่ามุมเป็นลบ) และเราจะใช้ค่ามุม $(-\phi)$ แทนค่าใน ϕ ที่อยู่ในสมการ (3.9) และเรารู้ว่า $\cos(-\phi) = \cos(\phi)$, $\sin(-\phi) = -\sin\phi$ ∴ ผลการหมุนตามเข็มนาฬิกาเป็นมุม ϕ จะมีค่าดังนี้คือ

$$(X',Y') = (X,Y)\begin{bmatrix}\cos\phi & -\sin\phi\\ \sin\phi & \cos\phi\end{bmatrix}$$
$$= (X\cos\phi + Y\sin\phi,\ -X\sin\phi + Y\cos\phi) \qquad (3.10)$$

ผลลัพธ์ที่ได้ของแต่ละส่วน X',Y' ของ Retate Matrix เรายังไม่รู้แน่ชัดว่าได้มาอย่างไร ในตอนนี้เราก็จะแสดงวิธีการหาค่าส่วนประกอบ X',Y'

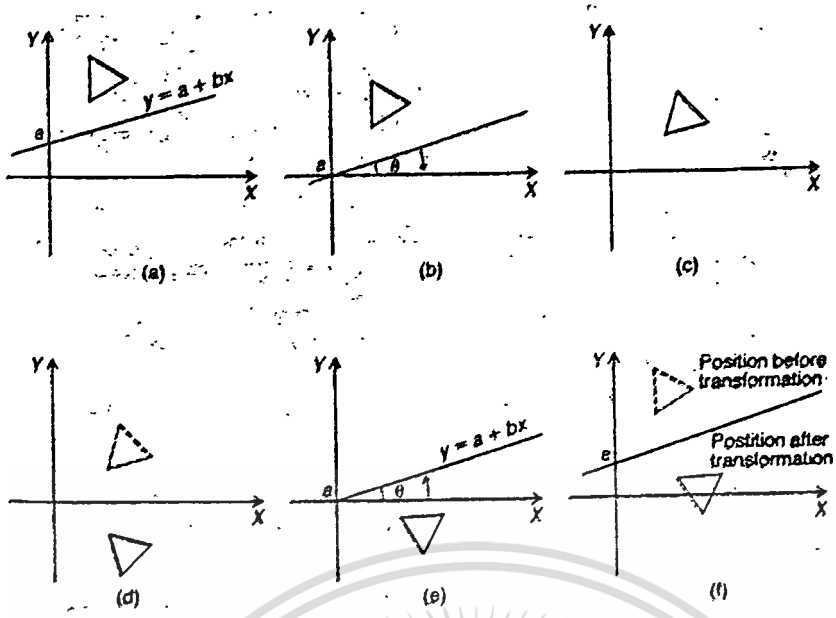


FIGURE 3.7 Reflection around an arbitrary axis: (a) translation, (c) rotation, (d) reflection, (e) rotation, and (f) translation

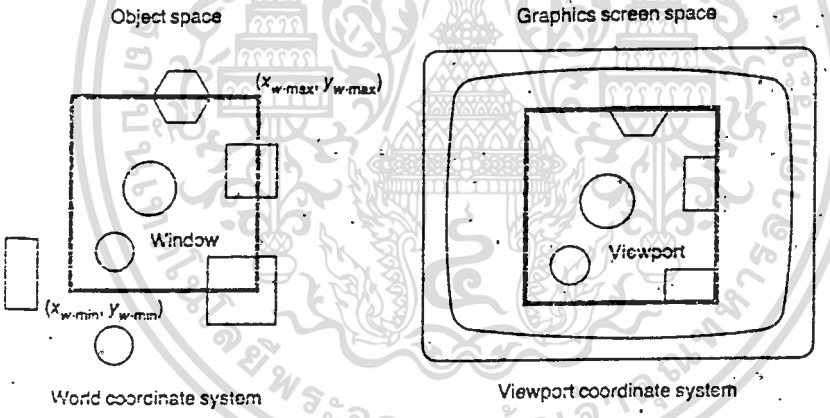


FIGURE 3.8 Window vs viewport

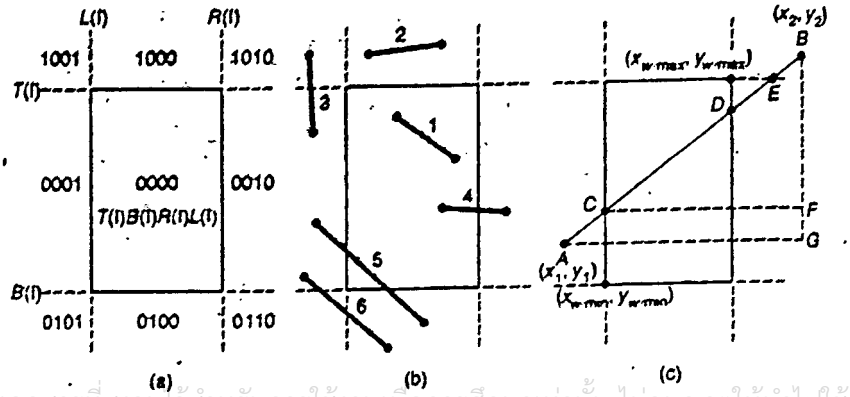


FIGURE 3.9 Clipping line segments

โดยการพิจารณารูป 3.3 รูป 3.3 แสดงให้เห็นว่าจุดP หมุนทวนเข็มนาฬิกาไปเป็นมุม ϕ ทำให้เกิดตำแหน่งใหม่ที่จุด P' (หมายเหตุว่า ระยะทางจากจุดกำเนิด (0, 0) ไปยังP และP' จะมีค่าเท่ากันทั้งนี้เนื่องจากเป็นการหมุนรอบจุด Origin และระยะทางเหล่านี้จะถูกกำหนดให้มีค่า =r จากรูปเราจะได้ค่า X' และ Y' มีค่าดังนี้

$$X = r\cos\phi$$

$$Y = r\sin\phi$$

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

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

จากผลลัพธ์ที่ได้นี้จะเหมือนในสมการ 3.9

3.2 Transfomation of lines and objects

เส้นตรงเส้นหนึ่งจะได้มาจากตำแหน่ง Vector 2 ตำแหน่งที่กำหนดด้วยจุดปลายของ coordinate ทั้งสองและการ Transform เส้นตรงสามารถกระทำได้โดยการ Transform จุดปลายทั้งสองและก็วาดเส้นตรงขึ้นใหม่ระหว่างจุดปลายที่ Transform นี้ ซึ่งกฎเกณฑ์นี้จะกระทำได้โดยวิธีการ Transform ทั้งหมดที่ได้กล่าวมาแล้ว ตั้งแต่ scalung จนถึง Rotation

ถ้าเราต้องการสร้างภาพวัตถุที่ทำขึ้นจากเส้นจำนวนมาก และภาพของวัตถุจะทำได้โดยการ Transform จุดทั้งหมดที่มีอยู่ในภาพวัตถุนั้นและก็เชื่อมโยงเส้นระหว่างจุดให้ถูกต้อง โดยใช้สมการ 3.1 กับจุดแต่ละจุดของวัตถุนั้น ถ้าวัตถุนั้นประกอบด้วย n จุดรูปแบบการ Transform ก็จะถูกแสดงได้ดังนี้

Transform object coordinate		original object coordinate	Transformation Matric	
$\begin{bmatrix} P'_1 \\ P'_2 \\ P'_3 \\ P'_n \end{bmatrix}$	$= \begin{bmatrix} X'_1, Y'_1 \\ X'_2, Y'_2 \\ X'_3, Y'_3 \\ X'_n, Y'_n \end{bmatrix}$	$= \begin{bmatrix} X_1, Y_1 \\ X_2, Y_2 \\ X_3, Y_3 \\ X_n, Y_n \end{bmatrix}$	$X \begin{bmatrix} A & B \\ C & D \end{bmatrix}$	(3.11)

3.2.1 Scaling

ตามตัวอย่าง พิจารณารูป ในรูป 3.4 (a) ที่ถูกกำหนดโดยจุดทั้ง 3 คือ (0, 0), (30, 0) และ (15, 30) ถ้าต้องการที่จะขยายสามเหลี่ยมนี้ให้ใหญ่ขึ้นเป็น 2 เท่าของขนาดเดิม เราก็จะทำได้โดยการเลือก $A = D = 2$ เพื่อให้เห็นผลของการ Scale เราจะคำนวณหาค่า P_1', P_2', P_3' ได้ดังนี้

$$\begin{bmatrix} P_1' \\ P_2' \\ P_3' \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 30 & 0 \\ 15 & 30 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 60 & 0 \\ 30 & 60 \end{bmatrix}$$

สังเกตุได้ว่าการขยายให้ใหญ่ขึ้นจะเกี่ยวข้องกับจุดกำเนิดของระบบ coordinate X,Y เสมอ (ดูในรูป 3.4(b))

3.2.2 Rotation

ในการหมุนวัตถุ n จุดในทิศทางทวนเข็มนาฬิการอบจุด Origin ของระบบ coordinate เราจะทำได้ดังนี้

$$\begin{bmatrix} P_1' \\ P_2' \\ \vdots \\ P_n' \end{bmatrix} = \begin{bmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ \vdots & \vdots \\ X_n & Y_n \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \qquad (3.12)$$

รูปสามเหลี่ยมในรูป 3.4 B เมื่อหมุนไปเป็นมุม 30 องศา (0.5236 Radian) ในทิศทางทวนเข็มนาฬิการอบจุดกำเนิดซึ่งจะเขียนแทนด้วย Matrix ดังนี้

$$\begin{bmatrix} P_1' \\ P_2' \\ P_3' \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 60 & 0 \\ 30 & 60 \end{bmatrix} \begin{bmatrix} 0.866 & 0.5 \\ -0.5 & 0.866 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 52 & 30 \\ -4 & 67 \end{bmatrix}$$

สามเหลี่ยมที่หมุนเสร็จแล้วจะเป็นดังรูป (3.4)

3.2.3 Translation

วัตถุสามารถย้ายตำแหน่งได้ (เลื่อนไปบนระนาบ) โดยการเพิ่มขนาดเข้าไปยังแต่ละจุดของวัตถุและรูปแบบของ Matrix ของวัตถุที่ต้องการ

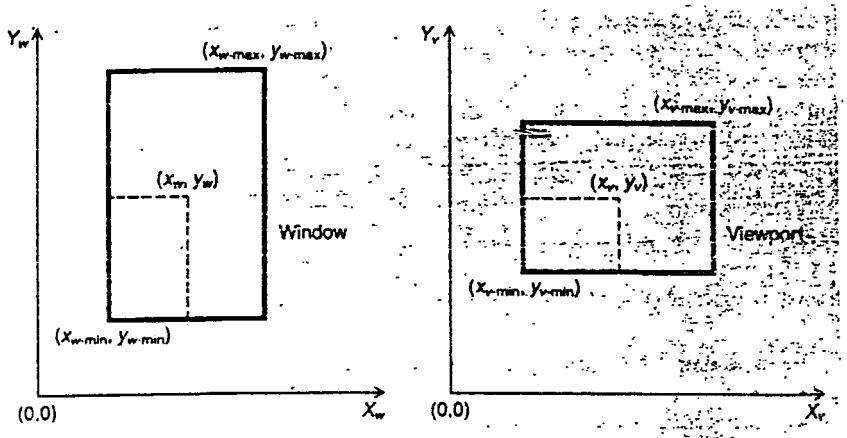


FIGURE 3.10 Window to viewport mapping

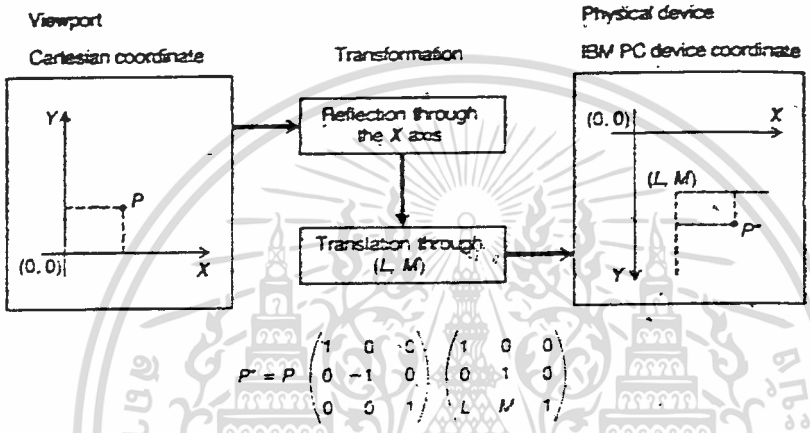
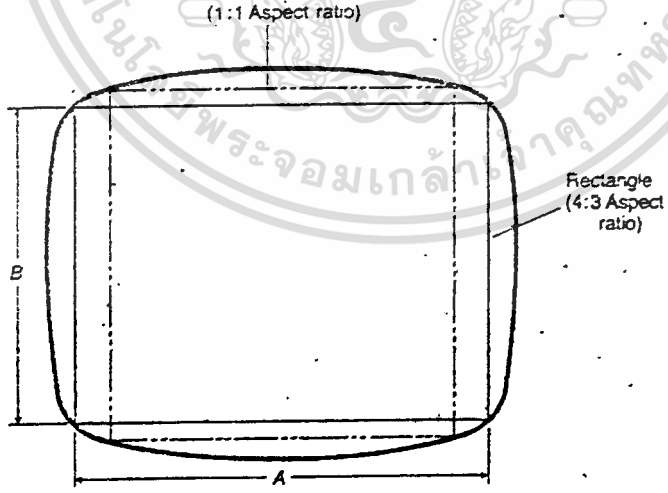


FIGURE 3.11 Transformation process from the viewport coordinate system to the IBM PC device coordinate system



Diagonal screen size	A (mm)	B (mm)	Square (mm)	Typical monitor
9"	148	111	116	Portable IBM PC
12"	216	162	172	
13"	240	180	191	IBM PC
16"	283	213	226	
19"	348	261	276	TV
20"	356	267	283	
21"	363	276	293	
23"	408	306	324	TV

FIGURE 3.12 Raster dimensions as functions of screen size

translate ก็มีรูปแบบดังนี้

Translate Coordinate Original Coordinate Translate Magnetude

$$\begin{bmatrix} P_1' \\ P_2' \\ \\ \\ P_n' \end{bmatrix} = \begin{bmatrix} X_1 , Y_1 \\ X_2 , Y_2 \\ \\ \\ X_n , Y_n \end{bmatrix} + \begin{bmatrix} TX , TY \\ TX , TY \\ \\ \\ TX , TY \end{bmatrix} \quad (3.14)$$

เช่นตัวอย่าง ให้พิจารณาสามเหลี่ยมในรูป 3.4 (c)ถ้าเราต้องการจะย้ายตำแหน่งไปทางขวา 100 หน่วย และขึ้นบน 50 หน่วย ดังนั้น YX= 100, TY = 50 เราก็สามารถคำนวณหาตำแหน่งจุดใหม่ได้ดังนี้

$$\begin{bmatrix} P_1' \\ P_2' \\ P_3' \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 52 & 30 \\ -4 & 67 \end{bmatrix} + \begin{bmatrix} 100 & 50 \\ 100 & 50 \\ 100 & 50 \end{bmatrix} = \begin{bmatrix} 100 & 50 \\ 152 & 80 \\ 96 & 117 \end{bmatrix}$$

ผลที่ได้จากการ Transform จะเป็นดังรูป 3.4

3.2.4 Limitation of Standard coordinate system

มีข้อจำกัดอยู่ 2 แบบในการใช้ Transformation Matrix ขนาด 2 x 2 คือ

ข้อจำกัดแรก คือ การ Transformation ทั้งหมดนี้เราจะต้องนำมาเริ่มต้นที่จุด origin เพราะว่าการ Transform แต่ละแบบถูกสร้างขึ้นมาใช้กับจุด origin เท่านั้น เพราะฉะนั้น ถ้าเราต้องการที่จะหมุนรอบจุดหนึ่งจุดใด เราก็ไม่สามารถใช้ Transformation Matrix เหล่านี้ได้ ในทำนองเดียวกับการ Reflect ก็จะใช้ได้เฉพาะตามแกน X และแกน Y เท่านั้น รวมทั้งจุด origin ด้วยแต่ไม่สามารถทำได้บนจุดหนึ่งจุดใดหรือแกนหนึ่งแกนใดที่นอกเหนือไปจากจุดและแกนดังกล่าว

ข้อจำกัดที่ 2 ถ้าเราต้องการกระทำเกี่ยวกับการ Translate ซึ่งโดยรูปแบบแล้วการtranslateจะแตกต่างไปจากการ Scale และการ Rotate (คือ การ Translate จะเป็นการบวก ส่วนการ Rotate และการ Scale จะเป็นการคูณ)

ใน computer graphic การ Transform เหล่านี้ไม่สามารถที่จะนำมาใช้แสดงด้วย Matrix ที่กล่าวมาแล้ว แต่จะต้องใช้ Matric ที่รวมกันของ Matric เบื้องต้นเหล่านี้และลำดับการทำงานของ Matric จะต้องมีการกำหนดขึ้นอย่างแน่นอน เพราะถ้ารูปแบบตำแหน่งของ Matric เปลี่ยนไปก็จะทำให้ได้ผลลัพธ์ที่แตกต่างกันไป และเพื่อให้เกิดความสะดวกทางคณิตศาสตร์เราก็จะใช้ Matric เหล่านี้ในรูปแบบของ Homogenous เพื่อว่าการ Transform แบบพื้นฐานเหล่านี้ทั้งหมดที่กล่าวมาแล้วสามารถที่จะนำมารวมกันได้โดยการคูณ โดยวิธีของ Homogenous coordinate system จะช่วยให้เราสามารถที่จะกำจัดข้อจำกัดของ Matric เหล่านี้ลงได้

3.3 Homogenous Coordinate System

ในระบบ Homogenous จุด (X,Y) จะถูกแทนด้วย (X,Y,H) โดยค่า Scale factor H จะต้องไม่เป็น 0 ต่อจากนั้นเราต้องมีการ Normalize ระบบ Homogenous ของจุด (X,Y) ให้เป็น (X/H,Y/H, 1) เนื่องจาก H เป็นค่าคงที่ใดที่ไม่ใช่ 0 โดยปกติแล้วเรามักจะให้ H มีค่า =1 เพราะฉะนั้นการหารจึงไม่จำเป็นต้องใช้ ผลลัพธ์ที่ได้คือ จุดในระบบ Homogenous จะมีค่า (X,Y,1) ซึ่งมีค่าเท่ากับระบบ Coordinate ที่จุด (X,Y) เราจะใช้รูปแบบของ Homogenous นี้ในส่วนที่จะกล่าวต่อไป

3.3.1 Revised Transformation Matrix

ผลที่เกิดขึ้นจากการใช้ระบบ Homogenous นั้นคือ เราต้องมีการขยาย Transform Matrix ของเดิมที่มีขนาด 2x 2 เพราะว่าจุดในระบบ Homogenous นี้จะแทนด้วยแถว Vector ที่มี 3 ส่วน ฉะนั้น Transformation Matrix ขนาด 2 x2 จะขยายไปเป็น Matrix ขนาด 3 x3 ดังนี้

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \longrightarrow \begin{bmatrix} A & B & 0 \\ C & D & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3.15)$$

ถ้าต้องการเพิ่มคุณสมบัติการ Translate เข้าไปใน Matrix ขนาด

3 x3 เราก็ทำได้โดยการเปลี่ยนแถวที่ 0 จาก (0, 0, 1) ไปเป็น (L,M,1) ซึ่งผลที่ได้จะเป็นการ Translate แบบเชิงเส้นของจุด P ใด ๆ ไปทางแนวนอน L หน่วยและไปทางแนวตั้ง M หน่วย เพราะฉะนั้น Matrix ที่ได้จะอยู่ในรูป form ดังนี้

$$T = \begin{bmatrix} A & B & 0 \\ C & D & 0 \\ L & M & 1 \end{bmatrix} \qquad (3.16)$$

Vector คอลัมภ์ที่ 3 ใน Transformation Matrix ขนาด 3x3 จะเป็นตัวช่วยในการรวมรูปแบบการกระทำแบบ Translate การ scale และการ Rotate ให้อยู่ในรูปแบบการคูณได้ผลของการ Transform จุด (X,Y) เขียนได้ดังนี้

$$(X' \; Y' \; 1) = (X, Y, 1)\, T$$

ซึ่งจะได้ค่า $X' = AX + CY + L$

$Y' = BX + DY + M$

ในส่วนที่จะกล่าวต่อไป เราจะแสดงให้เห็นว่ารูปแบบ Matrix ใหม่ที่มีการดัดแปลงมาจากการรวม Matrix พื้นฐานเข้าด้วยกันทำได้อย่างไร

3.3.2 Translation

เมื่อเราพิจารณาที่การ Translate, Matrix ที่ได้ให้ไว้ในสมการ 3.6 ก็สามารถดัดแปลงได้ดังนี้

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ L & M & 1 \end{bmatrix} \qquad (3.17)$$

เราจะแสดงให้เห็นวิธีในการ Translate ให้เราพิจารณาสามเหลี่ยมรูปหนึ่งซึ่งจุดทั้ง 3 อยู่ที่ตำแหน่ง (15,30), (16,30), (30,60) และเราต้องการ Translate สามเหลี่ยมรูปนี้ไปเป็นระยะทาง (20,50) ดังนั้นการสร้าง Vector ของตำแหน่งใหม่จะเป็นดังนี้

$$P' = \begin{bmatrix} 15 & 30 & 1 \\ 60 & 30 & 1 \\ 30 & 60 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 20 & 50 & 1 \end{bmatrix} = \begin{bmatrix} 35 & 80 & 1 \\ 80 & 80 & 1 \\ 50 & 110 & 1 \end{bmatrix}$$

และอะไรจะเกิดขึ้นถ้าเราเคลื่อนย้ายสามเหลี่ยมนี้ไปอีก 30 หน่วย ทางแกน X และ 10 หน่วยตามแกน Y เพราะฉะนั้น เราจะได้การ Translate ครั้งที่ 2 เป็นดังนี้

$$P'' = P' \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 30 & 10 & 1 \end{bmatrix} = P \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 20+30 & 50+10 & 1 \end{bmatrix}$$

จะเห็นได้ว่าผลลัพธ์ของการ Translate 2 ครั้ง เป็นการบวกแต่ลำดับขั้นตอนที่จะได้ผลผลัพธ์ก็คือ การคูณโดยทั่วไปแล้ว เราจะแสดงรูปแบบได้ดังนี้

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ L_1 & M_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ L_2 & M_2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ L_1+L_2 & M_1+M_2 & 1 \end{bmatrix} \qquad (3.18)$$

3.3.3 Scaling

Scaling Matrix ในสมการ 3.5 จะแสดงด้วย ขนาด 3 x3 ดังนี้คือ

$$\begin{bmatrix} A & 0 & 0 \\ 0 & D & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3.19)$$

ในขณะที่ผลลัพธ์ของการ Translate ในแต่ละลำดับขั้นจะนำมาบวกกัน แต่ผลลัพธ์ในการ Scale ในแต่ละลำดับขั้นจะนำมาคูณกัน

frist Scaling Second Scaling Conbine Scaling

$$\begin{bmatrix} A_1 & 0 & 0 \\ 0 & D_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_2 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} A_1A_2 & 0 & 0 \\ 0 & D_1D_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.20)$$

3.3.4 Rotation

การ Rotate เป็นมุม θ ในระบบ Homogenous จะมีรูปแบบดังนี้

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.21)$$

เราจะรวมการ Rotate 2 ครั้ง ที่ติดต่อกันโดยวิธีการบวกดังนี้คือ

frist Rotation Second Rotation Conbine Rotation

$$\begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & \sin\theta_2 & 0 \\ -\sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_1+\theta_2) & \sin(\theta_1+\theta_2) & 0 \\ -\sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.22)$$

3.3.5 Inverse of Transfomation

พิจารณาที่ Matrix ทั้ง 2 นี้

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

Matrix ทั้ง 2 นี้จะให้ผลลัพธ์ที่ตรงข้ามกัน นั่นคือ ถ้ามีการคูณด้วย Matrix T_1 การ Translate จะไปอยู่ยังตำแหน่ง (-L,-M) ในขณะที่คูณด้วย Matrix T_2 การ Translate จะไปอยู่ยังตำแหน่ง (L,M) ถ้าเราเอา Matrix ทั้ง 2 มาคูณติดต่อกัน โดยการคูณ T_1 ด้วย T_2 หรือคูณ T_2 ด้วย T_1 จะทำให้เกิด Matrix เอกลักษณ์ขึ้นซึ่งทำให้จุดเดิมไม่มีการเปลี่ยนแปลงซึ่งผลคูณ Matrix ทั้งสองนี้จะมีค่าดังนี้คือ

$$T_1T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -L & -M & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ L & M & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3.23)$$

ถ้ามี Matrix คู่ใดก็ตามซึ่งทำให้เกิดค่าการคูณของ Matrix ทั้ง 2 เป็น Matrix เอกลักษณ์แล้ว เราจะเรียก Matrix ทั้ง 2 ว่าเป็น Inverse ซึ่งกันและกัน

โดยทั่วไปแล้ว Inverse Matrix จะเขียนแทนด้วย T^{-1} โดยเหตุนี้เอง $T_2 = T_1^{-1}$ Transformation Matrix ที่กล่าวมาแล้วในบทก่อนจะมีค่า Inverse Matrix เป็นของตัวมันเอง เช่นตัวอย่าง Matrix ของการหมุนทวนเข็มนาฬิกาเข็มมุม 30 องศา

$$R(30) = \begin{bmatrix} \cos 30^\circ & \sin 30^\circ & 0 \\ -\sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Matrix ของ Matrix ดังกล่าวนี้ก็คือ การหมุน 30 องศา ในทิศทางตามเข็มนาฬิกา

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

และเมื่อเราคูณ R(30)เข้ากับ R(-30) เราจะได้ Matrix เอกลักษณ์

3.3.6 Transformation a Coordinate System

ถึงจุดนี้ เราได้เรียนรู้การ Transform กลุ่มของจุดกลุ่มหนึ่งไปเป็นกลุ่มของจุดในอีกรูปแบบหนึ่งที่มีระบบ Coordinate เดียวกัน และยังมีอีกวิธีที่ต่างกันแต่ให้ผลลัพธ์คล้ายกันก็คือ วิธีการ Transform กลุ่มของจุด โดยการเปลี่ยนแปลงระบบ Coordinate เพื่อที่จะแสดงให้เห็นวิธี การนี้ทำได้อย่างไร เราจะตรวจสอบด้วยกรณีพิเศษ 2 กรณีคือ

เราจะเริ่มต้นโดยพิจารณาในรูป 3.5(a) จุด P' ได้มาจากการหมุนจุด P ในทิศทางทวนเข็มนาฬิกาเป็นมุม 30 องศา ในทำนองเดียวกันการหมุนระบบ coordinate XY ไปในทิศทางตามเข็มนาฬิกา 30 องศา ก็จะทำให้เกิดระบบ coordinate X'Y' ตามที่เห็นในรูป 3.5 (b)

ระบบ coordinate ใหม่นี้จะทำให้จุด P นี้มีค่า coordinate ที่เปลี่ยนไปและเมื่อมีการเปรียบเทียบจุด P' ที่ระบบ coordinate XY และจุด P ในระบบ coordinate X'Y' จะเห็นว่ามีผลลัพธ์ที่ได้เหมือนกัน Matrix ที่จะนำมาใช้ในแทนการ Transform ของระบบ coordinate หนึ่งก็คือ Inverse Matrix ที่ใช้ในการTransform จุดในระบบ coordinate อีกระบบหนึ่งซึ่งความสัมพันธ์ดังกล่าวนี้จะมีประโยชน์อย่างมากในวิธีการสร้าง Computer graphic

กรณีที่ 2 ให้พิจารณารูป 3.5 (c) เมื่อจุด P อยู่ที่ตำแหน่ง (3,3) (โดยกำหนดให้อยู่ในระบบ coordinate XY) และจะมีตำแหน่ง coordinate 13.5, 2.5) ในระบบ coordinate X'Y' การ Transform ที่ใช้ในการเปลี่ยนระบบ coordinate จากXY ไปเป็น X'Y' สามารถทำได้จาก การTranslate จุด origin ไปยังตำแหน่ง O' ต่อมาก็ต้อง scale แกนทั้ง 2 และสิ้นสุดด้วยการหมุนระบบ Coordinate X'Y' ไปตามเข็มนาฬิกาเป็นมุม θ เพราะฉะนั้นผลรวมของ Matrix ที่ได้จะเป็นดังนี้

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2\cos\theta & -2\sin\theta & 0 \\ 2\sin\theta & 2\cos\theta & 0 \\ -4\cos\theta-2\sin\theta & 4\sin\theta-2\cos\theta & 1 \end{bmatrix} \qquad (3.24)$$

เพราะฉะนั้น เราจะได้ค่าP' มีค่าดังนี้

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

ถ้าเรารู้ค่ามุม θ เราก็สามารถหาจุด P' ได้ ในตัวอย่างของเรา จุด P' (3.5,2.5) รู้ค่าแล้วในที่นี้เราจะ solve หาค่า θ โดยการให้

$\sin\theta = U$ และ $\cos\theta = (1 - U^2)^{\frac{1}{2}}$ (หมายเหตุ $\sin^2\theta + \cos^2\theta = 1$)
จากความสัมพันธ์นี้จะได้ว่า

$$2\cos\theta + 4\sin\theta = 3.5$$
$$-2\sin\theta + 4\cos\theta = 2.5$$

แทนค่าสมการในเทอมของ U และรวมเทอมต่าง ๆ ให้เป็นกลุ่มจะทำให้ได้สมการ

$$18.5U^2 + 10U + 8.25 = 0$$

แก้สมการหาค่า U จะได้ค่าประมาณ 0.45614 หรือได้ค่ามุม θ 26.85 องศา การ Transform จากระบบ coordinate X'Y' ไปเป็นระบบ coordinate XY จะถูกกระทำโดยการใช้ Inverse Matrix ในสมการ 3.24 โดยทั่ว ๆ ไปแล้ว ถ้าเรากำหนดให้ Transformation Matrix คิดค่าผลคูณของ Matrix n Matrix

$$T = T_1 \cdot T_2 \cdot \ldots \cdot T_{n-1} \cdot T_n$$

และเราจะหาค่า T^{-1} ได้จากความสัมพันธ์คือ

$$T^{-1} = T_n^{-1} T_{n-1}^{-1} \ldots\ldots T_2^{-1} \cdot T_1^{-1} \qquad (3.25)$$

เพราะฉะนั้นจากจุด P' ถ้าเราต้องการหาจุด P เดิมเราก็จะทำได้โดย

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

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

เพราะฉะนั้นจุด $P = P' \cdot T^{-1}$

$$= (1.75\cos\theta - 1.25\sin\theta + 2, 1.75\sin\theta + 1.25\cos\theta + 1, 1)$$
$$= (3, 3, 1)$$

วิธีการ Transform ระบบ coordinate นี้มีประโยชน์มากเมื่อมีวัตถุหลายชิ้นและวัตถุแต่ละชิ้นก็กำหนดโดยระบบ coordinate ของตัวมันเอง

และเราจะทำการรวม เพื่อให้ระบบ coordinate เหล่านี้ให้มีรูปแบบของ coordinate เดียวกันเพียง coordinate เดียว โดยเป็น coordinate แบบ global

3.4 Sequential 2-D Transformation

Homogenous ที่กล่าวมาแล้วในบทก่อน ๆ จะทำให้เราสามารถรวมการ Transform แบบพื้นฐานเข้าด้วยกัน เช่น Translate, Rotate, Scale เพื่อทำให้เกิด Transformation Matrix ตามที่ต้องการ ในเนื้อหาส่วนนี้จะแสดงให้เห็นถึงการรวมกันของ Matrix แต่ละอย่างที่จะถูกนำมาใช้แทนการ Rotate ของวัตถุหนึ่งรอบจุด ๆ หนึ่ง และการ Reflect วัตถุรอบแกนใดแกนหนึ่ง

3.4.1 2D Rotation about an arbritory Point

ในส่วนก่อนนั้น เราได้พิจารณาการหมุนที่เกิดขึ้นรอบจุด Origin แล้ว ในเวลานี้เราจะพิจารณาการหมุนวัตถุรอบ ๆ จุด $O\ (q_1, q_2)$ เพราะว่าเรารู้เพียงแต่การหมุนรอบจุดกำเนิดเท่านั้น เราจะแบ่งปัญหาของจุด Origin ไปเป็น 3 ปัญหาที่แตกต่างกัน ซึ่งทั้ง 3 ปัญหานี้ก็ต้องการลำดับของการ Transform 3 กลุ่มคือ

กลุ่มที่ 1 ย้ายจุดศูนย์กลางการหมุนไปยังจุด Origin

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -q_1 & -q_2 & 1 \end{bmatrix}$$

กลุ่มที่ 2 หมุนไปเป็นมุมที่ต้องการ

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

กลุ่มที่ 3 **Translate** ผลลัพธ์ที่ได้กลับไปยังตำแหน่งจุดศูนย์กลางของ Q เดิม

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

ผลการหมุนรอบจุดใด ๆ ก็คือ การรวม Transformation Matrix ทั้ง Matrix ข้างบนนี้ ซึ่งก็คือผลคูณของ Matrix ทั้ง 3

$$T_1 = T_1 \, R(\theta) \, T_2$$

$$= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ q_1(1-\cos\theta)+q_2\sin\theta & q_2(1-\cos\theta)-q_1\sin\theta & 1 \end{bmatrix} \quad (3.26)$$

เช่นตัวอย่าง เราจะพิจารณาสามเหลี่ยมที่กำหนดด้วยจุดเหล่านี้คือ (10,20), (20,20) และ (15,30) และจะสมมุติว่าให้มีการหมุนรอบจุด Q (5,25) ในทิศทางทวนเข็มนาฬิกา ถ้าทำการ Transform ทีละ step ซึ่งแสดงให้เห็นดังรูป 3.6 และผลลัพธ์ที่ได้สุดท้ายของจุดทั้ง 3 ของสามเหลี่ยมจะมีค่าเป็น (11.83, 23.17) (20.49, 28.17) และ (11.16, 34.33)

3.4.2 Reflection About an Srbitrary Axis

สมมุติว่าเราต้องการ Reflect สามเหลี่ยมหนึ่งรอบแกน $y = a + bx$ เราก็จะต้องสร้าง Transformation Matrix ที่เราต้องการได้จากการรวม Transformation Matrix แบบพื้นฐานเข้าด้วยกัน

ขั้นตอนทั้งหมดในการ Reflect สามเหลี่ยมรอบแกน $y = a + bx$ เป็นดามนี้

ขั้นตอนที่ 1 ย้ายตำแหน่งจุด origin ไปยังตำแหน่ง (0,a) (ดูในรูป 3.7 b)

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

ขั้นตอนที่ 2 หมุนแกนดังกล่าวในทิศทางตามเข็มนาฬิกา เป็นมุม θ ในที่นี้คือ แกน $y = a + bx$ และแกนดังกล่าวนี้จะวางอยู่บนแกน x ส่วนค่ามุม θ จะหาได้จากslop ของแกน $y = a + bx$ ในที่นี้คือ b , $\cos\theta = 1/r$, $\sin\theta = b/r$ เมื่อ $r = (b^2+1)^2$ เราจะแทน Parameter เหล่านี้ในสมการ (3.21) ทำให้ได้ว่า

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

สภาพภายหลังการหมุนเป็นมุม θ เป็นดังรูป 3.7(c)

ขั้นตอนที่ 3 Reflect สมมเหลี่ยมรอบแกน x ดูในรูป 3.7(d)

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

ขั้นตอนที่ 4 ใช้ Inverse Matrix ในขั้นตอนที่ 2 ดูในรูป 3.7(e)

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

ขั้นตอนที่ 5 ใช้ Inverse Matrix ในขั้นตอนที่ 1 ดูในรูป 3.7

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

วัตถุประสงค์ของขั้นตอนที่ 4 และขั้นตอนที่ 5 ก็คือ การส่งวัตถุกลับไปที่ระบบ coordinate เดิม ระบบ coordinate ที่เกิดขึ้นในขั้นตอนที่ 1, 2, 3 จะถูกนำมาใช้เป็นขั้นตอนในการคำเนินกรรมวิธีเท่านั้น การ Transform ที่เสร็จสิ้นสมบูรณ์นั้นจะประกอบด้วย Transformation Matrix 5 ขั้นตอน ดังกล่าวคือ

$$T_1 = T_1 R(-\theta)\, T_2 (R(\theta)\, T_1^{-1} \qquad (3.29)$$

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

ลำดับขั้นตอนทั้งหมดเหล่านี้แสดงให้เห็นดังรูป (3.7) และส่วนที่จะกล่าวต่อไปคือ การสร้าง Computer Program ที่จะกระทำการเกี่ยวกับ Viewing Transform และวาดภาพที่ได้ลงบนจอภาพ

3.5 View Port Planning of 2D graphic

เมื่อมาถึงขวงนี้เราได้รู้วิธีการที่จะเปลี่ยนแปลงวัตถุให้อยู่ในรูปแบบต่าง ๆ โดยการนำ Transformation Matrix พื้นฐานมารวมกันใน abstract space และจะเรียก abstract space อีกชื่อหนึ่งว่า world coordinate ในส่วนนี้เราจะให้รายละเอียดของวิธีการ Transform ระบบ world coordinate ให้เป็นรูปแบบของcoordinate ของจอภาพที่นำมาใช้กับ coordinate ของ screen ได้ในการทำดังกล่าวนี้เราต้องใช้วิธีการ Viewing 3 อย่างด้วยกันคือ Window dipping, window to view port Mapping, สร้าง coordinate ให้เหมาะสมกับลักษณะของจอภาพ

3.5.1 Windoe Clipping

Window and View port

ขอบเขตของสี่เหลี่ยมผืนผ้าของเนื้อที่สิ่งที่น่าสนใจในระบบ world coordinate เราจะเรียกว่า window และลักษณะการนำเอา window

ไป Transform เป็นภาพใน Screen display เราจะเรียกว่า View-port หรือให้ความหมายของ window ในรูปแบบอื่นคือส่วนที่เรากำหนดพื้นที่ที่เราต้องการแสดงผลและเราจะใช้ View port ในการกำหนดตำแหน่งบน Screen เพื่อใส่ window รูป 3.8 แสดงให้เห็นความสัมพันธ์ของ window กับ Viewport

ตามที่เห็นมาแล้วในรูป 3.8 ปกติแล้ว window จะถูกเลือกโดยผู้ใช้และจะไม่รวมกับข้อมูลของรูปภาพทั้งหมดที่กำหนดในฐานข้อมูล ถ้าโปรแกรมไม่มีการตรวจเช็ค Visibility ก่อนที่จะส่งค่าข้อมูลออกทางจอภาพแล้วค่า coordinate ทั้งหมดอาจจะอยู่ด้านนอกของจอภาพ โดยเหตุนี้เองจึงต้องมีการ clip เกิดขึ้น ก่อนที่จะมีการส่งกลุ่มที่นำออกแสดงผลไปให้จอภาพ เพื่อที่จะกำจัดส่วนที่มองไม่เห็นของภาพที่อยู่ด้านนอกขอบเขตสิ้นสุด window

Clipping Point

การ clip จุดหนึ่งจุดใดสามารถทำได้โดยง่าย โดยการเปรียบเทียบ coordinate ของจุดนั้นกับขอบเขตสิ้นสุด window ถ้าจุดหนึ่งจุดใดอยู่นอกขอบเขตสิ้นสุด window จุดนั้นจะไม่ถูก Plot บนจอภาพ (ดูในรูป 3.8) ถ้าขอบเขตสิ้นสุด X coordinate อยู่ที่ Xwmin และ Xwmax และขอบเขตสิ้นสุด Y coordinate อยู่ที่ตำแหน่ง Ywmin, Ywmax เงื่อนไขที่จะพบได้เมื่อจุด (X,Y) อยู่ในสภาวะมองเห็นคือ

$$Xwmin \leq X \leq Xwmax$$

$$Ywmin \leq Y \leq Ywmax$$

Clipping lines

การ clip เส้นมีความยุ่งยากมากกว่าจุด เพราะว่าวิธีการนี้ต้องมีการตรวจเช็คและแบ่งชนิดเส้นตรงที่ต้องการตรวจสอบออกเป็นส่วน ๆ คือ ส่วนที่มองไม่เห็นหมด ส่วนที่มองเห็นหมด ส่วนที่มองไม่เห็นเป็นบางส่วนและเห็นเป็นบางส่วน สำหรับตัวอย่างส่วนของเส้นตรงจะอยู่ภายใน window ถ้าไม่มีจุดหนึ่งจุดใดของเส้นตรงเส้นนี้อยู่ภายนอกขอบเขตสิ้นสุด window

Alporithm ที่ตามมานี้เป็นพื้นฐานของ the cohen sutherled Clipping

เพื่อที่จะแสดงให้เห็นลำดับขั้นตอนของ Alporithm เราจะพิจารณารูป 3.9 (a) เป็น window ที่ถูกแบบออกเป็น 9 ส่วนโดยเส้นตรง 4 เส้น และเส้นตรง 4 เส้นที่ตัดกันนี้จะกำหนดขอบเขตสิ้นสุด window แต่ละจุดของเส้นตรงจะถูกกำหนดโดย computer Bit code ที่จะแสดงว่าจุดดังกล่าวอยู่ในสภาวะเช่นไรอยู่บน window, อยู่ล่าง window, อยู่ซ้าย window , อยู่บน window และเราจะให้ตัวแปรทั้ง 4 ตัวสำหรับจุดแต่ละจุดดังนี้

T(I) = จุด I อยู่บนขอบเขตสิ้นสุด window

B(I) = จุด I อยู่ล่างขอบเขตสิ้นสุด window

R(I) = จุด I อยู่ด้านขวาของเขอบเขตสิ้นสุด window

L(I) = จุด I อยู่ด้านซ้ายขอบเขตสิ้นสุด window

แต่ละตัวแปร Bir code นี้จะถูกset ค่าเป็น 1 ถ้าค่าความสัมพันธ์ระหว่างจุดนั้นกับ window มีค่าเป็นจริงแต่ถ้าค่าความสัมพันธ์เป็นเท็จแล้วตัวแปร Bit code จะมีค่าเป็น 0 สภาวะ Viribility ของเส้นตรงจะถูกกำหนดโดยการเปรียบเทียบค่า Variable code ของจุดปลายทั้งสองคือ

- ถ้าทั้ง 2 จุด (1 และ 2) วางอยู่ภายใน window เส้นตรงตลอดทั้งเส้นก็จะมองเห็น (ดูส่วนของเส้นตรงที่ 1 ในรูป 3.9) และเงื่อนไขนี้จะเกิดขึ้นได้ เมื่อ ตัวแปร Bit code ของจุดทั้งสองมีค่าตามสมการ

$$T(1)+B(1)+R(1)+L(1)+T(2)+B(2)+L(2) = 0 \quad (3.31)$$

- ถ้าจุดทั้ง 2 อยู่ภายนอกขอบเขตสิ้นสุด window เดียวกันเส้นตรงนั้นจะมองไม่เห็น (ดูเส้นตรงที่ 2 และ 3 ในรูป 3.9) เงื่อนไขนี้จะตรวจเช็คโดยการตรวจสอบผลบวกของผลคูณแต่ละตัวแปร Bit code ที่กำหนดโดยที่ผลบวกของผลคูณนี้จะต้องมีค่าไม่ใช่ 0

$$T(1)T(2)+B(1)B(2)+R(1)R(2)+L(1)L(2) \neq 0 \quad (3.32)$$

- ถ้าจุดทั้งสองอยู่ภายนอก windowและไม่เข้ากฎในกรณีที่กล่าวมาแล้วทั้ง 2 อย่าง สภาวะของเส้นดังกล่าวไม่สามารถกำหนดขึ้นในทันที และเงื่อนไข

ที่จะเป็นไปได้ของสภาวะเช่นนี้คือ

$$T(1)T(2)+B(1)B(2)+R(1)R(2)+L(1)L(2) = 0 \quad (3.33)$$

ในกรณีนี้ก็มีความเป็นไปได้ 2 รูปแบบคือ 1 ส่วนของเส้นตรงดังกล่าวมองเห็นเป็นบางส่วนหรือส่วนของเส้นตรงมองไม่เห็นทั้งหมด (ดูเส้นที่ 5 และเส้นที่ 6 ในรูป 3.9 (b) ถ้ามีด้านหนึ่งด้านใดอยู่ภายใน window และอีกด้านหนึ่งอยู่ภายนอก window เส้นตรงดังกล่าวจะอยู่ในสภาวะที่มองเห็นเป็นบางส่วนและไม่เห็นเป็นบางส่วน (ดูเส้นตรงที่ 4 ในรูป 3.9 (b)

เมื่อไรก็ตามที่เราพบเงื่อนไขในสมการ 3.33 เราจะต้องแบ่งส่วนของเส้นตรงโดยการหาจุดตัดของเส้นนั้นกับขอบเขตสิ้นสุด window ด้านใดด้านหนึ่งและตัดด้านที่อยู่ภายนอกขอบเขตสิ้นสุด window ออกไป สำหรับตัวอย่างเช่น เส้นตรง AB (ในรูป 3.9 C) จะถูกแบ่งที่จุด C และตัดส่วน AC ทิ้งไป เพราะว่าแต่ละเส้นถูกตัดโดยขอบเขตสิ้นสุด window อาจจะเป็นขอบเขตสิ้นสุดทางด้านแนวตั้งหรือขอบเขตสิ้นสุดทางด้านแนวนอน การหาจุดตัด coordinate (X,Y) ที่ตัดกับขอบเขตสิ้นสุด window ดังกล่าวก็จะหาได้โดยง่าย ตัวอย่างในการคำนวณหาจุดตัดของ coordinate ที่จุด C (Xw min,Y), เราพิจารณาในรูป 3.9(C) ABG และ CBF เป็น มุมฉากและสามเหลี่ยมทั้งสองเป็นสามเหลี่ยมคล้ายดังนั้นเราจะได้ว่า

$$\frac{BG}{AG} = \frac{BF}{CF} = \frac{Y_2-Y_1}{X_2-X_1} = \frac{Y_2-Y}{X_2-X_{wmin}} \quad (3.34)$$

จัดเป็นกลุ่มให้เรียบร้อยจะได้ว่า

$$Y = Y_2 - \frac{Y_2 - Y_1}{X_2 - X_1}(X_2 - X_{emin}) \quad (3.35)$$

ในขณะนี้รู้ค่าสภาวะจุดปลายของ A และ C แล้วเพราะว่าจุดปลายทั้งสองวางอยู่นอกขอบเขตสิ้นสุดเดียวกับส่วนของเส้นตรง AC จะมองไม่เห็นและต้องตัดทิ้งไปต่อไปเราต้องตรวจสอบเส้นตรง CB ว่าอยู่ในสภาวะเช่นไรในที่นี้อยู่ในสภาวะ 3.33 ดังนั้นเราจะคำนวณหาจุดตัดได้ที่จุด D และตัดเส้น DB ทิ้งไป

ในที่สุดเราต้องตรวจสอบเส้น CD และผลการตรวจสอบ CDอยู่ในสภาวะที่มองเห็นจึงนำส่วนของ CDไปผ่านกรรมวิธีอื่นอีก (line clipping Subroutine)

ไม่มีความจำเป็นที่ต้องใช้ในเครื่อง IBM PC เพราะว่าในเครื่องได้เตรียมให้ไว้แล้ว)

3.5.2 Window to Viewport Mapping

ภายหลังการ Clip ภาพที่ตัดกับขอบเขตสิ้นสุด window แล้ว ขั้นต่อไปก็ต้องทำการ Map ภาพดังกล่าวให้เป็น Viewport การเปลี่ยนแปลงจุดการมองอาจกระทำได้โดยวิธีการ Window to Viewport Transformation

เพื่อให้เข้าใจวิธีการนี้ให้พิจารณารูป (3.10) ซึ่งในที่นี้ต้องการเปลี่ยนจุด (Xw,Yw) ให้เป็นจุด (Xv,Yv) โดยวิธีใดวิธีหนึ่งเพื่อที่จะให้ได้ค่าความสัมพันธ์ภายในสี่เหลี่ยมกับตัวแปรต่าง ๆ ที่ได้ให้ไว้ในรูป 3.10 การ Transform window coordinate ไปเป็น Viewport coordinate ก็มีลักษณะเหมือนกับการย้ายจุด (Xw,Yw) ไปยัง (Xw.min,Yw.min) ซึ่งทำได้โดยการ Scale ที่เหมาะสมและต่อมาก็มีการ Translate จากระบบ world coordinate ไปยังระบบ coordinate ใหม่ (Viewport) เพราะฉะนั้นต้องใช้ขั้นตอนของการ Transform แบบพื้นฐาน 3 แบบดังนี้คือ

$$(Xv,Yv,1)=(Xw,Yw,1)\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ -Xwmin & -Ywmin & 1\end{bmatrix}\begin{bmatrix}S_1 & 0 & 0\\ 0 & S_2 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ Xvmin & Yvmin & 1\end{bmatrix}$$

$$= S_1(Xw-Xw.min)+Xvmin, S_2(Yw-Ywmin)+Yvmin, 1 \qquad (3.36)$$

เพื่อที่จะทำให้ได้สัดส่วนที่เหมาะสม (ตำแหน่งที่สัมพันธ์กับภาพในรูปสี่เหลี่ยม) โดยใช้ Scale factor S_1 และ S_2 ที่มีค่า set ไว้ดังนี้

$$S_1 = \frac{Xvmax - Xvmin}{Xwmax - Xwmin} \qquad (3.37)$$

และ

$$S_2 = \frac{Yvmax - Yvmin}{Ywmax - Ywmon} \qquad (3.38)$$

(สังเกตุว่า $S_1 \neq S_2$ จะทำให้รูปทรงผิดแบบไป)

หลังจากได้ค่า Viewport Program เรียบร้อยแล้ว เราก็สามารถคำนวณหา Viewport coordinate ได้ดังนี้

$$Xv = S_1 (Xw - Xwmin) + Xvmin \qquad (3.39)$$

$$Yv = S_2 (Yw - Ywmin) + Yvmin \qquad (3.40)$$

และจะเห็นได้ว่า window coordinate และ view coordinate จะมีค่าเท่ากันเมื่อ $S_1 = S_2 = 1$

3.5.3 Generating Physical Device Coordinate

จอภาพ graphic ที่นำมาใช้ในระบบคอมพิวเตอร์ทั่ว ๆ ไป มีระบบ coordinate(pixel)แตกต่างไปจาก coordinate ทางคณิตศาสตร์ ซึ่งความแตกต่างดังกล่าวนี้จะแสดงให้เห็นในรูป (3.11) เพื่อให้ความหมาย Screen Coordinate เหมือนกับในเทอมของระบบ Cartesian Coordinate ทั่ว ๆ ไป เราต้องมีการย้ายตำแหน่งของ Pixel โดยวิธีการ Transform เพราะว่าจอภาพของ Micro Computer เช่น IBM PC มีค่าตามแกน Y เพิ่มขึ้นเมื่อนับลงมาด้านล่าง ดังนั้นจุด (Xv,Yv) บนระบบ Coordinate แบบ Viewport สามารถกระทำได้โดยการ Reflect จุด (Xv,Yv) ตามแกน X และจุด (0,0) ของจอภาพถูกกำหนดไว้ในตำแหน่งทางด้านบนซ้ายมือของจอภาพต่อไปเราต้อง Translate จุด origin นี้ไปยังตำแหน่งบนจอภาพที่ห่างจากตำแหน่งเดิมเป็นระยะ (L,M) เพราะฉะนั้นค่า Device coordinate จะหาค่าได้ดังนี้

$$(X_s, Y_s, 1) = (Xv, Yv, 1) \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ L & M & 1 \end{bmatrix}$$

$$(Xs, Ys, 1) = (Xv+L, M-Yv, 1) \qquad (3.41)$$

หรือ

$$Xs = S_1 (Xw - Xwmin) + Xvmin + L \qquad (3.42)$$

$$Ys = S_2 (Ywmin - Yw) - Yvmin + M$$

เราจะใช้สมการสุดท้าย (3.42) ในโปรแกรมทั้งหมดที่อยู่ในหนังสือเล่มนี้ ส่วนค่า Scaling Parameter S_1 และ S_2ในสมการ 3.42 จะต้องออกแบบให้

computer screen เพื่อให้มีอัตราส่วนความกว้างของจอและความสูงของจอ 1:1 อย่างไรก็ตามความแตกต่างระหว่าง scale เหล่านี้ยังคงมีอยู่ในคอมพิวเตอร์ทั่วไป สำหรับตัวอย่าง เช่น IBM PCใน Mode Medium Resolution จะมีค่า coordinate 320 x 200 ซึ่งมีความหมายว่าทาง Horizontal Unit จะมีค่า = $\frac{200}{320}$ ของทาง Vertical Unit ในคอมพิวเตอร์ graphic มันมีความสำคัญมากที่ขนาด scale ทาง Vertical และ Horizontal จะต้องมีระยะห่างเท่ากัน เพื่อให้สี่เหลี่ยมจตุรัสที่ปรากฏบนจอจะมีด้านทั้งหมดมีความยาวเท่ากันและอีกอย่างก็คือจุดทุกจุดบนวงกลมจะมีระยะห่างจากจุดศูนย์กลางเท่ากัน ซึ่งจะทำได้โดยการปรับค่า Xs Coordinate ในสมการ (3.42)

$$Xs = SCF\ (S_1\ (Xw-Xwmin) + Xvmin + L\ ) \qquad (3.43)$$

เมื่อ SCF = Screen Scaling adjustment factor

เนื่องจากจอภาพแต่ละแบบก็จะมีลักษณะของ screen แตกต่างกันไป ดังนั้นค่า SCF จำเป็นต้องกำหนดตามลักษณะทางกายภาพของจอภาพนั้น รูป 3.12 แสดงให้เห็นเนื้อที่แสดงผลที่จะได้รับมากที่สุดบนย่านหนึ่ง ๆ ของขนาด Monitor ต่าง ๆ สำหรับตัวอย่างเช่นตัวอักษรที่แสดงลักษณะของ Amdek color II monitorมีความยาวทาง Horizontal 240 mm และความยาวทาง Vertical 180 mm เพราะฉะนั้นวิธีการที่จะกำหนดค่า SCF ของเครื่องดังกล่าวนี้ได้ดังนี้คือ ขั้นตอนที่ 1 (ความละเอียดของจอภาพ 320 x 200) คำนวณหาอัตราส่วน H กับ V

$$H = \frac{\text{ความยาวทางแนวนอน}}{\text{จำนวนจุดสูงสุดทางแนวนอน}} = \frac{240}{320}$$

$$V = \frac{\text{ความยาวทางแนวตั้ง}}{\text{จำนวนจุดสูงสุดทางแนวตั้ง}} = \frac{180}{200}$$

ขั้นตอนที่ 2 คำนวณหาค่า SCF ได้ดังนี้

$$SCF = \frac{V}{H} = 1.2$$

และเราจะคูณ Horizontal Coordinate ด้วยค่า SCF ในการออกแบบ Viewport แต่ละครั้ง ใน Mode High Resolution จุดทางแนวนอนจะมีขนาดเป็น 2 เท่า (640) ดังนั้นค่า Scale factor จึงมีค่า = 2.4

3.5.4 Zooming

หลังจากที่ได้ภาพจาก DATA Base แล้วในบางครั้งเราอาจมีความจำเป็นที่ต้องการตรวจสอบส่วนของรูปภาพนั้นให้ละเอียดยิ่งขึ้น เช่นตัวอย่างเราอาจต้องขยายบางส่วนของภาพนั้นให้มีขนาดเต็มจอภาพในการทำดังกล่าวนี้จะทำได้โดยการกำหนดสี่เหลี่ยมบนจอภาพ ที่จะกำหนดส่วนเล็กของภาพปัจจุบันที่นำมาแสดงเป็นรูปภาพขยายต่อไป (ดูรูป 3.13) หมายเหตุว่ารูปสี่เหลี่ยมที่ขยายนี้ไม่ใช่ window ของตัวมันเอง แต่จะเป็น window ที่ได้จาก Transform ส่วนของภาพนั้นลงบนจอภาพ เพราะว่าเราเพียงต้องการรายละเอียดของภาพในพื้นที่สี่เหลี่ยม window ใหม่จะถูกกำหนดภายใน window เก่า แต่การ Zoom จะเสียเวลามากในการหาค่า New window Parameter ใหม่ในระบบ World Coordinate (X'wmin,Y'wmin) และ (X'wmax, Y'wmax) ซึ่งค่าเหล่านี้จะมีค่าตรงกับ coordinate ทะแยง (X_{a1} ,Y_{a1}) และ (X_{a2} ,Y_{a2}) ของพื้นที่สี่เหลี่ยมบนจอภาพ ต่อมาเราต้องตั้งค่า Viewport parameter (Xvmin, Yvmin) ตามขนาดของ Screen ที่ต้องการ Window clipping routine จะถูกนำมาใช้กับขอบเขตสิ้นสุด window ใหม่ด้วย เมื่อเราแทนค่า window ใหม่และค่า Viewport Parameter ลงในสมการ 3.42 เราก็จะได้รับ screen coordinate ใหม่

วิธีการ Transform เพื่อให้ได้รับจุด (X'w min,Y'w min)และ (X'w max, Y'w max) ของscreen coordinate จะถูกทำได้โดยใช้ Inverse ของสมการ 3.41 และ 3.36 ตามลำดับ ต่อมาเราก็แทนผลลัพธ์ของ Inverse Transform ที่ติดต่อกันได้ตามนี้

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ L & M & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xvmin & -Yvmin & 1 \end{bmatrix} \begin{bmatrix} 1/s_1 & 0 & 0 \\ 0 & 1/s_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

เมื่อให้ค่า Screen Coordinate (Xa_1, Y_{a1})และจะต้องหา $(X'wmin, Y'wmin)$ เราจะคำนวณหาได้ดังนี้

$$(X'wmin, Y'wmin, 1) = (x_{a1}, Y_{a1}, 1) \cdot T \qquad (3.44)$$

วิธีที่มีประสิทธิภาพมากในการเก็บค่าข้อมูลภาพทั้งหมดเมื่อรูปภาพรูปแรกได้ถูกสร้างขึ้น โดยการกำหนดสี่เหลี่ยมบนจอภาพให้เป็นส่วนของภาพที่ต้องถูกขยายให้ใหญ่ขึ้น แล้วก็ทำการอ่านข้อมูลรูปภาพภายในสี่เหลี่ยมและเขียนจุดเหล่านี้ขึ้นใหม่บนจอภาพในการเก็บและเคลื่อนที่ส่วนของ Computer's Memory จะต้องใช้อุปกรณ์พิเศษและต้องใช้โปรแกรมภาษาเครื่องเสมอ (ผลของการ Zoom จะทำได้โดยการใช้คำสั่ง window หรือ view ที่มีอยู่ในเครื่อง IBM PC กลับไปดูในส่วนที่ 2.3.7 และ 2.3.8)

3.6 Screen display for 2-D graphic on IBM PC

ขณะนี้เราได้วิธีการทำภาพในรูปแบบต่าง ๆ แล้ว แต่เรายังไม่กล่าวถึงวิธีการสร้าง Computer Program ที่จะทำหน้าที่ในการเกิดภาพเหล่านี้ได้อย่างถูกต้อง จุดประสงค์ของส่วนนี้คือ การเขียนโปรแกรมที่มีพื้นความรู้ที่อธิบายในส่วนก่อน ๆ

3.6.1 Program Struture

. Inputting world Coordinate รูป 3.14 แสดงให้เห็น 2D viewing และกรรมวิธีในการ Plot

ในขั้นตอนแรกที่ต้องเขียน Program Computer คือ เราจะต้องเตรียมข้อมูลให้เรียบร้อย และมีรูปแบบ 3 อย่างที่ใช้ในการเตรียมข้อมูลในการสร้างภาพ 2- D

<u>ลำดับแรก</u> ข้อมูลอาจจะป้อนเข้าทาง Keyboard โดยผ่านทางคำสั่ง DATA ภายในโปรแกรม

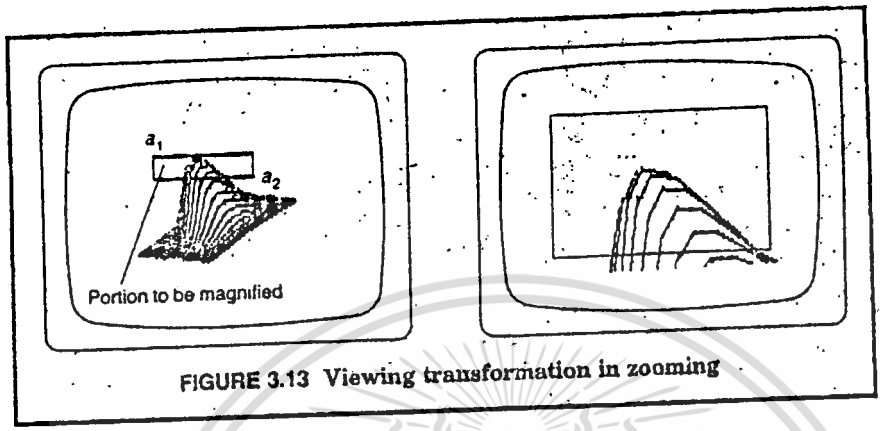


FIGURE 3.13 Viewing transformation in zooming

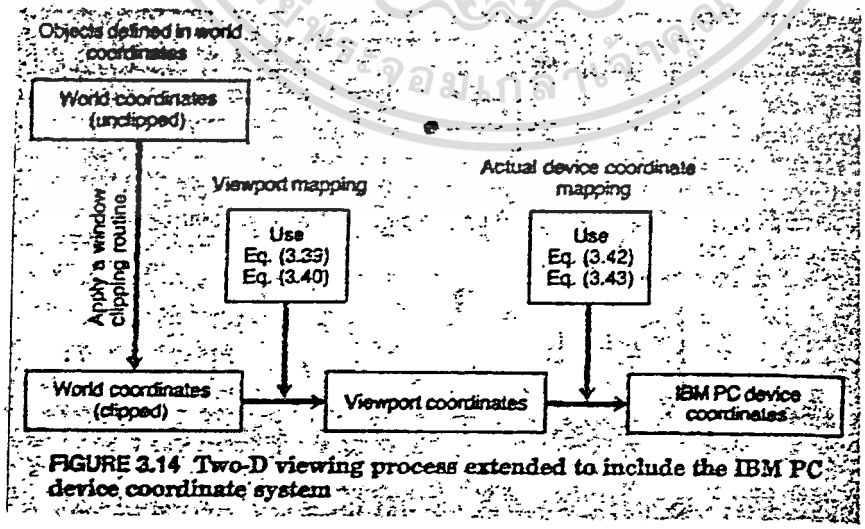


FIGURE 3.14 Two-D viewing process extended to include the IBM PC device coordinate system

ลำดับที่สอง ตั้งกลุ่มข้อมูล file อาจจะสร้างขึ้นตามรูปแบบอินพุท เพื่อให้ผู้ใช้เชื่อมโยงข้อมูล file เข้ากับ 2 -D graphic Program

ลำดับสาม กำเนิดโดยตั้งไว้ใน Program ของมันเองอาจจะประกอบด้วยคำสั่ง line ที่จะทำการกำเนิดจุด Coordinate โดยผ่านความสัมพันธ์ทางคณิตศาสตร์ เช่น Y= f(X) หรือ X = f(t) รูปแบบหลังนี้ถูกนำมาใช้บ่อยมากเมื่อต้องการจะ Plot รูปภาพที่มีพื้นฐานของ funtion ทางคณิตศาสตร์

2 - D Transformation Matrix generator

ภายหลังการให้ค่า Input DATA Coordinate ได้สร้างเสร็จเรียบร้อยแล้ว ขั้นตอนต่อไปก็คือ การเชื่อมโยงของผู้ใช้กับรูปแบบของการ Transform ที่ต้องการใช้ ตามที่เห็นมาแล้วใน section 3.4 รูปแบบของการ Transform ที่รวมกับหลาย ๆ อย่างนี้ ทำขึ้นเพื่อให้วัตถุอยู่ในตำแหน่งและรูปร่างที่เหมาะสม การรวม Matrix พื้นฐานของ Transform หลายอย่างนี้เป็นเรื่องที่น่าเบื่อถ้าต้องทำด้วยมือ แต่ในโปรแกรม 3.1 ได้จัดการเกี่ยวกับหน้าที่นี้แล้วโดยการออกแบบลักษณะของการ Transform ต่าง ๆ ต่อมาก็ให้ computer ทำการคูณ Matrix เข้าด้วยกันซึ่งกล่าวถึงสมการ 3.6 ซึ่งจะเห็นว่า Transformation Matrix ประกอบด้วย Matrix 9 Matrix เพื่อใช้ในการควบคุมรูปร่างของวัตถุ 2 มิติ แต่เนื่องจากตำแหน่งในคอลัมภ์ Vector ที่สามใน Matrix ขนาด 3 x3 ถูก fix ไว้โดยตลอดและจะมี Parameter เหลือเพียง 6 ตัวที่จะต้องกำหนดค่าให้คือ (A, B,C,D,L,M) โดยความจำเป็นแล้วโปรแกรม 3 จะคำนวณหาค่าจากส่วน 6 ส่วนของ Matrix. เท่านั้น แล้วถ้ารู้ค่าส่วน 6 ส่วนนี้ของ Matrix เราก็สามารถคำนวณหาค่าการ Transform ได้

Clipping

ตามที่กล่าวไว้แล้วในส่วน 3.5.1 เราจะต้องกำหนด Window Parameter เพื่อกำหนดว่าส่วนไหนของภาพที่เราต้องการแสดงผล รายละเอียดของรูปภาพไม่ว่าจะเป็นจุดหรือเส้น เราต้องใช้ window Clipping

```
100 ' Program 3.1: Subroutine to Determine the Transformation Matrix
110     CLS: KEY OFF : SCREEN 2
120 '------ Type of Transformation ---------
130 LOCATE 2,20: PRINT"TRANSFORMATION MENU":PRINT
140 PRINT TAB(20)"1. Translation"
150 PRINT TAB(20)"2. Scaling"
160 PRINT TAB(20)"3. Reflection"
170 PRINT TAB(20)"4. Rotation"
180 PRINT TAB(20)"5. Combined Transformations"
190 PRINT: PRINT TAB(25)"< Input Choice >":INPUT C1
200 ON C1 GOSUB 250,280,320,450,520
210 DEF FNX(X,Y)=A*X + C*Y + L              ' FNX(X,Y) : Transformed X coordinate
220 DEF FNY(X,Y)=B*X + D*Y + M              ' FNY(X,Y) : Transformed Y coordinate
230 RETURN
240 END
250 '-------- TRANSLATION -------------
260 PRINT: PRINT TAB(20)"< Translation thru (L,M) > ":INPUT L,M
270 A=1:B=0:C=0:D=1  : RETURN
280 '-------- SCALING     -------------
290 PRINT:PRINT TAB(20)"< Scaling Factors for X,Y >":INPUT A,D
300 PRINT TAB(20)"< Shear Factors for  B,C >":INPUT B,C
310 L=0:M=0          : RETURN
320 '-------- REFLECTION  -------------
330 PRINT:PRINT TAB(20)"REFLECTION MENU"
340 PRINT TAB(20)"1. Reflection thru X-axis"
350 PRINT TAB(20)"2. Reflection thru Y-axis"
360 PRINT TAB(20)"3. Reflection thru Origin"
370 PRINT:PRINT TAB(20)"< Input Choice >":INPUT C2
380 IF C2=1 THEN 430
390 IF C2=2 THEN 420
400 A=-1:D=-1: GOTO 430
410 A=1: D=-1: GOTO 430
420 A=-1: D=1
430 B=0: C=0: L=0: M=0
440 RETURN
450 '-------- ROTATION   ---------------
460 PRINT: PRINT TAB(20)" Rotation Angle(degree) : + for counterclockwise,"
470         PRINT TAB(20)"                        - for clockwise.":
480 INPUT DEGREE
490 QS=DEGREE*3.14159/180
500 A=COS(QS):B=SIN(QS):C=-B:D=A:L=0:M=0
510 RETURN
520 '------ COMBINED TRANSFORMATIONS ----------
530 PRINT:PRINT TAB(25)"Number of Transformations Required:"
540 INPUT N1
550 FOR I=1 TO N1
560       PRINT:PRINT TAB(20) I;": Type of Transformation -  1,2,3,or 4.";
570       INPUT C4
580       ON C4 GOSUB 250,280,320,450
590       IF I=1 THEN 630 ELSE 600
600       A1=A:B1=B:C1=C:D1=D:L1=L M1=M
610       A=A1*A2 + B1*C2
620       B=A1*B2 + B1*D2
630       C=C1*A2 + D1*C2
640       D=C1*B2 + D1*D2
650       L=L1*A2 + M1*C2 + L2
660       M=L1*B2 + M1*D2 + M2
670       IF I= N1  THEN 700
680       A1=A B1=B: C1=C:D1=D:L1=L M1=M
690 NEXT I
700 RETURN
```

routine ให้เหมาะสมกับจุดหรือเส้นเหล่านั้น routine ส่วนนี้ยังนำมาใช้ในการตรวจสอบว่าจะยอมรับหรือคัดทิ้งจุดหรือเส้นในแต่ละ Transform ในแบบ World Coordinate

Viewport Planning and Plot

และเมื่อมีการยอมรับจุดและเส้นทั้งหมดแล้ว เราก็จะนำส่วนนั้น Plot ลงบนจอภาพ โดยที่เราต้องวางแผนเกี่ยวกับ Viewport โดยการกำเนิดค่า (Xvmin, Yvmin) และ (Xvmax, Yvmax) ซึ่งจะเป็นการกำหนดเนื้อที่ของภาพบนจอภาพและค่า Parameter เหล่านี้จะถูกนำมาเป็น I/P ของ Program รวมทั้งค่า SCF ด้วย (Screen scale Adjustment factor) ต่อมาเมื่อค่า Viewing Parameter ถูกกำหนดขึ้นแล้วเราก็สามารถคำนวณหา Screen Coordinate จากสมการ 3.4.3 ต่อไปเราก็จะทำการ Plot ค่า coordinate เหล่านี้ลงบนจอภาพ

3.6.2 Illustrative Example

Rotating a triangle

เป็นตัวอย่างแรกของการใช้ Transformation Matrix พร้อมกับนำออกแสดงผลเราจะให้โปรแกรมนี้ทำการหมุนรูปสามเหลี่ยมรอบ ๆ จุดใดจุดหนึ่ง พิจารณารูปสามเหลี่ยมที่จุดกำเนิดในระบบ World Coordinate ที่มีการกำหนดจุดและเส้นตามนี้

Line	Points (X,Y)
$\overline{P_1P_2}$	$P_1(10, 20)$
$\overline{P_2P_3}$	$P_2(20, 20)$
$\overline{P_3P_1}$	$P_3(15, 30)$

และเราต้องการหมุนรูปสามเหลี่ยมนี้รอบจุด Q(5,25) (ตัวอย่างนี้เคยนำมากล่าวแล้วในรูป 3.6) ตามที่ได้กล่าวมาแล้วในส่วน 3.4.1 เราต้องใช้ขั้นตอน Transform 3 อย่างในการหมุนสามเหลี่ยมรอบจุดใดจุดหนึ่ง

.Program 3.2 จะ Plot แต่ละการ Transform สามเหลี่ยม และผลลัพธ์การหมุนแสดงในรูป 3.15

Nested Octagon

ตัวอย่างที่สองคือ การใช้ Transformation Matrix ในการ Plotรูปแปดเหลี่ยมให้ต่อเนื่องกันไปตลอดใน viewport ที่แตกต่างกันไปในการสร้างภาพดังรูป 3.6 ประกอบด้วยขั้นตอนหลายขั้นตอนดังนี้

ขั้นตอนที่ 1 ลำดับแรกเราต้องกำหนดจุดศูนย์กลางอยู่ที่จุด origin และเราจะ fix ตำแหน่งหนึ่งของจุดทั้ง 8 จุดของรูปแปดเหลี่ยม จากรูปแบบทางเรขาคณิต เรารู้ว่าจุดที่ติดต่อกันไปของรูปแปดเหลี่ยมนั้นจะอยู่ห่างจากกัน 45 องศา เพราะฉะนั้นจุดที่ 2 ก็จะหาได้จากการหมุนจุดแรกไปเป็นมุม 45 องศา ในตัวอย่างของเรา เราให้จุดแรกตั้งอยู่ในตำแหน่ง (30,0) จากสมการ (3.21) เราจะได้จุดที่สองคือ (21.21, 21.21) โดยเราสมมุติให้หมุนในทิศทางทวนเข็มนาฬิกาเป็นมุม 45 องศา ต่อมาจุดที่ 3 ก็หาได้จากการคำนวณจุดที่ 2 ที่หมุนไปเป็นมุมอีก 45 องศา ในวิธีการเดียวกันนี้ เราก็จะได้จุดทั้งหมดดังรูป 3.16 (a)

ขั้นตอนที่ 2

ในการทำรูปแบบนี้ให้ง่ายขึ้นโดยไม่ต้องใช้ window clipping routine จะทำได้โดยการ set ค่า window Parameter ที่ตำแหน่งจุด (−40,−40)และ (40,40) และก็จะวางรูปแปดเหลี่ยมดังกล่าวนี้ภายใน window นี้ และการย้ายตำแหน่งของข้อมูล window (รูปแปดเหลี่ยม) ไปยัง viewport เราจะ set ค่า Viewport Parameter ที่ (40,40) และ (200,200) ดูในส่วน (3.5.3) ต่อมาเราก็ใช้สมการ (3.4.2) เพื่อคำนวณหา Screen Coordinate โดยลำดับแรกต้องคำนวณหาจุด และ โดยการใช้สมการ 3.37 และ 3.38

$$S_1 = \frac{200-40}{40-(-40)} = 2$$

$$S_2 = \frac{200-40}{40-(-40)} = 2$$

สำหรับจุดที่ 1 ของรูปแปดเหลี่ยมคือ (30,0) จะหาค่า Viewport Coordinate มีค่าดังนี้

$$Xv = 2\ 30 - (-40) + 40 = 180$$

$$Yv = 2\ 0 - (-40) + 40 = 120$$

และ PC device coordinate คือ

$$Xs = Xv + L = 180 + 0 = 180$$

$$Ys = -Yv + M = -120 + 199 = 79$$

ต่อไปเราก็ทำการ Scale Coordinate Xs โดยการใช้สมการ (3.43)

$$Xs = SCF.Xs = 2.4Xs = 432$$

ขั้นตอนที่ 3 ภายหลังที่คำนวณหาค่า และ Mapping Viewport coordinate ไปเป็น Screen Coordinate เรียบร้อยแล้ว เราจะทำการเชื่อมโยงเส้นระหว่างจุด โดยการใช้คำสั่ง Line เพราะฉะนั้นเราจะได้รูปแปดเหลี่ยมที่สมบูรณ์บนจอภาพ

ขั้นตอนที่ 4

ในตอนนี้เราจะเพิ่มโปรแกรม line ในการกำเนิดภาพแปดเหลี่ยมที่ต่อเนื่องกันไปภายในรูปแปดเหลี่ยมรูปแรก (วาดไว้แล้วใน step ที่ 3) โดยที่รูปแปดเหลี่ยมแต่ละรูปจะเลื่อนไปทีละน้อยของแปดเหลี่ยมรูปก่อนและจะเคลื่อนไปตามทิศทางทวนเข็มนาฬิกา มีการลดขนาดรูปแปดเหลี่ยมจะได้รับจากการคูณโดยสมการ (3.19) และการหมุนรูปแปดเหลี่ยมแต่ละรูปที่ละน้อยเราก็ใช้สมการ (3.21) เพราะฉะนั้นการเปลี่ยนแปลงจากรูปแปดเหลี่ยมจากรูปหนึ่งไปเป็นรูปแปดเหลี่ยมรูปต่อไป (เปลี่ยนขนาดและหมุน) ถูกกระทำได้ดังนี้คือ

$$(X',Y',1) = (X,Y,1) \begin{bmatrix} A & 0 & 0 \\ 0 & D & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

เมื่อ A = D ดังนั้นเราจะเขียนโปรแกรมหาค่า X' และ Y' ได้ดังนี้ คือ

$$X' = X.A.\cos(\theta) - Y.A.\sin(\theta)$$

$$Y' = X.A.\sin(\theta) - Y.A.\cos(\theta)$$

ในตัวอย่างนี้ เราจะสมมุติ A = 0.95 และ O = 5° ต่อไปเราจะใช้ Program 3.3 วาดรูปแปดเหลี่ยมที่ต่อเนื่องกันไป

ตามที่แสดงไว้แล้วในส่วน 2.3.8 และ 2.3.9 คำสั่ง window และ view สามารถนำมาใช้ในการmap ระบบ World Coordinate ให้อยู่ในรูปแบบ PC'S Screen Coordinate โปรแกรม 3.4 จะทำการวาดรูปแปดเหลี่ยมที่ต่อเนื่องกันไป โดยการใช้คำสั่ง window และ View Statement ผลลัพธ์ที่ได้อยู่ในรูป 3.16

```
10 ' Program 3.2 : Object Rotation through an Arbitrary Point
15 CLS: KEY OFF: SCREEN 2
20 '----- Input Screen Parameters --------
25 INPUT " Enter center coordinates (CX,CY) for plot =";CX,CY
30 INPUT " Enter Screen Scaling Factor (SCF) ="; SCF
35 GOSUB 100 : CLS
40 ' ---- Input World Coordinates --------
45 FLAG = 0
50 FOR I=1 TO 4
55 READ X,Y
60  XS = SCF*FNX(X,Y) + CX
65  YS = CY - FNY(X,Y)
70 IF FLAG > 0 THEN 80
75 LINE (XS,YS)-(XS,YS): FLAG = 1 : GOTO 85     'Line clipping is provided
80 LINE -(XS,YS)                                'through the LINE statements.
85 NEXT I
90 END
95 DATA 10,20,20,20,15,30,10,20
96 ' ---- attach Program 3.1 -----
```

PROGRAM 3.2 Object rotation through an arbitrary point

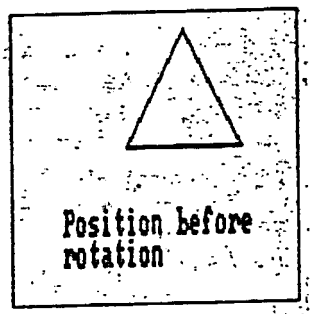
```
100 ' Program 3.3: nested Octagons
110 CLS: KEY OFF: SCREEN 2
120 PI = 3.14159
130 XWMIN=-40: YWMIN=-40                        'Set window
140 XVMIN=40 : YVMIN = 40                       'Set viewport
150 S1 =2    : S2=2   : SCF = 2.4               'Se' scale factors
160 L=0 : M=190                                 'Translation parameters
170 XW=30: YW=0
180 C=COS(PI/4): S=SIN(PI/4)
190 CN=COS(PI/36): SN=SIN(PI/36) : SF = .95
200 FOR J= 1 TO 20                              'Draw 20 nested octagons
210     IF J=1 THEN 240
220     XW = SF*XW*CN - SF*YW*SN
230     YW = SF*XW*SN + SF*YW*CN
240     FOR I=0 TO 8                            'Draw single octagon
250         XS=S1*(XW-XWMIN) + XVMIN + L
260         XS=SCF*XS
270         YS=S2*(YWMIN-YW) - YVMIN + M
280         IF I=0 THEN PSET (XS,YS)
290         LINE -(XS,YS)
300         XN=XW*C -YW*S: YW=XW*S +YW*C : XW=XN
310     NEXT I
320 NEXT J
```

PROGRAM 3.3 Nested octagons

```
100 ' Program 3.4: An alternate method to draw nested octagons
110 CLS: KEY OFF: SCREEN 2
120 PI = 3.14159: SCF=2.4
130 WINDOW (-75,-75) - (75,75)
140 VIEW (150,0 )-(340,190)
150 XW=30: YW=0
160 C=COS(PI/4): S=SIN(PI/4)
170 CN=COS(PI/36): SN=SIN(PI/36) : SF = .95
180 FOR J= 1 TO 20
190     IF J=1 THEN 220
200     XW = SF*XW*CN - SF*YW*SN
210     YW = SF*XW*SN + SF*YW*CN
220     FOR I=0 TO 8
230         XS=XW
240         XS=SCF*XS
250         YS=YW
260         IF I=0 THEN PSET (XS,YS)
270         LINE -(XS,YS)
280         XN=XW*C -YW*S: YW=XW*S +YW*C : XW=XN
290     NEXT I
300 NEXT J
310 END
```

PROGRAM 3.4 An alternate method to draw nested octagons

```
Enter center coordinates (CX,CY) for plot =? 320,100
Enter Screen Scaling Factor (SCF) =? 2.4

TRANSFORMATION MENU

1. Translation
2. Scaling
3. Reflection
4. Rotation
5. Combined Transformations

     < Input Choice >? 5

Number of Transformations Required? 3

 1 : Type of Transformation - < 1,2,3,or 4>? 1

< Translation thru (L,M) > ? -5,-25

 2 : Type of Transformation - < 1,2,3,or 4>? 4

  Rotation Angle(Degree) : + for counterclockwise,
                           - for clockwise.? 30

 3 : Type of Transformation - < 1,2,3,or 4>? 1

< Translation thru (L,M) > ? 5,25
```



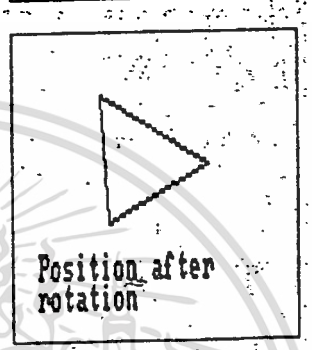


FIGURE 3.15 Rotating a triangle through an arbitrary point

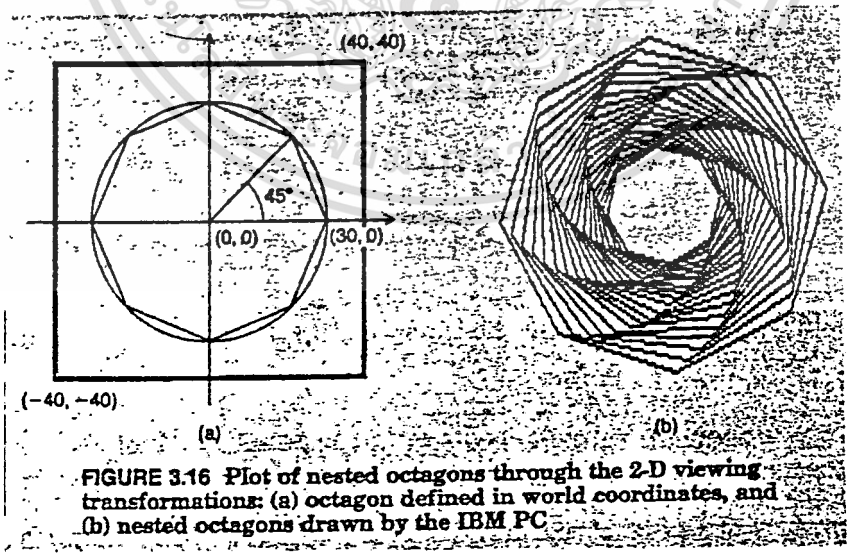


FIGURE 3.16 Plot of nested octagons through the 2-D viewing transformations: (a) octagon defined in world coordinates, and (b) nested octagons drawn by the IBM PC

บทที่ 4

Mathematrical Elements in 3-D graphic

ในบทนี้จะแสดงให้เห็นถึงวิธีการแสดงภาพ 3 มิติ เช่น การวาดโครงร่าง หรือรูปภาพที่มีสีระดับสีต่าง ๆ กัน วิธีการกำเนิดภาพ 3 มิติ จะมีส่วนเพิ่มเติมที่ยุ่งยากขึ้นมาอีกในการแสดงออกจอภาพเนื่องจากจอภาพคอมพิวเตอร์เป็นขนาด 2 มิติ ดังนั้นเราจำเป็นต้องมีวิธีการบางอย่างเพื่อที่จะแสดงรูปร่างวัตถุ 3 มิติ ลงในจอภาพคอมพิวเตอร์ขนาด 2 มิติ ซึ่งวิธีการนี้จะถูกกระทำด้วยกรรมวิธี Perspective Projection ซึ่งจะทำการเปลี่ยนแปลงวัตถุ 3 มิติลงบนระนาบที่มีขนาด 2 มิติ ในบทนี้ยังจะกล่าวถึงพื้นฐานเบื้องต้นของการ Projection และ Transformation ในรูปแบบ 3 มิติและก็อธิบายให้เห็นกฎเกณฑ์ทางคณิตศาสตร์มาเกี่ยวข้องกับวิธีการดังกล่าวเหล่านี้ได้อย่างไร

4.1 Coordinate System

ในการที่จะแสดงจุดในโลกของ 3 มิตินั้น เราก็จำเป็นต้องมีระบบ Coordinate แบบ 3 มิติ เราต้องใช้รูปแบบที่ต้องมีแกน 3 แกนตั้งฉากซึ่งกันและกัน ระบบ Coordinate 3 มิติ สามารถทำได้จากระบบ 2 มิติเดิมโดยการเพิ่มแกน Z ขึ้นมาและแกน Z นี้จะต้องผ่านจุด origin ด้วย ซึ่งจะทำให้เกิดระนาบขึ้นมาอีก 2 ระนาบคือ ระนาบXZ , ระนาบYZ และแกน Z ที่เพิ่มขึ้นมานี้ทำให้เกิดระบบใหม่ขึ้นมาอีก 2 ระบบแล้วแต่ทิศทางของแกนZ ถ้าแกน Z ชี้ตรงมายังสายตาของผู้สังเกตุระบบดังกล่าวจะถูกเรียกเป็นระบบมือขวา (righthand system) ในทำนองเดียวกัน ถ้าแกน Z พุ่งออกจากสายตาผู้สังเกตุ ระดับดังกล่าวจะเรียกว่าเป็นระบบมือซ้าย (left hand System) ซึ่งรูปแบบทั้ง 2 อย่างนี้จะแสดงให้เห็นดังรูป 4.1

เพื่อให้เกิดความสะดวกเราจะใช้ระบบมือขวาเป็นระบบ World Coordinate System

4.1.1 Point Representation in 3-D world

โดยปกติแล้วจะมีรูปแบบทั่ว ๆ ไป 2 แบบที่จะแสดงจุดในระบบ 3 มิติคือ

1. แบบระพิกัดฉาก (Rectangular Coordinate)
2. แบบระบบพิกัดทรงกลม (Spherical Coordinate)

Rectangular Coordinate System

จุดต่าง ๆ ที่ใช้ในระบบแบบนี้จะมีรูปแบบเรียงกันมา 3 ตัวคือ (X,Y,Z) ซึ่งค่า X,Y,Z เหล่านี้ได้มาจากการวัดระยะทางจากจุดนั้นไปยังระนาบ YZ,XZ,XY ตามลำดับ ตามรูป 4.2

Spherical Coordinate System

ในระบบแบบนี้แต่ละจุดจะมีรูปแบบเรียงกันมา 3 ตัวคือ (D,θ,ø) ระยะทางจากจุด P ไปยังจุด origin จะมีค่า = D และมุมระหว่างเส้นวัดไปทิศทางบวกของแกน Z จะมีค่า = ø ส่วนมุมระหว่าง $\overline{OP}$ และวัดไปทางทิศทางบวกของแกน X คือ θ (คือส่วนฉายของ $\overline{OP}$ ลงบนระนาบ XY) มุม θ นี้จะถูกวัดในทิศทางทวนเข็มนาฬิกาจากแกน X ดูรูป (4.2)

4.1.2 Relation Betureen the rectangular coordinate and Spherical coordinate system

เนื่องจากระบบทั้งสองคือระบบ Rectangular Coordinate และระบบ Spherial Coordinate จะถูกนำมาใช้เปลี่ยนแปลงกันใน Computer graphic และบ่อยครั้งที่เดียวที่จำเป็นจะต้องเปลี่ยนระหว่างระบบทั้งสอง ดังนั้นจึงต้องมีการหาค่าความสัมพันธ์ระหว่างระบบทั้งสอง และจากกฎเกณฑ์ทางตรีโกณมิติเราจะหาค่าความสัมพันธ์ได้ดังนี้

เมื่อต้องการเปลี่ยนจาก Spherical เป็น Rectangular เราจะหาค่า X,Y,Z จาก D, θ, ø ได้ดังนี้

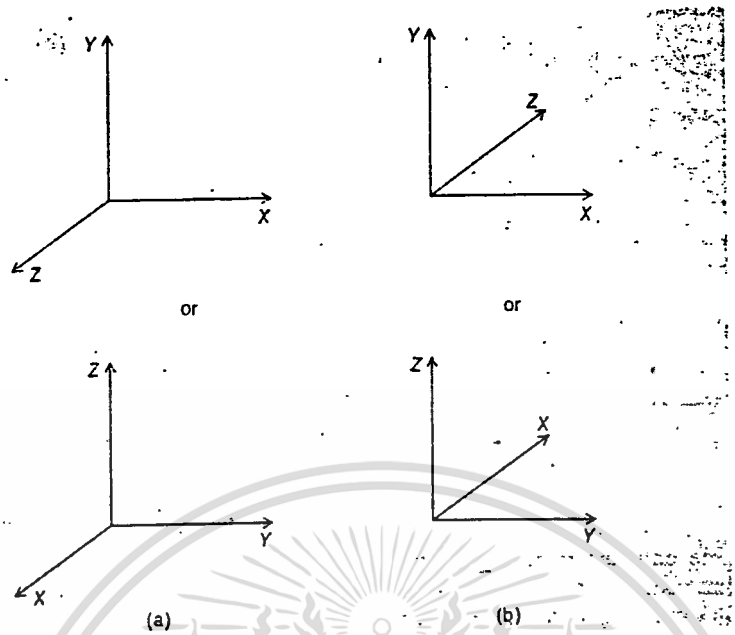


FIGURE 4.1 Coordinate systems: (a) right-handed system and (b) left-handed system

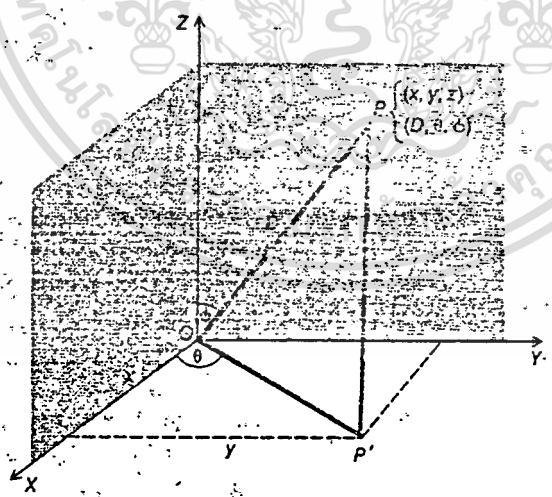


FIGURE 4.2 Relationship between the rectangular and spherical coordinate systems

$$X = D \sin\phi \cos\theta$$
$$Y = D \sin\phi \sin\theta$$
$$Z = D \cos\phi \qquad (4.1)$$

และเมื่อต้องการเปลี่ยนจาก Rectangular เป็น spherical เราจะหาค่า D, θ, ∅ จาก X, Y, Z ได้ดังนี้

$$D = X^2 + Y^2 + Z^2$$
$$\theta = \tan^{-1}(y/x)$$
$$\phi = \cos\theta^{-1}(Z/D)$$

สำหรับตัวอย่าง เช่น สมมุติว่ามีจุดอยู่ที่ (4,2,3) ในระบบ Rectangular Coordinate จะเปลี่ยนได้เป็นจุด (5,4, 27°, 56°) ในระบบ spherical Coordinate

4.2 Transformation Matrix

ตามที่กล่าวมาแล้วในบทที่ 3 เราจะรับเอาระบบ homogenous coordinate มาใช้ในระบบ Transformation แบบ 3 มิติ และเพื่อให้สะดวกเราจะใช้ระบบ Rectangular Coordinate ในการแทน Vector (X,Y,Z) ดังนั้นการสร้าง Transformation Matrix ขนาด 4 x 4 สำหรับจุด ๆ หนึ่งในระบบ 3D homogenous coordinate จะมีรูปแบบดังนี้

$$\left[\begin{array}{ccc:c} A & B & C & 0 \\ D & E & F & 0 \\ G & H & I & 0 \\ \hdashline L & M & N & 1 \end{array}\right] \qquad (4.2)$$

และใน Transformation Matrix นี้ยังสามารถแบ่งได้เป็น 3 Matrix ย่อยคือ

$$\left[\begin{array}{c|c} \begin{matrix} I \\ (3x3) \end{matrix} & \begin{matrix} III \\ (4x1) \end{matrix} \\ \hline \begin{matrix} II \\ (1x3) \end{matrix} & \end{array}\right] \qquad (4.3)$$

Matrix ย่อยที่ 1 คือ ส่วนที่จะนำมาใช้ในการ Scaling shearing, reflection และ Rotation วัตถุ 3 มิติ

Matrix ย่อยที่ 2 จะเป็นส่วนช่วยในการสร้าง Linear Translation

Matrix ย่อยที่ 3 จะเป็นส่วนที่รวมการ Translate และการ Transformation อื่นในรูปแบบการคูณได้

ในที่นี้เราจะกล่าวให้เห็นถึงคุณสมบัติต่าง ๆ ของ Matrix ย่อยในส่วนนี้ และจะยกตัวอย่างให้เห็นว่า Matrix เหล่านี้มีผลต่อการเปลี่ยนแปลงต่อปิรามิดอย่างไร ดูรูป 4.3

4.2.1 <u>3D Translation</u>

การ Transformation นี้จะทำการย้ายตำแหน่งจากจุด ๆ หนึ่ง (X, Y, Z) ไปยังจุด ๆ ใหม่ (C',Y',Z') เป็นระยะทาง (L,M,N) จะถูกแทนด้วย Transformation Matrix ดังนี้คือ

$$(X',Y',Z',1)=(X,Y,Z,1)\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ L & M & N & 1 \end{bmatrix} \qquad (4.4)$$

เมื่อ L,M,N คือระยะทางที่ต้องเคลื่อนจากจุด X,Y,Z ในรูป 4.3 (b) จะแสดงให้เห็นการย้ายรูปปิรามิดไปเป็นระยะทาง 1,2,1 หน่วย ในทิศทาง X,Y,Z ตามลำดับ

4.2.2 <u>3D Scaling</u>

รูปแบบของ Transformation Matrix ขนาด 4x 4 ในการ

scaling จะมีลักษณะดังนี้

$$(X',Y',Z',1) = (X,Y,Z,1)\begin{bmatrix} A & 0 & 0 & 0 \\ 0 & E & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.5)$$

จาก Matrix ข้างบนนี้จะแสดงให้เห็นว่าเราสามารถขยายแต่ละ Coordinate แยกกันได้โดยการควบคุมการเคลื่อนที่ของโดออดิเนตแต่ละแกน เช่น ถ้าต้องการให้มีการขยายให้เท่า ๆ กันทุก ๆ ส่วนจะต้องทำการปรับค่าให้ A = E =I และถ้าต้องการให้วัตถุขยายขึ้นเป็น 2 เท่า เราก็ต้องปรับค่า A = E =I=2 ผลของการ scaling จะแสดงให้เห็นในรูป 4.3 (C)

4.2.3 3D Shearing

ในบทที่ 3 2D shearing จะได้มาจาก (off diagonal) เทอมทะแยงทางซ้ายของ Matrix ยอยขนาด 2x 2 ของ Transformation Matrix ขนาด 2 มิติ แต่ shearing ในระบบ 3 มิติก็คือส่วนที่เพิ่มเติมมาจาก shearing 2 มิติคือ จะได้มาจาก (off diagonal) เทอมทะแยงทางด้านซ้ายมือของ Matrix ยอยขนาด 3 x 3 เพราะฉะนั้นรูปแบบของ 3 D sluaring จะเป็นดังนี้

$$(X',Y',Z',1) = (X,Y,Z,1)\begin{bmatrix} 1 & B & C & 0 \\ D & 1 & F & 0 \\ G & H & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.6)$$

การ shearing ของรูป Pyramid แสดงให้เห็นดังรูป 4.3 (d)

4.2.4 3 - D Reflection

วัตถุจะถูกกลับข้างไปยังอีกระนาบหนึ่งได้โดยการเปลี่ยนส่วนทะแยงมุมทางด้านขวาของ 3 D Transformation Matrix ตามที่กล่าวมาแล้วในบทที่ 3 วัตถุใด ๆ จะถูกสะท้อนกลับไปยังอีกระนาบหนึ่งได้โดยการสร้างรูป

สะท้อนของทุก ๆ จุดของวัตถุไปลงบนด้านตรงข้ามของระนาบ เช่นตัวอย่าง ถ้าต้องการสะท้อนวัตถุให้ไปตกลงบนระนาบ XY ก็ทำได้โดยการเปลี่ยนค่าของ Z Coordinate เท่านั้น โดยการเปลี่ยนให้มีเครื่องหมายตรงกันข้ามกับที่เป็นอยู่ในเวลานั้น ๆ รูปแบบการ Transform จะมีลักษณะดังนี้คือ

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

ในทำนองเดียวกัน ถ้าต้องการให้วัตถุไปอยู่บนระนาบ YZ ก็ทำได้โดยการเปลี่ยนเครื่องหมาย X รูปแบบการ Transform จะมีลักษณะดังนี้คือ

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

และสำหรับการให้วัตถุตกไปอยู่บนระนาบ XZ ก็ทำได้โดยการเปลี่ยนเครื่องหมาย Y รูปแบบการ Transform จะมีลักษณะดังนี้คือ

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

ตัวอย่างการทำให้ปิรามิดไปอยู่บนระนาบ YZ จะถูกแสดงให้เห็นดังรูป 4.3 (C)

4.2.5 3 D Rotation

ก่อนที่จะพิจารณาการสร้างการหมุนแบบ 3 มิติรอบแกนใด ๆ เราต้องมีความรู้เกี่ยวกับการหมุนรอบแกน X,Y,Z ก่อน สำดับแรกให้เราพิจารณาการหมุนรอบแกน X ในกรณีนี้ค่าของ X จะไม่มีการเปลี่ยนแปลง ดังนั้น Transformation Matrix จะมีค่า =0 อยู่ในแถวที่ 3 และคอลัมภ์ที่ 3 โดยไม่รวม 1 ในแนวทะแยงหลัก และเทอมอื่นก็ว่าหนดได้เหมือนกับการหมุนในระบบ

2 มิติ เช่น ในสมการ 3.9 ซึ่งก็เป็นตัวอย่างการหมุนรอบแกน ในระบบ 3 มิติ รูปแบบนี้ทำให้เกิด Transformation Matrix ของการหมุนรอบแกน Z จะมีค่าดังนี้

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

โดยทั่วไปแล้วเราจะวัดการหมุนของ θ ในทิศทางทวนเข็มนาฬิการอบจุด origin เมื่อมองที่ตำแหน่งจุด origin จากจุดบนแกน X และการ Transformation Matrix ของการหมุน θ รอบแกนX และแกน Y ก็จะมีค่าเป็นดังนี้

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

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

การหมุนระบบ 3 มิติรอบแกนหนึ่ง ๆ ในระบบถูกแสดงให้เห็นดังรูป 4.4

ตัวอย่างการหมุนรอบแบบ 3 มิติ ให้เราพิจารณาปิรามิดที่ให้ในรูป 4.3 (a) ในขณะที่ Pyramid ถูกหมุนทวนเข็มนาฬิกาเป็นมุม 30 องศา รอบแกน Z โดยใช้สมการ 4.10 เราจะได้ Coordinate ใหม่ดังนี้

A เดิม = 3,1,0

B เดิม = 3,5,0

C เดิม = 1,4,0

D เดิม = 2,3,3

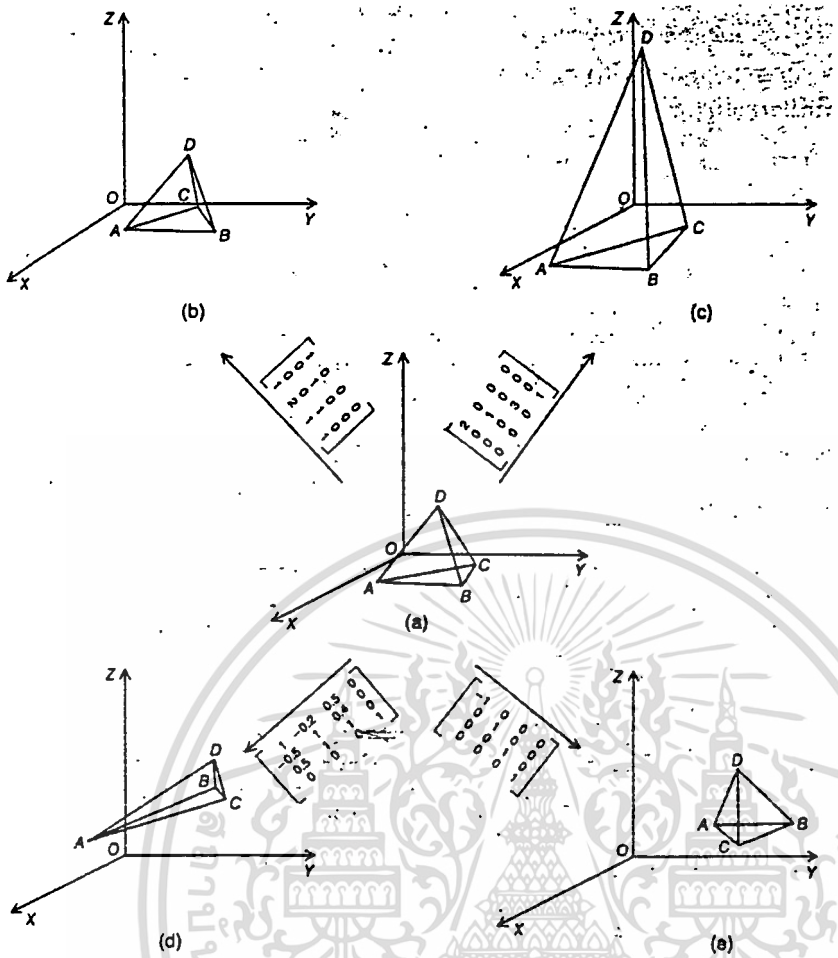


FIGURE 4.3 Three-dimensional transformations: (a) original position, (b) translation, (c) scaling, (d) shearing, and (e) reflection

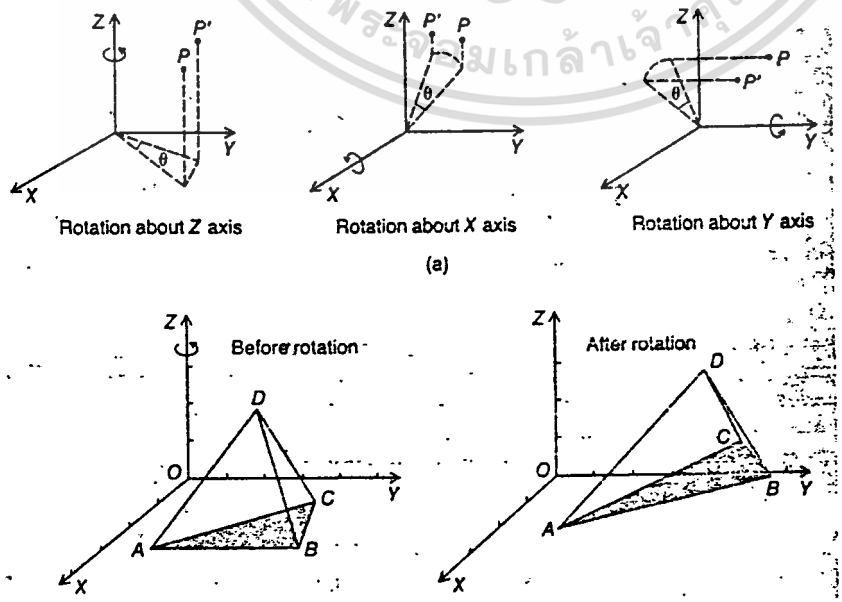


FIGURE 4.4 Three-dimensional rotations: (a) rotation of a point and (b) rotation of an object

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

A',B',C',D' คือ coordinate ใหม่

$$= \begin{bmatrix} 2.1 & 2.37 & 0 & 1 \\ 0.1 & 3.83 & 0 & 1 \\ 1.13 & 3.96 & 0 & 1 \\ 0.232 & 3.60 & 3 & 1 \end{bmatrix}$$

ผลของการ Transform เป็นดังรูป 4.46

สมมุติว่ารูปปริซึมเดียวกันนี้ถูกหมุนเป็นมุม −30 องศา (ตามเข็มนาฬิกา) รอบแกน Z เมื่อ $\theta = -30$ และจากสมการ 4.10 เราจะได้ว่า

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

และเมื่อเราคูณ Transformation Matrix $R(30^{\circ})$ เข้ากับ $R(-30^{\circ})$ เข้าด้วยกันผลจะเป็นอย่างไร ? ผลที่ได้ก็คือ Inverse matrix ของ I

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

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

นี่ก็หมายความว่าเมื่อเราใช้ Matrix หมุนตามเข็มนาฬิการอบแกนใด ๆ ก็จะเหมือนกันกับการ Inverse Matrix การหมุนในทิศทางทวนเข็มนาฬิกาของแกนเดียวกัน ตัวอย่างข้างบนไม่ใช่กรณีเดียวเท่านั้น แต่มันจะแทนกฎเกณฑ์ทั่ว ๆ ไป ซึ่งเราจะพบว่ามันมีค่ามากใน computer graphic

กฎเกณฑ์พื้นฐานแบบนี้หมายความว่า

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

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

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

เป็นลักษณะที่สำคัญที่ควรจะจำไว้ว่า ลำดับการคูณของ Matrix จะมีผลต่อผลลัพธ์ที่ได้นี่ย่อมหมายความว่าการหมุนแบบ 3D นั้นไม่ใช่กฎการสลับที่ เพื่อที่จะแสดงให้เห็นจริง จงพิจารณาการหมุน θ 1 รอบแกน Z ต่อจากนั้นก็หมุนเป็นมุม θ 2 รอบแกน X ผลลัพธ์ที่ได้เป็นดังนี้

$$R(\theta_1)_{Z} \cdot R(\theta_2)_{X} = \begin{bmatrix} \cos\theta 1 & \sin\theta 1\cos\theta 1 & \sin\theta 1\cos\theta 2 & 0 \\ -\sin\theta 1 & \cos\theta 1\cos\theta 2 & \cos\theta 1\sin\theta 2 & 0 \\ 0 & -\sin\theta 2 & \cos\theta 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.16)$$

เมื่อกลับการทำงาน (หมุน θ2 รอบแกน X ตามด้วย θ1 รอบแกน Z) เราจะได้ Matrix ดังนี้

$$R(\theta_2)X \cdot P(\theta_1)Z = \begin{bmatrix} \cos\theta 1 & \sin\theta 1 & 0 & 0 \\ -\cos\theta 2\sin\theta 1 & \cos\theta 2\cos\theta 1 & \sin\theta 2 & 0 \\ \sin\theta 2\sin\theta 1 & -\sin\theta 2\cos\theta 1 & \cos\theta 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.17)$$

เมื่อมีการเปรียบเทียบผลลัพธ์ที่ได้จะเห็นว่าผลลัพธ์ที่ได้จะไม่เหมือนกัน เพราะฉะนั้นความจริงที่ว่าการหมุนนั้นไม่ใช่กฎการสลับที่จะต้องจำไว้เมื่อมีการคูณ matrix 3 มิติ เข้ามาเกี่ยวข้อง

4.2.6 Change of coordinate system

มาจนถึงจุดนี้ เราได้แสดงให้เห็นวิธีการ Transform จุดหรือกลุ่มของจุดจากรูปแบบหนึ่งไปเป็นรูปแบบหนึ่งซึ่งทั้งสองกลุ่มนี้ยังคงอยู่ใน coordinate เดียวกันในกรณีเช่นนี้ระบบ coordinate ยังคงไม่มีการเปลี่ยนแปลง และวัตถุถูกเปลี่ยนโดยเปรียบเทียบกับจุด origin ของระบบ coordinate เท่านั้น ใน computer graphic บ่อยครั้งที่เราจำเป็นต้องสร้างระบบโคออดิเนตใหม่โดยไม่มีการเปลี่ยนตำแหน่งของจุดหรือวัตถุนั้น ซึ่งการกระทำในลักษณะนี้จะเกี่ยวข้องกับการ Transform ระบบ coordinate ในกรณีเช่นนี้เราสามารถอ้างอิงถึงจุดทั้งหลายในเทอมของระบบ coordinate เดิมกับ coordinate ใหม่ที่เปลี่ยนไปเพื่อให้เห็นจริงให้พิจารณารูป 4.5 จะเห็นว่าจุด P มี coordinate อยู่ที่ (2,3,4) ในระบบ coordinate (X,Y,Z) แต่จะมีโคออดิเนตอยู่ที่ (1,2,2) ในระบบ coordinate (X',Y',Z')

การเปลี่ยนระบบ coordinate จาก X,Y,Z ไปเป็น X',Y',Z' จะกระทำได้โดยการย้ายตำแหน่งจุด origin O ไปยังตำแหน่ง O' เพราะฉะนั้นจุดกำเนิด O' สำหรับระบบโคออดิเนต X',Y',Z' จะตั้งอยู่ในตำแหน่ง (1,1,2) รูปแบบการ Transform จนเป็นดังนี้

$$P' = P.T = (2,3,4,1) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -1 & -2 & 1 \end{bmatrix} = (1,2,2,1)$$

ตอนนี้เราจะพิจารณาผลการหมุนระบบ coordinate ในทิศทางตามเข็มนาฬิกาเป็นมุม 30 องศา รอบแกน Z ซึ่งจะทำให้เกิดระบบ coordinate (X',Y',Z') ดังรูป 4.6 (b) จะสังเกตุได้ว่าจุด P ที่ดูงอยู่ยังคงไม่มีการเปลี่ยนแปลงและสมมุติว่าเราหมุนจุด P ในทิศทางทวนเข็มนาฬิกาเป็นมุม 30 องศา รอบแกน Z ก็จะไม่ทำให้รูปแบบของระบบ coordinate เดิมเปลี่ยนไปซึ่งแสดงให้เห็นจริงดังรูป 4.6 (a)

ขณะนี้ก็เป็นที่แน่นอนว่าตำแหน่งการหมุนจุด P' ที่สัมพันธ์กับระบบ Coordinate (X,Y,Z) ก็มีลักษณะเหมือนกับตำแหน่ง P ที่สัมพันธ์กันการหมุนระบบ Coordinate (X' Y' Z') เพราะฉะนั้น matrix ที่ต้องการใช้แทนการ Transform Coordinate ก็คือ Inverse ที่ใช้ในการ Transform จุดที่ไม่มีการเปลี่ยนแปลง Coordinate เพื่อให้เห็นจริงสมมุติว่าจุด P ในรูป 4.6 (a) อยู่ในตำแหน่ง (1,3,2) จากสมการ (7.10) Transformation Matrix ที่ใช้ในการหมุนจุด P' คือ

$$P' = (P.R(30)z) = (1,3,2,1) \begin{bmatrix} \cos 30^\circ & \sin 30^\circ & 0 & 0 \\ -\sin 30^\circ & \cos 30^\circ & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= (-0.63,\ 3.1,\ 2.1)$$

ผลลัพธ์ที่ได้นี้มาจากการหมุนในทิศทางทวนเข็มนาฬิการอบแกน Z ตำแหน่ง P นี้จะสัมพันธ์กับการหมุนระบบ Coordinate (X'Y'Z') ในทิศทางตามเข็มนาฬิการอบแกน Z ก็ยังมีค่าเป็น (−0.63, 3.1, 2.1) ตัวอย่างนี้ไม่ใช่กรณีพิเศษแต่มันจะแทนกฎเกณฑ์ทั่วไปที่เราทำให้ได้รูปแบบ 2 มิติ ที่แสดงลักษณะของวัตถุ 3 มิติ

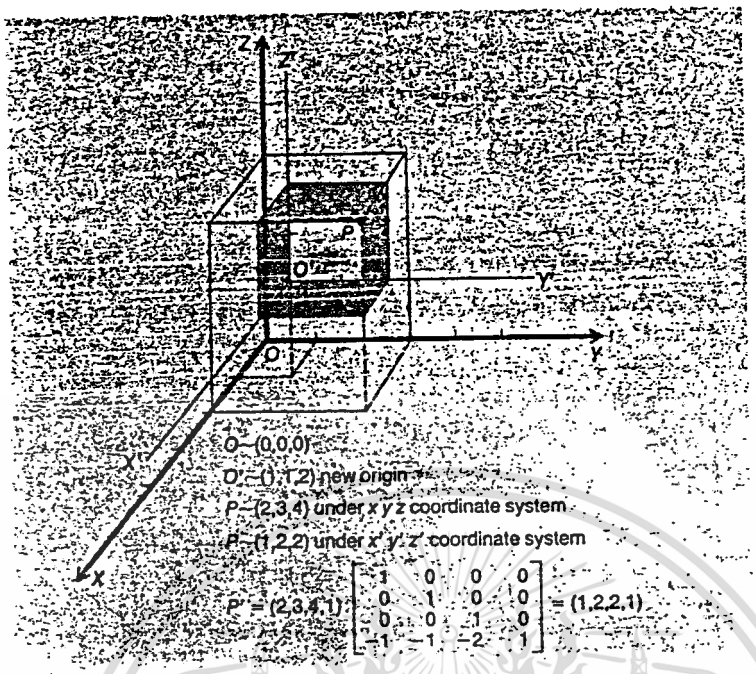


FIGURE 4.5 Point P and two different coordinate systems

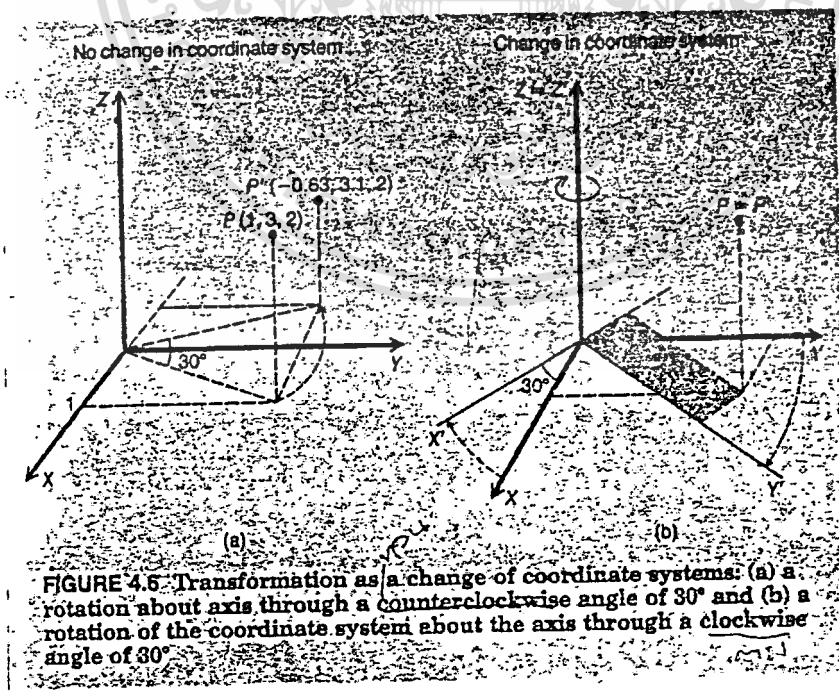


FIGURE 4.6 Transformation as a change of coordinate systems: (a) a rotation about axis through a counterclockwise angle of 30° and (b) a rotation of the coordinate system about the axis through a clockwise angle of 30°

4.2.7 3D Rotation about an Arbritary Axis

การหมุนรอบจุด ๆ หนึ่งของแกนใด ๆ จะกระทำได้โดยการรวมพื้นฐานเกี่ยวกับ Transformation ที่กล่าวไว้แล้วในส่วนก่อน เราจะเริ่มต้นโดยพิจารณารูป 4.7 (a) เมื่อเราต้องการหมุนจุด P รอบแกน OJ ที่ผ่านจุด origin และเราจะกำหนดจุด Q (q_1, q_2, q_3) เป็นจุดที่จุด P หมุนไปรอบ ๆ เมื่อมีการหมุนรอบแกน OJ

Direction Cosine

ในการกำหนดทิศทางการหมุนของแกน เราต้องใช้ Cosine 3 ทิศทางคือ cosx, cosy, cosz (มุม r,b,x ถูกกำหนดไว้ดังรูป 4.7b) ทิศทาง Cosine ของแกนหมุนจะหาได้จากในรูป 4.7 (b) เราจะเห็นว่าสามเหลี่ยม OAQ เป็นสามเหลี่ยมมุมฉากมีมุม A เป็นมุมฉากดังนั้นจะได้ว่า $\cos = q/r$ เพราะค่า $r = (q_1^2 + q_2^2 + q_3^2)^{\frac{1}{2}}$ เราสามารถกำหนดทิศทาง cosine ได้ตามนี้

$$\cos\alpha = \frac{q_1}{r} = \frac{q_1}{(q_1^2 + q_2^2 + q_3^2)^{\frac{1}{2}}}$$

$$\cos\beta = \frac{q_2}{r} = \frac{q_2}{(q_1^2 + q_2^2 + q_3^2)^{\frac{1}{2}}} \qquad (4.18)$$

$$\cos\gamma = \frac{q_3}{r} = \frac{q_3}{(q_1^2 + q_2^2 + q_3^2)^{\frac{1}{2}}}$$

จากค่าต่าง ๆ เหล่านี้จะทำให้ได้ค่า

$$\cos^2\alpha + \cos^2\beta + \cos^2\gamma = \frac{q_1^2 + q_2^2 + q_3^2}{r^2} = 1 \qquad (4.19)$$

คุณสมบัติที่สำคัญของทิศทาง cosine คือว่าถ้าเลือกขนาด A,B,C ให้มีค่าเป็นสัดส่วนกับทิศทาง cosine (เรียกว่าขนาดทิศทาง) เราก็สามารถที่จะสร้างความสัมพันธ์ระหว่างค่าของทิศทาง cosine ทั้งสามได้ ให้พิจารณารูป 4.7 (c)

Vector K จะเป็นจุดเริ่มต้นที่จุด origin ของระบบพิกัดฉาก และค่าของ (a,b,c) จะอยู่ในทิศทางบวกตามแกน (X', Y', C') เราจะต้องเลือกค่าของ A,B,C ที่จะทำให้ $a = \cos\alpha$, $b = \cos\beta$, $c = \cos\gamma$ เมื่อค่า γ, α, β ได้ถูกกำหนดไว้ก่อนหน้านั้นแล้ว ดังนั้นค่าความยาว $\overline{OK}$ จะมีค่าดังนี้คือ

$$r' = (a^2 + b^2 + c^2)^{\frac{1}{2}} = (\cos^2 + \cos^2 B + \cos^2 r) = 1 \quad (4.20)$$

สามเหลี่ยม O'A'K เป็นสามเหลี่ยมมุมฉากโดยมีมุม A เป็นมุมฉาก, เพราะฉะนั้นจะได้ว่า $\cos\alpha' = \cos\alpha / r' = \cos\alpha / 1 = \cos\alpha$ ในทำนองเดียวกันเราก็สามารถพิสูจน์ได้ว่า $\cos\beta' = \cos\beta$ และ $\cos\gamma' = \cos\gamma$ นี้ย่อมแสดงให้เห็นว่า OQ และ O'K มีทิศทาง cosine เดียวกัน

Transformation Matrix

ในทันทีที่ทิศทาง cosine ของแกน OJ ได้กำหนดขึ้นแล้ว เราจะต้องดำเนินขั้นตอนการหมุนจุด P เป็นมุม θ รอบแกน OJ ได้ดังนี้

ขั้นตอนที่ 1 ย้ายตำแหน่งจุด origin ไปอยู่ในตำแหน่งใหม่ที่ทำให้กลายเป็นจุด origin ใหม่เพื่อให้สังเกตุได้ เราจะให้แกนที่เกิดขึ้นภายหลังการ Transform จะให้เป็นระบบแกน (X',Y',Z') และระบบแกน X',Y',Z' ที่ได้จากการ Transfoim ครั้งสุดท้าย ก็จะเป็นระบบแกน XYZ มาตรฐาน ผลลัพธ์ของการ Translate ถูกเขียนเป็น Matrix ได้ดังนี้

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -q_1 & -q_2 & -q_3 & 1 \end{bmatrix}$$

ขั้นตอนที่ 2 ปรับแกน OJ ให้ทับแกน Z ซึ่งทำได้โดยใช้ 2 ขั้นตอนคือ

1. หมุนรอบแกน X และต่อมาหมุนรอบแกน Y เพราะว่าทั้งในรูป 4.7(b) และ 4.7(c) อยู่ในเทอมของทิศทาง OJ, เราจะใช้รูป 4.7 c กำหนดการหมุนตลอดแกน ถ้าเรา project O'K ลงบนระนาบ Y'Z' เราจะได้ผลลัพธ์ดังรูป 4.8 (a) ต่อจากนั้นหมุน O'K ตามเข็มนาฬิกาเป็นมุม ϕ 1 รอบแกน X จะทำให้ O'K วางอยู่ในระนาบ X'Y' จากนี้เราสามารถคำนวณหาค่า $\cos\phi$ 1, $\sin\phi$ 1 ได้ดังนี้

$$\cos\phi 1 = c/v$$

$$\sin\phi 1 = b/v$$

เมื่อ $v = (b^2+c^2)^{\frac{1}{2}}$ และเมื่อใส่ Parameter เหล่านี้ลงในสมการ 4.11 เราจะได้ Transformation Matrix ของการหมุนนี้คือ

$$R_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/v & b/v & 0 \\ 0 & b/v & c/v & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ตำแหน่งที่เปลี่ยนไปภายหลังการหมุนครั้งแรกเป็นดังรูป 4.8 (b) มุมการหมุน 02 สามารถกำหนดได้ด้วยวิธีคล้าย ๆ กัน น่าสังเกตุว่าเมื่อ $O'K = r' = 1$ ยอมหมายความว่า KW จะมีค่าเป็น $v = O'K = ((KW)^2+a^2)^{\frac{1}{2}} = r'$, $r' = (a^2+b^2+c^2)$ $(KW)^2 = b^2+c^2 = v^2$ เนื่องจากเราวัดมุม 02 ในทิศทางตามเข็มนาฬิกาเราจะได้ว่า

$$\cos(-\phi 2) = \cos\phi 2 = v$$

$$\sin(-\phi 2) = -\sin\phi 2 = -a$$

ค่า Parameter เหล่านี้จะนำมาใส่ในสมการ 4.12 ทำให้ได้ว่า

$$R_2 = \begin{bmatrix} v & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & v & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

เพราะฉะนั้นการหมุน O'K ในทิศทางตามเข็มนาฬิกา เป็นมุม ϕ 2 รอบแกน Y' จะเป็นการปรับแกน O'K ให้อยู่แนวเดียวกับแกน Z

<u>ขั้นตอนที่ 3</u> ให้ทำการหมุนเป็นมุม θ รอบแกน Z ของระบบ Coordinate ใหม่ Transformation Matrix ที่ได้จะมีค่า

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

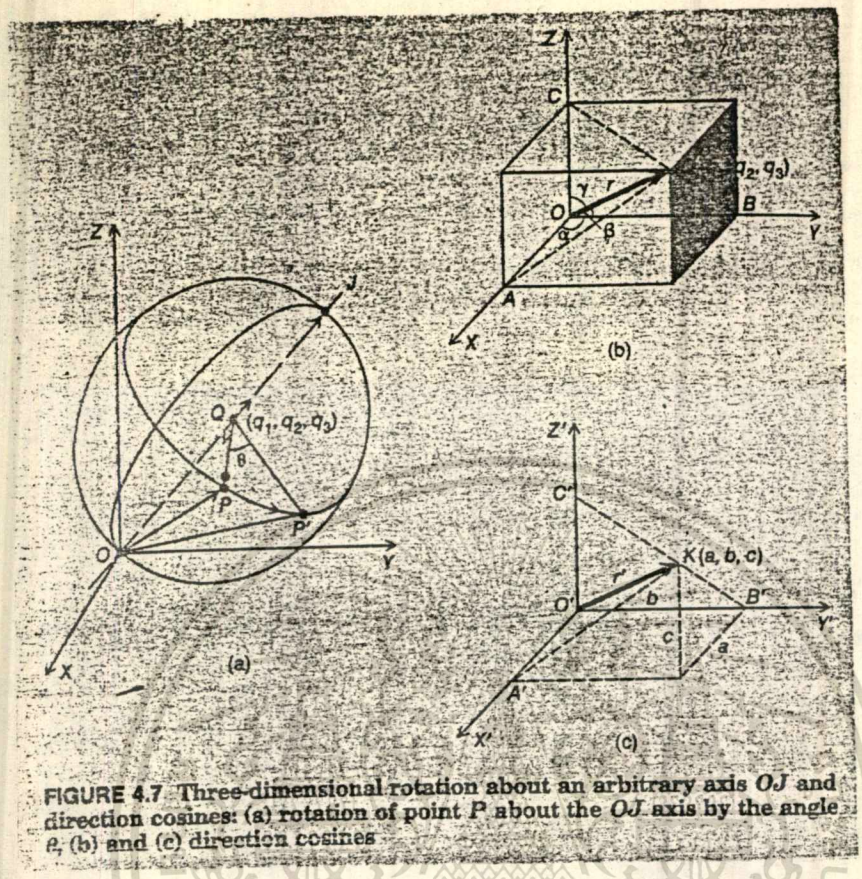


FIGURE 4.7 Three-dimensional rotation about an arbitrary axis OJ and direction cosines: (a) rotation of point P about the OJ axis by the angle θ, (b) and (c) direction cosines

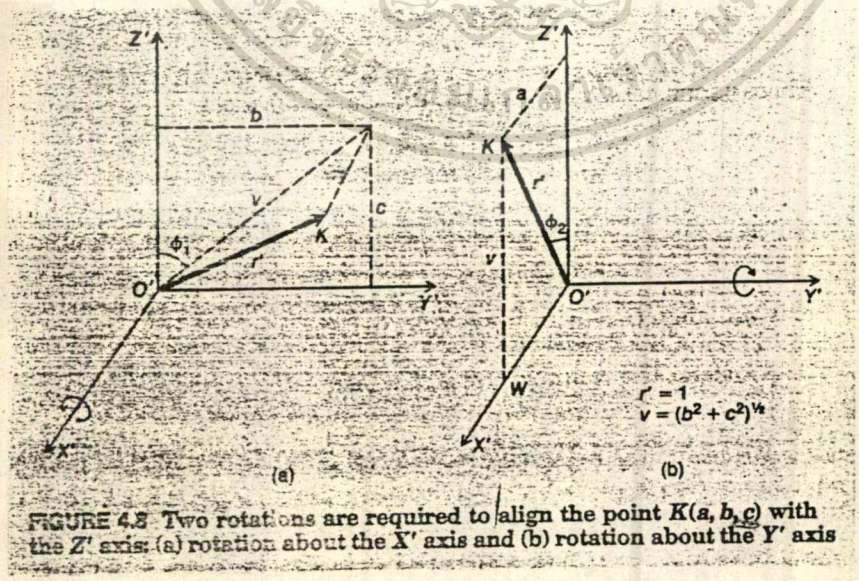


FIGURE 4.8 Two rotations are required to align the point $K(a, b, c)$ with the Z' axis: (a) rotation about the X' axis and (b) rotation about the Y' axis

ค่ามุม θ ข้างบนนี้จะถูกวัดในทิศทางทวนเข็มนาฬิกา แต่ถ้าเป็นการหมุนตามเข็มนาฬิกาเราก็จะแทนค่า θ ใน $R_z(\theta)$ ด้วย $-\theta$

<u>ขั้นตอนที่ 4</u> ส่งวัตถุกลับไปยังระบบ coordinate เดิม ซึ่งกระทำได้โดยใช้ Inverse ของ step 3 และ Inverse ของ step 1 ลำดับของการ Inverse ต้องเรียงลำดับให้ถูกต้องคือ ต้อง Inverse R_2^{-1} ก่อนแล้วตามด้วย R_1^{-1} และ T_1^{-1}

การ Transform ที่สมบูรณ์ของทั้ง 4 ขั้นตอนคือ

$$T = T_1 \cdot R_1 \cdot R_2 \cdot R_z(\theta) \cdot R_2^{-1} \cdot R_1^{-1} \cdot T_1^{-1}$$
$$= R_1 \cdot R_2 \cdot R_z(\theta) R_2^{-1} \cdot R_1^{-1}$$

เราสามารถที่จะรวมเป็น Transformation Matrix เพียง Matrix เดียวได้และจะมีค่าดังนี้

$$T = \begin{bmatrix} A & B & C & 0 \\ D & E & F & 0 \\ G & H & I & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.21)$$

เมื่อ
$$A = a^2 + (1-a^2)\cos\theta$$
$$B = ab(1-\cos\theta) + c\sin\theta$$
$$C = ac(1-\cos\theta) - b\sin\theta$$
$$d = ab(1-\cos\theta) - c\sin\theta$$
$$e = b^2 + (1-b^2)\cos\theta$$
$$f = bc(1-\cos\theta) + a\sin\theta$$
$$G = ac(1-\cos\theta) + b\sin\theta$$
$$H = ac(1-\cos\theta) - a\sin\theta$$
$$I = c^2(1-c^2)\cos\theta$$
$$a = \cos\alpha = q_1/r$$
$$b = \cos\beta = q_2/r$$
$$c = \cos\gamma = q_3/r$$
$$r = (q_1^2 + q_2^2 + q_3^2)^{\frac{1}{2}}$$

สุดท้ายเราก็คำนวณการหมุนตำแหน่งของจุด P ไปยังตำแหน่ง P' ได้ดังนี้คือ

$$P' = (X,Y,Z, 1) \cdot T \qquad (4.22)$$

ตัวอย่าง

เพื่อแสดงให้เห็นลำดับขั้นการคำนวณ, เราจะใช้ตัวอย่างที่มี Parameter P, Q และ Q ดังนี้

$P(X,Y,Z) = (1,2,3)$

$Q(q_1,q_2,q_3) = (4,8,5)$

$\theta = 30^\circ$

ต่อมาใช้สมการ 4.18 จะได้ทิศทาง cosine มีค่าดังนี้

$\cos\alpha = 4/105 = 0.39036 = a$

$\cos\beta = 8/105 = 0.78072 = b$

$\cos\gamma = 5/105 = 0.48795 = c$

$r = (4^2 + 8^2 + 5^2) = 105$

และ $\cos 30^\circ = 0.866, \sin 30^\circ = 0.5$ นำค่า Parameter เหล่านี้ใส่ลงในสมการ 4.21 ทำให้ได้

$$T = \begin{bmatrix} 0.866 & 0.285 & -0.365 & 0 \\ -0.203 & 0.948 & 0.246 & 0 \\ 0.416 & -0.144 & 0.898 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

∴ เราจะได้ตำแหน่งของ P ใหม่คือ

$$P' = (1,2,3,1) \cdot T = (1.73, 1.75, 2.83, 1)$$

Rotation to the rotation about the standard axis

เราสามารถที่จะแสดงให้เห็นว่าการหมุนรอบระบบ Coordinate มาตรฐานเป็นรูปแบบที่พิเศษของ Matrix การหมุนทั่ว ๆ ไป (ตามสมการ 4.10, 4.11, 4.12) โดยเราจะพิจารณาในกรณีเฉพาะการหมุนรอบแกน Z สำหรับในกรณีนี้จะมีค่า Coordinate Translate เข้ามาเกี่ยวข้องเพราะว่าการหมุนถูกกระทำรอบแกนมาตรฐาน เพราะฉะนั้นค่า Q = (0,0,0) และ

OJ = OZ และทิศทางของ cosine เมื่อคำนวณหาแล้วจะมีค่าดังนี้

$$\cos\alpha = \cos 90 = 0 = a$$
$$\cos\beta = \cos 90 = 0 = b$$
$$\cos\gamma = \cos\theta = 1 = c$$

เมื่อเรานำค่า Parameter เหล่านี้ใส่ลงไปในสมการ 4.21 จะทำให้ได้ Matrix การหมุนมีค่าดังนี้

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

เราจะเห็นได้ว่า Matrix ที่ได้เหมือนกับ Matrix ในสมการ 4.10

4.3 Viewing in Three Dimension

จนกระทั่งถึงจุดนี้แล้วเราได้มีความรู้เกี่ยวกับ 3 มิติ และการ Transform ในลักษณะต่าง ๆ มาบ้างแล้วในส่วนนี้เราจะแสดงให้เห็นว่าการแสดงภาพ 3 มิติลงบนจอภาพคอมพิวเตอร์ เพราะว่าจอภาพคอมพิวเตอร์เป็นขนาด 2 มิติ เราจึงจำเป็นต้องมีวิธีการบางอย่างที่จะทำให้เกิดภาพ 2 มิติที่แทนรูปร่างของวัตถุ 3 มิติ วิธีการดังกล่าวคือ Perspective Projection และจะถูกนำมาใช้สร้างภาพ 2 มิติที่เรียนแบบจากวิธีการมองวัตถุ 3 มิติจริง เราจะกล่าวถึงรูปแบบ Perspective Projection ทั้ง 2 ระบบคือ Central Projection และ Arbitrary Projection ภายหลังที่เราได้รับภาพ Perspective ของวัตถุ 3 มิติลงบนระนาบ 2 มิติแล้วก็ต้องมีการเปลี่ยนรูปแบบจาก world coordinate ไปเป็น eye coordinate ต่อจากนั้นก็จะใช้ Viewing Transformation คำนวณหาตำแหน่งต่าง ๆ ที่อยู่บนจอภาพคอมพิวเตอร์ของแต่ละจุดบนภาพ 3 มิติของวัตถุนั้น

4.3.1 Central Projection

รูป 4.9 คือ รูปแบบของ Central Projection ที่มีการใช้มากใน computer graphic ในรูปแบบของ Central Projection จุดสังเกตุ (viewpoint) จะต้องตั้งอยู่ในตำแหน่งใดบนแกนหนึ่งของระบบ world coordinate ในขณะที่แกนของระบบ eye coordinate จะถูกสร้างไว้ที่ตำแหน่งจุดศูนย์กลางของ Projection เราสมมุติว่าศูนย์กลางของการ Projection ถูกตั้งอยู่บนแกน Z นี้ย่อมหมายความว่า เส้นทางการมองถูกปรับให้เป็นเส้นตรงเดียวกันกับแกน Z ในระบบ World coordinate เพื่อว่าจะทำให้จอภาพขนานกันกับระนาบ XY Perspective display จะได้รับจากการฉายจุดแต่ละจุดของวัตถุหนึ่ง ๆ ลงบนจอภาพตามที่แสดงไว้ในรูป 4.9

เราจะใช้รูปแบบของระบบ eye coordinate เป็นแบบระบบมือซ้าย (left hand coordinate) และจะแทนระบบ eye coordinate ในเทอมของ (Xe,Ye,Ze) แกน Ze ชี้ตรงไปยังจุด origin ในระบบ world coordinate, แกน Xe ไปทางขวา และแกน Ye ขึ้นบน เราสามารถหรับแกน Xe และ Ye ให้ตรงกับแกน Xs และ Ys ของจอภาพได้ (ดูในรูป 4.9) ตามที่กล่าวมาแล้วในบทก่อน ระบบ world coordinate สำหรับจุดแต่ละจุดจะถูกแทนด้วยระบบมือขวา

และเพื่อให้เกิดความสะดวก จึงมีการกำหนดจุดสังเกตุ (viewport) ให้อยู่ในรูปแบบของ Spherical coordinate ซึ่งรูปแบบนี้จะช่วยให้ง่ายในการควบคุมตำแหน่งจุดสังเกตุ (viewer's eye)

โดยการใช้สัญญลักษณ์ต่าง ๆ ที่กล่าวมาแล้วในส่วนที่ 4.1.2 ซึ่งมีค่าต่าง ๆ ดังนี้

D = ระยะทางจากจุดสังเกตุไปยังจุดกำเนิดของระบบ world coordinate ที่มีตำแหน่งวัตถุตั้งอยู่

θ, ϕ = ทิศทางจากผู้สังเกตุที่มองไปยังวัตถุ

ในรูปแบบ Central Projection จุดสังเกตุจะตั้งอยู่บนแกน Ze ซึ่งปรับให้ตรงกับแกนใดแกนหนึ่งของระบบ World coordinate สำหรับตัวอย่างของเราในรูป 4.9 จุดสังเกตุของเราจะปรับให้อยู่ในแกนเดียวกับแกน Z ในกรณีเช่นนี้ระนาบ XeYe จะขนานกับระนาบ XY ทำให้ค่า θ, ϕ มีค่าเป็น 0

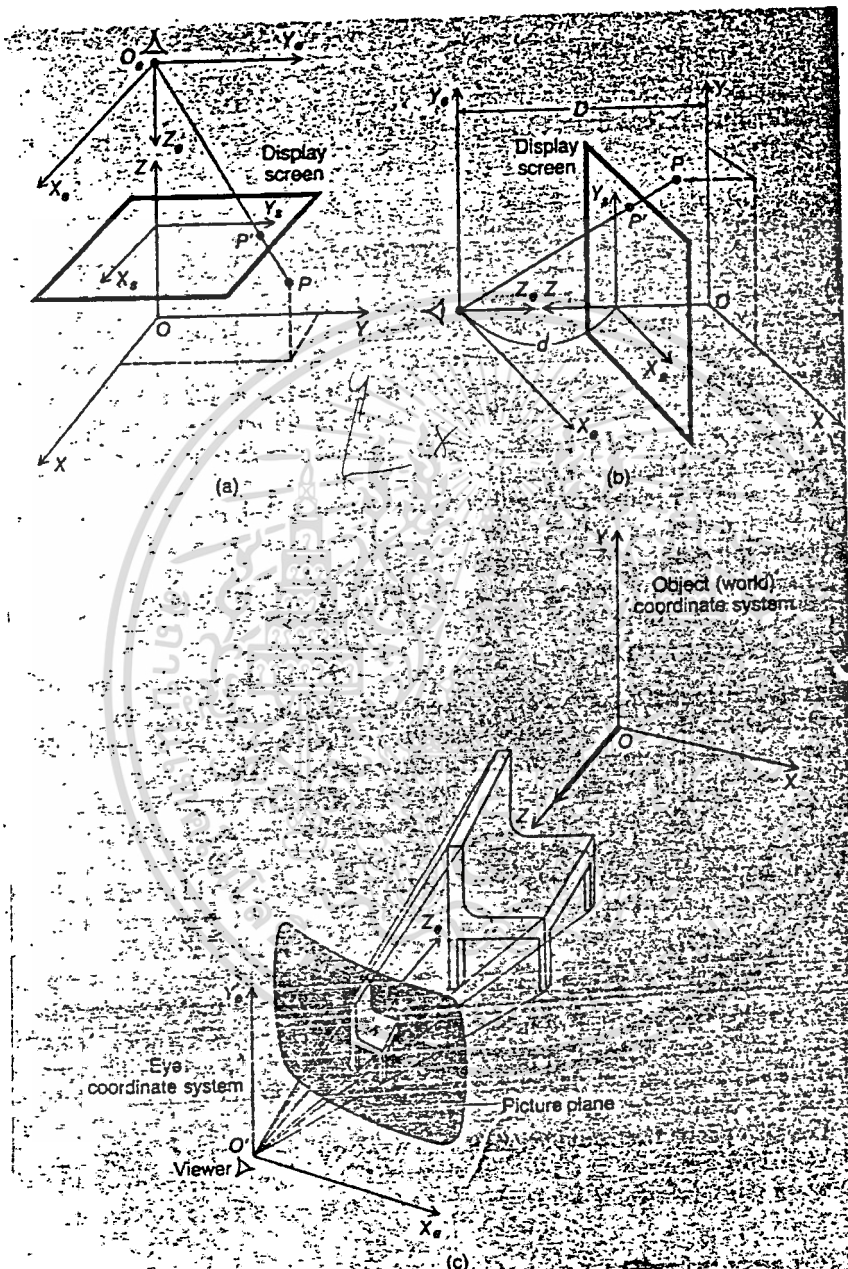


FIGURE 4.9 Central projection of a point onto the image plane (display screen): (a) and (b) are equivalent representations of the central projection and (c) is the central projection of an object.

เพราะฉะนั้นจุดสังเกตุของ Central Projection จะวางอยู่ในตำแหน่ง (D , 270°, 0) ในระบบ Spherical

Viewing Transformation

ในการคำนวณหาตำแหน่งบนจอภาพของจุด ๆ หนึ่งบนภาพ ในลำดับแรกเราต้อง Transform จุดจากระบบ World coordinate ไปเป็นจุดในระบบ eye coordinate โดยที่มีจุด origin อยู่ที่จุดสังเกตุและแกนZe จะต้องชี้ไปยังจุดกำเนิดของระบบ World coordinate

ให้จุด P (X,Y,Z) เป็นจุดในระบบ world coordinate เมื่อเราจะมองจุด P ที่จุดสังเกตุ Oe ค่าของ X และ y จะมีค่าคงเดิมเพราะว่าจุดสังเกตุอยู่ห่างจากระนาบ XY เป็นระยะทาง D,Ze ที่ได้ จึงมีค่า = D-Z เพราะฉะนั้นการเปลี่ยนค่าของ P ไปอยู่ในระบบ eye Coordinate สามารถจะทำได้โดยดังนี้คือ (ดูรูป 4.9)

$$Xe\ Ye\ Ze = (X,Y,D-Z) \qquad (4.23)$$

ค่าที่ได้นี้ได้โดยการสังเกตุ แต่ผลลัพธ์อย่างเดียวกันนี้สามารถจะทำได้โดยใช้ Viewing transformation การเปลี่ยนจุด P ในระบบ world Coordinate ให้เป็นจุดในระบบ eye Coordinate เราจะใช้ขั้นตอน 4 ขั้นตอนคือ

ขั้นตอนที่ 1 ย้ายตำแหน่งจุดกำเนิดไปยังจุดสังเกตุ โดยใช้ coordinate matrix ดังนี้

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

ขั้นตอนที่ 2 กลับทิศทางของแกน Z เพราะฉะนั้น Transformatrix Matrix จะมีค่า

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

ขั้นตอนที่ 3 รวม Transformation Matrix ทั้ง 2 เข้าด้วยกัน

$$T = T_1 T_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ขั้นตอนที่ 4 กำหนดให้อยู่ในรูปแบบ eye Coordinate

$$(Xe,Ye,Ze,1) = (X,Y,Z,1) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.24)$$

$$= (X,Y,D-Z,1)$$

จะเห็นว่าผลลัพธ์ที่ได้จะเหมือนกับสมการ 4.23

Conversion to Screen Coordinate

Screen Coordinate (Xs, Ys) ได้จากการฉายภาพของจุด P ที่วัดในระบบ eye Coordinate จะถูกหาได้โดยการพิจารณาที่ระนาบ YeZe ในรูป 4.10 a

โครงสร้างของ YeZe นี้ได้มาจากรูป 4.9 b และเราจะแทนค่า Parameter p เป็นระยะห่างระหว่างจอภาพกับจุดสังเกตุ และจากรูปจะเห็นได้ว่า สามเหลี่ยม AOeP และ BOeP เป็นสามเหลี่ยมคล้าย ดังนั้นจึงสามารถหาความ

สัมพันธ์ระหว่างสามเหลี่ยมสามเหลี่ยมคล้ายได้ดังนี้คือ

$$\frac{Ys}{d} = \frac{Ye}{Ze}$$

$$Ys = \frac{dYe}{Ze} = \frac{dY}{D-Z} \qquad (4.25)$$

ในการหาค่า Xs เราจะพิจารณาที่รูป 4.10 (b) ซึ่งสร้างมาจากระนาบ XeZe ที่ให้ไว้ในรูป 4.9 (b) และก็พิจารณาได้ในทำนองเดียวกันกับการหาค่า Ys ในที่นี้สามเหลี่ยม AOeE และสามเหลี่ยม BOeF เป็นสามเหลี่ยมคล้าย ดังนั้นเราจึงใช้คุณสมบัติของสามเหลี่ยมคล้ายเพื่อที่จะหาค่า Xs โดยที่ความสัมพันธ์ดังกล่าวคือ

$$\frac{Xs}{d} = \frac{Xe}{Ze}$$

$$Xs = \frac{dxe}{Ze} = \frac{dx}{D-Z} \qquad (4.26)$$

จากสมการ 4.25, 4.26 นั้นจะบอกให้เราทราบว่า

1. Ys และ Xs เป็น Linear funtion ของ d เพราะฉะนั้นการเคลื่อนจอภาพเข้าใกล้ผู้สังเกตุจะทำให้ได้ภาพฉายเล็กลง

2. การเพิ่มค่า D จะเป็นการเคลื่อนย้ายจุดสังเกตุออกจากวัตถุที่ถูกสังเกตุซึ่งจะเป็นผลทำให้วัตถุดูเหมือนว่ามีขนาดเล็กลง เมื่อ d มีค่าคงที่

3. Central Projection ไม่ต้องใช้วิธีการ Viewing Transformation ที่ยุ่งยากเพราะว่าจุดสังเกตุถูกปรับให้เข้ากับแกนใดแกนหนึ่งของระบบ world Coordinate แล้ว

4. การคัดแปลงที่ง่ายไปกว่าเดิมสามารถกระทำได้โดยการเลื่อนจอภาพไปอยู่ตรงระนาบ XY ของระบบ World Coordinate (D = d) และเลื่อนจุดสังเกตุให้ไปอยู่ที่ระยะอนันต์ นั่นคือ D = ∞ และ D = d เราจะได้ผลลัพธ์ของ Xs และ Ys ดังนี้

$$Xs = X \quad , \quad Ys = Y \qquad (4.27)$$

การ project แบบนี้จะถูกเรียกอีกชื่อหนึ่งว่า Orthographic และ Orthographic ก็เป็นรูปแบบหนึ่งของ Parallel Projection ซึ่งเส้นขนานทั้งหมดในวัตถุ 3 มิติจะถูก Transformation เป็นเส้นขนานกับรูปภาพของวัตถุนั้น

4.3.2 Projection Through an Arbritary Viewpoint

ขณะนี้ถ้าเราจะวางจุดสังเกตุที่จุดใด ๆ และยังคงวางที่ตำแหน่งห่างจากจอภาพเป็นระยะทาง d และจอภาพนี้ยังถูกสมมุติว่าจะต้องตั้งฉากกันเส้นสายตาของผู้สังเกตุที่มองมายังวัตถุที่จุด origin ดังนั้นระบบ eye coordinate ก็ยังคงต้องมีการหันเหแกนเพื่อว่าให้เส้นสายตาของผู้สังเกตุ (แกน Ze) ชี้ตรงไปยังจุด origin ของระบบ world coordinate และในทำนองเดียวกันแกน X ต้องชี้ไปทางขวาและแกน Y จะต้องชี้ขึ้นด้านบน รูป 4.11 แสดงให้เห็นถึงความแตกต่างระหว่าง Central Projection และการ projection ผ่านจุดใดจุดหนึ่ง

ในการที่จะกำหนด Screen eye Coordinate (Xs,Ys) ให้คล้องจองกับจุด P(X,Y,Z) เราก็สามารถกระทำได้ในทำนองเดียวกันกับวิธีการใน Central Projection คือจะต้องแบ่งออกเป็น 2 ส่วนคือ

ส่วนที่ 1 ต้องมีการกำหนด eye coordinate (Xe,Ye,Ze) ให้เหมาะสมกับจุดที่ตั้งอยู่ในระบบ World Coordinate (X,Y,Z)

ส่วนที่ 2 ต้องหาค่า Screen Coordinate (Xs,Ys) จากระบบ eye Coordinate (Xe,Ye,Ze) ที่ได้ในส่วนที่ 1

Viewing Transformation

การเปลี่ยนแปลงระบบ World Coordinate ไปเป็นระบบ eye coordinate จะต้องใช้ลำดับ Viewing Transfomation ในแต่ละ Viewing Transformation จะถูกทำสำเร็จโดยการ Transform ระบบ Coordinate นี่ย่อมหมายความว่าตำแหน่งที่ตั้งของจุด P จะไม่มีการเปลี่ยนแปลง แต่จุดสังเกตุของเราจะถูกเปลี่ยนแปลงไปเป็นระบบ eye coordinate ดังที่กล่าวมาแล้วในส่วน

4.2.6 การ Transform ระบบ Coordinate ก็คือการ Inverse Transformation ของการเคลื่อนที่จุดนั้น ๆ เราจะใช้ Viewing Transfomation ซึ่งผลลัพธ์ที่มองเห็นจะได้รับมาจากลำดับของระบบ Coordinate ใหม่จนกระทั่งได้ eye Coordinate system เป็นผลลัพธ์สุด

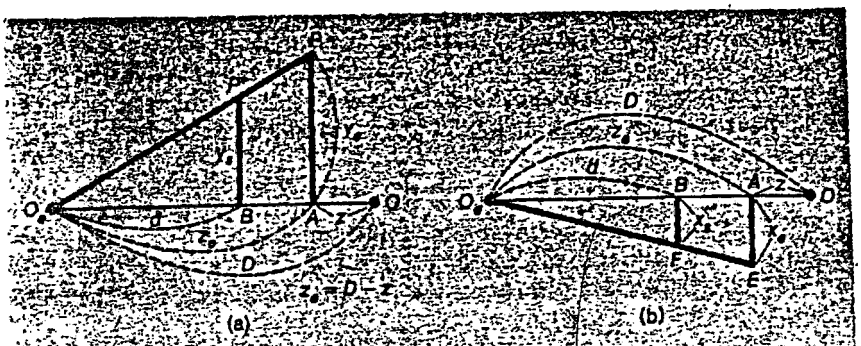


FIGURE 4.10 Relations among the projection parameters

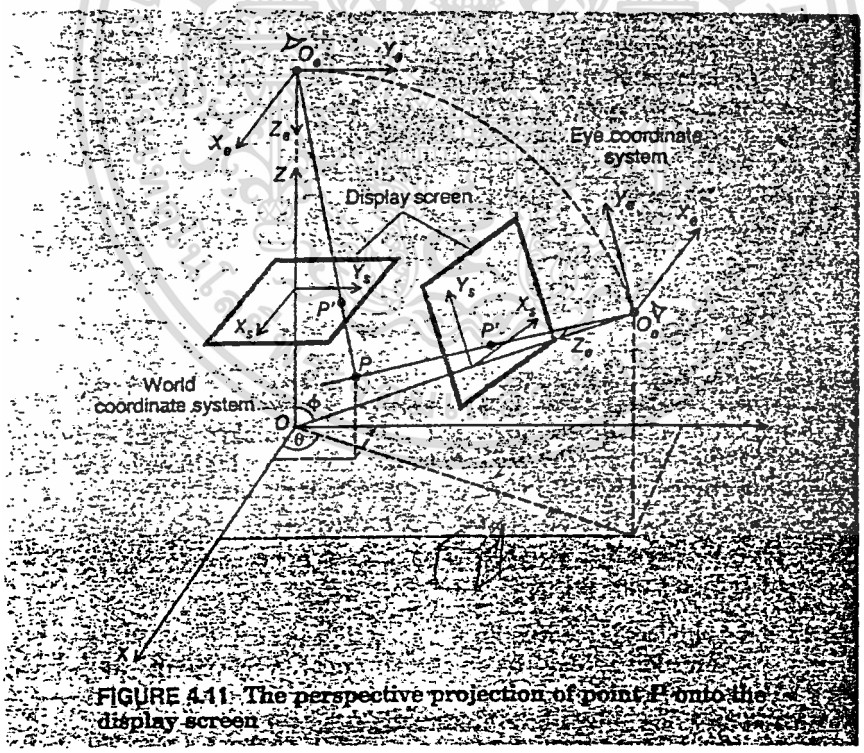


FIGURE 4.11 The perspective projection of point P onto the display screen

ท้าย เพื่อให้เกิดความสะดวกเราจะอ้างถึงแกนใดที่เกิดขึ้นใหม่จะให้เป็นแกน X',Y',Z' และระบบแกน X' Y' Z' ที่ได้มาหลังสุดก็จะเป็นระบบ eye Coordinate ผู้อ่านต้องเข้าใจว่าความแตกต่างระหว่างการ Transformation ของจุดใดกับการ Transformation ของแกน Coordinate โดยที่รูปแบบแรกนั้นตำแหน่งของจุดจะเปลี่ยนไปในขณะที่แบบหลังตำแหน่งของจุดไม่เปลี่ยนต่อไปให้เราเริ่มพิจารณาการ Viewing Transformation ลำดับแรก เราจะให้จุดสังเกตุอยู่ที่ Oe (ดังในรูป 4.11) ที่อ้างถึงโดยระบบ Spherical Coordinate (D, θ, ∅) จากสมการ 4.1 เรารู้ว่า (X = D sin∅cosθ, Y = D sin∅ sinθ, Z = Dcos∅) เมื่อค่า D,θ,∅ นั้นจะต้องถูกกำหนดขึ้นมาก่อน แล้วต่อมาเราก็จะทำการ Transformation ตามลำดับขั้นตอนดังนี้

ขั้นตอนที่ 1

ย้ายจุดกำหนดไปยังตำแหน่ง Oe เพื่อที่จะสร้างระบบ Coordinate ใหม่ที่จุดสังเกตุ (ดูในรูป 4.12) Transformation Matrix ที่ต้องใช้คือ

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -D\sin\emptyset\cos\theta & -D\sin\emptyset\sin\theta & D\cos\emptyset & 1 \end{bmatrix}$$

ขั้นตอนที่ 2

หมุนระบบแกนของแกน Z ตามเข็มนาฬิกาเป็นมุม 90− θ การหมุนนี้จะทำให้แกน Y ผ่านแกน Z (ดูในรูป 4.12 (b) Transformation Matrix ก็ยังคงใช้ผลการหมุนจุดรอบแกน Z ในทิศทางทวนเข็มนาฬิกาเป็นมุม 90−θ และเรารู้ว่า sin(90−θ) = cosθ และ cos(90−θ) = sinθ และโดยการใช้ Matrix (4.10) เราจะได้ Transformation Matrix ดังนี้คือ

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

ขั้นตอนที่ 3

หมุนโดยระบบแกน X ในทิศทางทวนเข็มนาฬิกาเป็นมุม 180−ø เพื่อที่จะทำให้แกน X ชี้ไปยังจุดกำเนิดของระบบ World Coordinate (ดูในรูป 4.12 (c) การ Transform นี้จะมีรูปแบบเหมือนกับการหมุนจุดรอบแกน Xในทิศทางตามเข็มนาฬิกาเป็นมุม 180−ø เนื่องจาก sin (180−ø) =sinø และ cos (180−ø)= -cosø ดังนั้นจะได้ Transform Matrix T_3 มีค่าดังนี้

$$T_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ขั้นตอนที่ 4

เปลี่ยนให้เป็นระบบมือซ้าย(left hand system) โดยกลับทิศทางของแกน x เพื่อให้ได้แกนทั้ง 3 เข้าสู่รูปแบบของระบบ eye coordinate (ดูในรูป 4.12 (d) ดังนั้น Transformation Matrix T_4 จะมีค่า

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

ขั้นตอนทั้ง 4 ขั้นตอนนี้เป็นการ Transformation ที่ต้องนำมาใช้ในการสร้าง Viewing Transform นั้นคือ

$$T = T_1 \cdot T_2 \cdot T_3 \cdot T_4$$

$$= \begin{bmatrix} -\sin\theta & -\cos\theta\cos\phi & -\cos\theta\sin\phi & 0 \\ \cos\theta & -\sin\theta\cos\phi & -\sin\theta\sin\phi & 0 \\ 0 & \sin\phi & -\cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.28)$$

เพราะฉะนั้นถ้าเรารู้ค่าจุด P ในระบบ World Coordinate แล้วเราก็สามารถที่จะหา Coordinate (Xe,Ye,Ze) ของระบบ eye ได้ดังนี้คือ

$$Xe\ Ye\ Ze\ 1 = X\ Y\ Z\ 1 \cdot T \quad (4.29)$$

$$Xe = -X\sin\theta - Y\cos\theta$$

$$Ye = -X\cos\theta\cos\phi - Y\sin\theta\cos\phi - Z\sin\phi$$

$$Ze = -X\cos\theta\sin\phi - Y\sin\theta\sin\phi - Z\cos\phi + D$$

หลังจากที่เราได้ eye coordinate แล้ว เราก็สามารถที่จะคำนวณหา screen coordinate(Xs,Ys)โดยใช้สมการ 4.25 และ 4.26

$$Xs = dxe/Ze \ , \ Ys = dye/Ze \quad (4.30)$$

และสุดท้ายเราจะแสดงให้เห็น Transformation Matrix ของระบบ Central Projection (สมการ 4.24) ก็คือรูปแบบหนึ่งของสมการ (4.29) เมื่อค่า θ และ ϕ vD ของระบบ central projection มีค่าเป็น 270°,0,0 ตามลำดับเมื่อแทนค่าเหล่านี้ลงในสมการ 4.28 จะทำให้ได้T มีค่าดังนี้

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

ซึ่งผลลัพธ์ที่ได้ก็คือการ Transformation Matrix ของระบบ Central Projection นั่นเอง

<u>ความสัมพันธ์ระหว่าง Parameter ต่าง ๆ</u>

เราจำเป็นต้องรู้ว่าค่า Viewing Parameter d θ ϕ D มีผลต่อการกำเนิดภาพบนจอภาพอย่างไร

(9 7)

– ถ้ามีการเปลี่ยนแปลงค่าของ θ และ ϕ จะทำให้เราสามารถมองเห็นวัตถุในตำแหน่งมุมที่ต่าง ๆ กัน

– ถ้ามีการเปลี่ยนแปลง D (ระยะทางจากจุดสังเกตุไปยังจุด origin)จะทำให้เราสามารถเลื่อนจุดสังเกตุเข้าใกล้ภาพหรือห่างจากภาพได้ (ดูในรูป 4.10)

– เมื่อจุดสังเกตุ (D,θ,ϕ) ตั้งไว้ในตำแหน่งที่และมีการเปลี่ยนแปลงค่า d จะมีผลต่อขนาดของภาพบนจอภาพ ถ้าจอภาพอยู่ใกล้จุดสังเกตุ (d มีค่าลดลง) ขนาดของภาพที่ได้จะมีขนาดเล็กลง (ดูในสมการ 4.15, 4.26, 4.10)

– เนื่องจาก D และ d ใช้ในการควบคุมขนาดของภาพ เราต้องมีการสร้างความสัมพันธ์ระหว่างค่าทั้งสองให้เหมาะสม เช่น เมื่อวัตถุอยู่ใกล้จุดสังเกตุวัตถุจะดูเหมือนมีขนาดใหญ่ขึ้นกว่าวัตถุที่อยู่ไกลออกไป เพราะว่าวัตถุนั้นจะครอบคลุมเนื้อที่การมองอย่างมากมาย สำหรับตัวอย่าง เช่น ภาพ Perspective ของลูกบาศก์ ด้านที่อยู่ใกล้จุดสังเกตุจะมีขนาดใหญ่กว่าด้านที่อยู่ไกลออกไป เพื่อที่จะลดผลที่เกิดขึ้นของการมองภาพแบบ Perspective เราอาจจะเพิ่มค่า D เพื่อใช้ในการลดขนาดของภาพที่มองเห็น ส่วนในการลดขนาดของภาพอย่างมากมาย โดยไม่มีผลต่อ Perspective อาจทำด้วยเพิ่มค่า d

4.3.3 Homogenous Screen Coordinate System

แทนที่เราจะเปลี่ยน Parameter ของ D และd ในการควบคุมขนาดของภาพเราสามารถเปลี่ยนค่า Xs และ Ys ให้เป็นเศษส่วนที่มีค่าน้อยโดยการหารค่า Xs และ Ys ด้วยขนาดศูนย์กลางของจอภาพ (เมื่อจุด origin ถูกตั้งอยู่ที่ตำแหน่งศูนย์กลางของ screen) ผลลัพธ์ที่ได้จะทำให้เกิดระบบ screen coordihate ใหม่มีค่าดังนี้คือ

$$Xs = dxe/sze \qquad (4.31)$$

$$Ys = dye/sze \qquad (4.32)$$

Viewing Pyramid ในรูป 4.13 แสดงให้เห็นเนื้อที่ของระบบ eye Coordinate ที่วัตถุต้องตั้งอยู่เนื้อที่นี้ ถ้าวัตถุนั้นอยู่ในสภาวะที่มองเห็น สำหรับจุดสังเกตุที่คงที่มีเพียงค่าอัตราส่วน $\frac{d}{s}$ เท่านั้นที่จะควบคุมการฉายภาพลงบนจอ

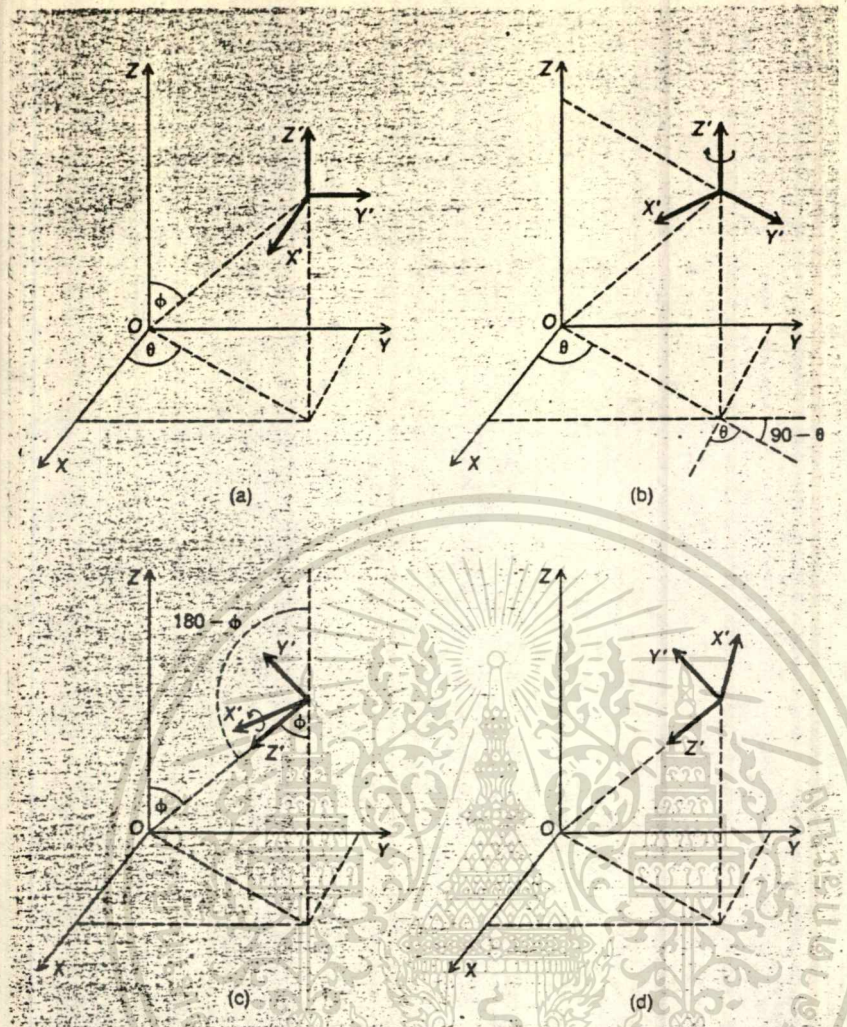
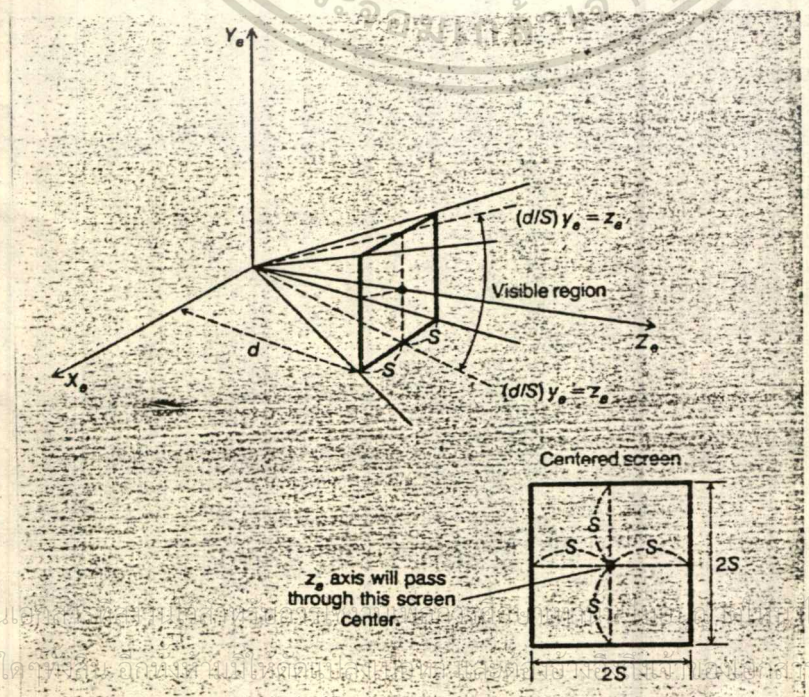


FIGURE 4.12 Four steps in establishing the viewing transformation from the world coordinates to the eye coordinates



ภาพซึ่งผลอันนี้ก็มีลักษณะคล้ายคลึงกับระยะทางจากจุดศูนย์กลางไปยังจุดรวมแสงของเลนส์กล้องที่เป็นตัวกำหนดค่าต้องการภาพเล็กใหญ่ขนาดเท่าไหร ถ้าอัตราส่วนนี้มีค่าน้อยมันก็จะมีลักษณะเหมือนกับการใช้ focal length สั้นในการถ่ายภาพ ภาพที่ได้จึงมีลักษณะคล้ายกับถ่ายด้วย len มุมกว้างในทำนองเดียวกัน ถ้าค่าอัตราส่วนของ dis มีค่ามากก็เหมือนเราปรับ focal length ยาวก็จะมีผลเหมือนกล้องส่องทางไกล

และมีสิ่งสำคัญอีกอย่างหนึ่งที่ควรจะไว้ว่า การหารด้วยค่า Screen Coordinate ด้วย S จะหาค่า Xs,Ys อยู่ในช่วงดังนี้คือ

$$-1 \leq Xs \leq 1 \qquad (4.33)$$

$$-1 \leq Ys \leq 1 \qquad (4.34)$$

4.4 Perspective Depth

ในส่วนนี้จะกล่าวถึงรูปแบบที่สำคัญของ Computer graphic ในการกำเนิดภาพของวัตถุ 3 มิติ ให้มีลักษณะเหมือนธรรมชาติ ความลึกของจุด (วัตถุ) จะเป็นพื้นฐานในการสร้าภาพให้เหมือนจริงในส่วนนี้เราจะแนะนำคณิตศาสตร์ที่จำเป็นในการใช้หาความลึก และเราจะใช้ค่าความลึกที่ได้นี้ในการเคลื่อนย้ายเส้นที่ถูกบังออกไปและยังรวมถึงการสร้าง 3-D clipping Algorithm และการสราง Algorithm สำหรับเส้นที่ถูกบังหรือผิวที่ถูกบังในบทที่ 5

4.4.1 Depth Measure for a Pavallel Projection

ความลึกของแต่ละจุดในภาพที่เราสร้างอยู่บนจอภาพจำเป็นที่ต้องรู้ค่าด้วย เพื่อที่จะเป็นตัวกำหนดว่าพื้นผิวใด ๆ บังเส้นหรือพื้นผิวอื่นหรือไม่ ? การเปรียบเทียบค่าความลึกแบบง่ายก็คงมีลักษณะเหมือนกับคำถามต่อไปนี้ ให้จุด $P_1 = (X_1, Y_1, Z_1)$ และ $P_2 = (X_2, Y_2, Z_2)$ และจุดทั้งสองบังกันหรือไม่ ? คำถามนี้สามารถตอบได้ดังนี้คือ ต้องเปรียบเทียบจุด P_1 และ P_2 อยู่บน Project เดียวกันหรือไม่ (เส้นทางการมองจากจุดสังเกตุ) ถ้าคำตอบตอบว่าใช่ ก็ต้องเปรียบเทียบค่า Z_1 และ Z_2 และการเปรียบเทียบนี้จะเป็นตัวบอกให้เรารู้ว่าจุดไหนอยู่ใกล้จุดสังเกตุมากกว่าอีกจุดหนึ่ง ถ้าคำตอบในส่วนแรกตอบว่าไม่ใช่ก็จะไม่มีการบังกันเกิดขึ้นก่อนที่เราจะแนะนำการคำนวณหาค่าความลึก เราจะใช้

Viewing Transformation ก่อนที่ได้ให้ไว้ในส่วน 4.3.2 โดยที่เราจะใช้วิธีการ 2 ขั้นตอนในการหาค่า Screen Coordinate โดยที่ลำดับแรกเราจะ Transform จากระบบ World Coordinate ไปเป็น eye Coordinate และต่อจากนั้นเราก็เปลี่ยนจากระบบ eye Coordinate เป็น Screen Coordinate

$$(X,Y,Z) \longrightarrow (Xe,Ye,Ze) \longrightarrow (Xs,Ys)$$

orthographic Projection สามารถที่จะมองเห็นในรูปแบบของ central projection (ดูในส่วน 4.3.1) โดยการให้จุดสังเกตุตั้งอยู่ที่ระยะอนันต์ของแกน Z ซึ่งในลักษณะนี้ ถ้า X = Xe Y=Ye และZe =D-Z การเปรียบเทียบความลึกก็จะกระทำได้ง่าย เพราะ coordinate ของ Z เพียงตัวเดียวก็สามารถที่จะกำหนดความลึกของแต่ละจุดได้ เนื่องจากค่าของ Z,Ze มีค่าเกี่ยวข้องกันในลักษณะที่ตรงกันข้าม ดังนั้น Ze ยิ่งน้อยเท่าไร จุดนั้นก็จะอยู่ใกล้จุดสังเกตุมากขึ้น

อย่างไรก็ตามสำหรับใน Perspective Projection การเปรียบเทียบความลึกนั้นค่อนข้างจะยุ่งยากกว่าเพราะว่าการ Project จุดแต่ละจุดลงบนจอภาพจะต้องเกี่ยวข้องกับทิศทางการมองจากจุดสังเกตุที่มุมต่าง ๆ ดังนั้น Viewing Transform จะทำให้ภาพของวัตถุผิดเพี้ยนไป เพื่อที่จะทำให้ส่วนของวัตถุมีขนาดเล็กลงเมื่อวัตถุนั้นอยู่ไกลออกไป การเปรียบเทียบความลึกของ Ze ต้องมีการดัดแปลงแก้ไขใหม่

4.4.2 Depth Measure for a perspective projection

ในลำดับแรกเราจะขยายระบบจาก Screen Coordinate 2 มิติ (Xs,Ys) ให้เป็นระบบ 3 มิติ (Xs,Ys,Zs) ในระบบ Screen Coordinate ที่เพิ่มเติมมานี้จะต้องคำนวณหาค่า Zs เพื่อที่จะทำให้ทราบค่าความลึกและการคำนวณนี้จะต้องไม่ทำให้ค่า Xs และ Ys เดิมเปลี่ยนไป ถ้าเราต้องการที่จะแสดงจุด Xs Ys Zs ลงบนจอภาพเราก็สามารถใช้เพียงค่าของ Xs และ Ys โดยตรงโดยตัดค่า Zs coordinate ทิ้งไป

ในลำดับที่ 2 เราก็จะใช้เส้นตรงในระบบeye coordinate เปลี่ยนเป็นเส้นตรงในระบบ Screen coordinate ซึ่งลักษณะนี้สามารถทำได้โดยการ Transform จุดปลายทั้งสองของเส้นตรงเส้นนั้นและวาดเส้นระหว่างจุดปลายทั้งสองที่ถูก Transform นี้ สำหรับระบบ eye coordinate ก็ยังคงใช้คุณสมบัติของการ Transform พื้นฐานมาใช้อยู่ นั่นคือระนาบในระบบ eye coordinate สามารถเปลี่ยนไปเป็นระนาบในระบบ Screen coordinate เพราะฉะนั้นเราอาจจะ Transform ค่า Ze โดยตลอดค่าของ Xe,Ye เพื่อที่จะทำให้ได้สมการระนาบในระบบ eye coordinate Transform ไปเป็นสมการระนาบในระบบ Screen Coordinate

$$AXe + BYe + CZe + D = 0 \quad (4.35)$$

$$\longrightarrow \quad A'Xs + B'Ys + C'Zs + D' = 0 \quad (4.36)$$

เราสามารถกำหนดตัวสัมประสิทธิ์ที่ไม่รู้ค่าในสมการระนาบได้ ถ้าเรารู้จุดสามจุดบนระนาบนั้น ๆ สำหรับ Perspective Projection เราจะหาค่า Xs และ Ys ได้ดังนี้

$$Xs = dXe / sZe \quad , \quad Ys = dYe / sZe$$

เมื่อ d คือระยะทางจากจุดสังเกตุถึงจอภาพและเมื่อหนค่า Xs และ Ys ลงในสมการ (4.36) ก็จะหาค่า Zs ได้โดย Zs จะมีค่า

$$Zs = \alpha - \beta/Ze \quad (4.37)$$

และ $\alpha = -D'/C'$

$\beta = (d/c's)(A'Xe + B'Ye)$

และเนื่องจากเราไม่สามารถรู้ได้ว่าสมการระนาบจะมาในรูปแบบใด เราจึงต้องใช้วิธีการของ Newman และ sproll ในการเลือกค่า α และ β

เราจำกัดขอบเขตสิ้นสุดการมองโดยการกำหนดค่า Ze ให้อยู่ในช่วง $K_1 \leq Ze \leq K_2$ ซึ่งจะทำให้ได้ Trancated Viewing Pyramid ในระบบ eye coordinate ดังรูป 4.14

เราก็จะประมาณให้ค่า $Ze = K_1$ ที่ทำให้ค่าของ Zs นั้นมีค่าน้อยที่สุด และค่า $Ze = K_2$ ที่ทำให้ K_2 มีค่ามากที่สุด ค่าต่าง ๆ ที่อยู่นอกเหนือย่านดังกล่าวนี้จะแสดงในระนาบ Scren coordinate และเพื่อให้เป็นรูปแบบเราจะกำหนด

ค่าของ Ze ให้อยู่ในช่วง 0 ถึง 1

– เพราะว่า Zs =(0→Ze)= K_1 และ Zs =(1→Ze) = K_2 จาก เงื่อนไขนี้ทำให้เราได้สมการเชิงเส้นมา 2 สมการคือ

$$0 = \alpha - \frac{\beta}{K_1}$$

$$1 = \alpha - \frac{\beta}{K_2}$$

และจะทำการ solve เพื่อหาค่าของ α และ β จากสมการนี้จะหาค่า α และ β ได้ดังนี้

$$\alpha = \frac{K_2}{K_2 - K_1}, \quad \beta = \frac{K_1 K_2}{K_2 - K_1}$$

– ต่อไปก็หาค่า Zs โดยการแทนค่า α และ β ลงในสมการ (4.37)

$$Zs = K_2(Ze-K_1)/(K_2-K_1)\ Ze \quad , \quad 0 \le Ze \le 1 \qquad (4.38)$$

– เมื่อนำจุดทั้ง 2 จุดมาเปรียบเทียบความลึก, จุดที่มีค่า Zs น้อยจุดนั้นจะอยู่ใกล้จุดสังเกตุมากกว่าจุดที่มี Zs มาก

4.5 Viewport Planning for 3D graphic

การออกแบบช่วงการมองของระบบ 3 มิติ นั้นจะเกี่ยวข้องกับการ clip ส่วนของเส้นตรงที่ตัดกับขอบเขตสิ้นสุดของเนื้อที่การมองที่กำหนดไว้ กรรมวิธีการ clipนั้นจะต้องแน่ใจว่าส่วนที่มองเห็นของภาพถูก Transform เป็นกลุ่มคำสั่ง plot ให้แก่อุปกรณ์จอภาพ ขอบเขตการclip ของภาพ 3 มิติ ก็คือพื้นผิวแต่ละด้านของ Viewing Pyramid (ดูในรูป 4.14)

4.5.1 Three Dimemsion Clipping

ถ้าเราต้องการกำหนดขอจำกัดของเนื้อที่การมอง, เราสามารถกระทำได้โดยการกำหนดระนาบส่วนหน้าและระนาบส่วนหลัง ดังเช่นที่แสดงไว้แล้วในรูป 4.14 ระนาบการ clip ทั้งสองนี้จะขนานกับระนาบแสดงผล display Plane และ ระนาบทั้งสอง ถูกกำหนดโดยระยะทางจากระนาบการมองปกติจากจุดการมองที่ตั้งไว้ ในทำนองเดียวกัน ระนาบเหล่านี้จะต้องตั้งฉากกับเส้นสายตา (แกน Z)

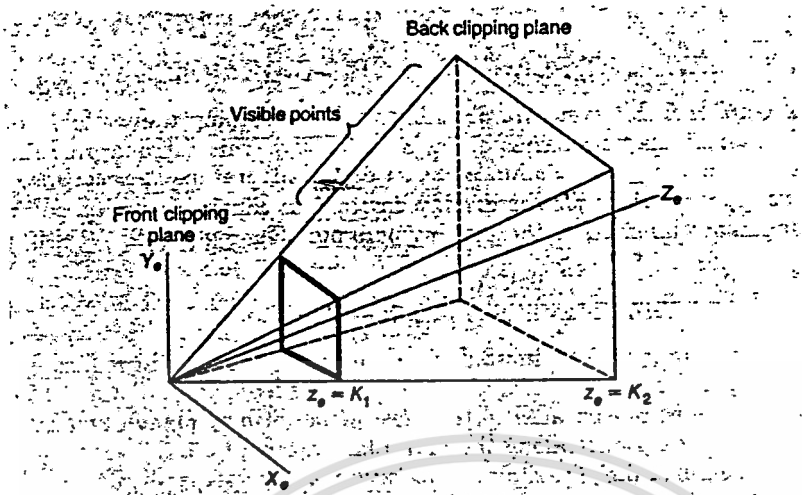


FIGURE 4.14 Truncated view volume in the eye coordinate system. (Visible points must have $K_1 \leq z_e \leq K_2$.)

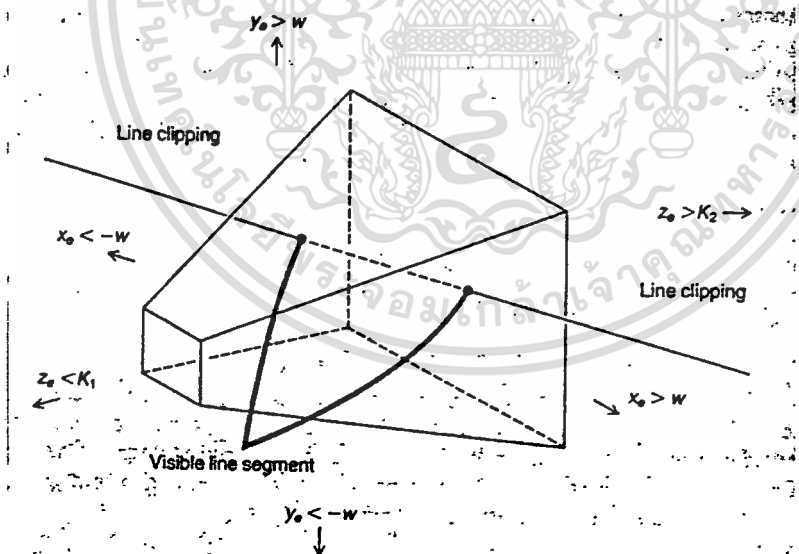


FIGURE 4.15 Three-dimensional clipping. Lines that pass through the view volume must be clipped before the perspective division is performed.

Point clipping

เงื่อนไขที่ต้องพบทั้ง 3 อย่างในจุดแต่ละจุดที่มองเห็นภายใน Transcate Pyramid โดยจากสมการ 4.31 ถึง (4.34) เราจะได้เงื่อนไขทั้ง 3 คือ

$$-1 \leq Xs \leq 1 \rightarrow -Ze \leq dXe/s \leq Ze \rightarrow -w \leq Xe \leq w \quad , \qquad (4.39)$$

$$-1 \leq Ys \leq 1 \rightarrow -Ze \leq dYe/s \leq Ze \rightarrow -w \leq Ye \leq w \quad , \qquad (4.40)$$

เมื่อ $w = \frac{s}{d} Ze$ และ $w > 0$

และจากสมการ (4.38) เราจะได้ช่วงการ clip อีก 1 อย่างคือ

$$0 \leq Ze \leq 1 \longrightarrow K_1 \leq Ze \leq K_2 \qquad (4.41)$$

และถ้าจุดใดไม่เข้าเงื่อไขเหล่านี้ จุดนั้นจะมองไม่เห็นและไม่นำออกสู่จอภาพ ถ้าจุดนั้นอยู่ใน 3 เงื่อนไขดังกล่าว ต่อมาก็จะใช้สมการ (4.31, (4.32) ก็จะถูกนำมาใช้เปลี่ยนแปลงระบบ eye coordinate เพื่อที่จะกำหนดต่าง ๆ บนจอภาพต่อไป ผลที่ได้ของสมการ (4.31) (4.32), 4.38) จะต้อง map ให้เข้ากับ Transated Pyramid และให้อยู่ในรูปแบบช่วงการมองแบบมาตรฐานในระบบ Screen Coordinate

Line Clipping

ตามที่เห็นมาแล้วใน 2D Clipping เส้นใด ๆ ไม่สามารถจะดำเนินวิธีการ clip ได้ง่ายเหมือนจุด เส้นจะต้องถูกตัดออกเมื่อถึงขอบเขตสิ้นสุดของ Viewing Pyramid ที่ให้นิยามไว้ในสมการ 4.39 ถึง 4.41 รูป 4.15 นั้นจะแสดงให้เห็นเนื้อที่การมองที่จำกัดไว้ซึ่งได้รับมาจากสมการ (4.39) ถึง (4.41)

เราสามารถที่จะสร้าง Algorithm การ clip ของ 3D ได้โดยการขยายเพิ่มเติมจากรูปแบบ 2D ที่ได้อธิบายไว้แล้วในส่วน 3.5.1 Algorithm นี้จะกำหนดว่าจุดปลายทั้งสองของเส้นที่อยู่ภายนอกขอบเขตสิ้นสุดของเนื้อที่การมอง โดยที่จุด 1 จุดจะนำมาคำนวณหาค่าสภาวะบิท code 6 บิท เพื่อกำหนดสภาวะของเส้นจากบิท code เหล่านี้ สำหรับจุดปลายใด ๆ เราสามารถกำหนดได้ว่า

I(I) = จุดที่อยู่เหนือเนื้อที่การมอง, Ye > w

B(I) = จุดที่อยู่ต่ำกว่าเนื้อที่การมอง, Ye < -w

R(I) = จุดที่อยู่ทางด้านขวาเนื้อที่การมอง, Xe > w

L(I) = จุดที่อยู่ทางด้านซ้ายของเนื้อที่การมอง, Xe < -w

BH(I) = จุดที่อยู่ทางด้านหลังของเนื้อที่การมอง, Ze > K_2

F(I) = จุดที่อยู่ทางด้านหน้าของเนื้อที่การมอง, Ze < K_1

ถ้าค่าของจุดเข้าตามเงื่อนไขใดก็จะปรับค่า Bit code นั้นให้มีค่าเป็น 1 ถ้านอกเหนือจากที่กำหนดไว้จะปรับค่า Bit code นั้นให้เป็น 0

เราจะพิจารณาเหตุการณ์ 3 กรณีต่อไปนี้คือ

<u>กรณีที่ 1</u> ถ้าเราสังเกตุว่า

$$T(1) + B(1) + R(1) + L(1) + BH(1) + F(1) + T(2) + B(2) + L(2) + BH(2) + F(2) = 0 \quad (4.42)$$

ในกรณีนี้จุดทั้งสองจะอยู่ภายในเนื้อที่การมองและเราก็ยอมรับเส้นที่ต่อจุดทั้ง 2 นี้เป็นส่วนที่มองเห็น

<u>กรณีที่ 2</u> ถ้าเราสังเกตุว่า

$$T(1).T(2)+R(1).R(2)+L(1).L(2)+B(1).B(2)+BH(1).BH(2)+F(1).F(2) \neq 0 \quad (4.43)$$

ในกรณีจุดทั้งสองวางอยู่บนด้านที่มองไม่เห็นของระนาบใดระนาบหนึ่งและเราจะต้องคัดเส้นนี้ทิ้งไป

<u>กรณีที่ 3</u> ถ้านอกเหนือจากกรณีทั้ง 2 แล้วสภาวะของเส้นไม่สามารถกำหนดได้ในทันที ในกรณีเช่นนี้เราจำเป็นต้องคำนวณหาจุดตัดของเส้นกับระนาบทั้งหมดของเนื้อที่การมอง

โดยการใช้ Parametric representation ของเส้นตรง ซึ่งกำหนดโดยจุดปลายทั้งสองของเส้นตรงนี้ในระบบ eye coordinate เราสามารถกำหนดจุดตัดโดยง่ายระหว่างเส้นกับระนาบใดระนาบหนึ่งของเนื้อที่การมอง ถ้าเราให้จุด P_1 (Xe_1, Ye_1, Ze_1) และ P_2 (Xe_2, Ye_2, Ze_2) เป็น eye coordinate ของจุดปลายทั้งสอง เราจะได้สมการ Parametric คือ

$$Xe = (Xe_2 - Xe_1)\ t + Xe_1$$
$$Ye = (Ye_2 - Ye_1)\ t + Ye_1$$
$$Ze = (Ze_2 - Ze_1)\ t + Ze_1 \qquad (4.44)$$

และ t จะอยู่ในช่วง 0 ถึง 1 และเมื่อt มีการเปลี่ยนแปลงจาก 0 ถึง 1 สมการทั้ง 3 นี้จะให้ Coordinate ของจุดทั้งหมดบนเส้นตรงเส้นนั้น

ในการคำนวณหาจุดตัดของเส้นกับระนาบด้านบนของเนื้อที่การมองเราจะแทนค่าตัวแปร Ye ด้วยค่าของเขตสิ้นสุดด้านบนด้วย w และแก้สมการหาค่า t สมการดังกล่าวถือ

$$(Ye_2 - Ye_1)\ t + Ye_1$$
$$= (s/d)\ Ze \qquad \text{ดูสมการ } 4.40$$
$$= \left(\frac{s}{d}\right)(Ze_2 - Ze_1)t - Ze_1 \qquad \text{ดูสมการ } 4.44$$

$$t = \frac{(s/d)\ Ze_1 - Ye_1}{(Ye_2 - Ye_1) - (s/d)(Ze_2 - Ze_1)} \qquad (4.45)$$

ถ้า t อยู่ในช่วง 0 ถึง 1 ก็จะมีการตัดกันภายนอกเนื้อที่การมอง และถ้า t อยู่ในช่วง 0 ถึง 1 แสดงว่ามีการตัดที่ระนาบหนึ่งระนาบใดของเนื้อที่การมอง ดังนั้นเราจะแทนค่า t ลงในสมการ 4.44 ทำให้ได้จุดที่มีการตัดและการคำนวณหาจุดตัดกับระนาบอื่นก็คือในวิธีเดียวกันกับข้างบนนี้ และจะมีการลองเพียงครั้งเดียวที่เราจะครอบคลุมการคำนวณค่าeye coordinate ของจุดทั้งหมดที่จะต้องถูกตัดทิ้งไป clipping test อย่างไรก็ตามการ clipping ในระบบ eye coordinate นั้นจะทำได้ง่ายกว่าในระบบ World Coordinate

หลังจากที่เราหาจุดตัดได้แล้ว เราจะใช้จุดตัดนี้แทนจุดปลายของด้านที่มองไม่เห็นของระนาบนั้น ต่อจากนั้นเราก็คำนวณหา Screen Coordinate ของจุดปลายที่ทำการ clip แล้ว

ตัวอย่าง เราจะพิจารณาที่เส้นตรงที่มีจุดปลายในระบบ eye coordinate คือ $P_1(Xe_1, Ye_1, Ze_1) = (1, 6, 2)$ และ $P_2(Xe_2, Ye_2, Ze_2) = (4, -2, 5)$ เพื่อให้ง่ายขึ้นเราจะสมมุติว่า s=d (s/d=1) และ 4 Ze 10 สำหรับจุด P_1 เราคำนวณหาค่า $w = (s/d)Ze = 2$ สำหรับจุด P_2 เราคำนวณ $w = (s/d)Ze_2 = 5$

ต่อจากนั้นก็คำนวณหา 6 Bit code ของแต่ละจุดได้ดังนี้

T(1)	= 1	T(2)	= 0
B(1)	= 0	B(2)	= 0
R(1)	= 0	R(2)	= 0
L(1)	= 0	L(2)	= 0
BH(1)	= 0	BH(2)	= 0
F(1)	= 0	F(2)	= 0

จากข้างบนนี้เราก็สามารถรู้ว่าจุด P_1 อยู่ภายนอกเนื้อที่การมอง แต่ P_2 อยู่ภายในเนื้อที่การมองซึ่งแสดงว่าจะต้องมีจุดตัดเพียงจุดเดียวเท่านั้นที่ต้องคำนวณหา (สังเกตุว่า ต้องไม่ใช่เงื่อนไขในสมการ 4-42 และ 4-43) ในลำดับแรก เราจะต้องเช็คจุดตัดกับระนาบบนของเนื้อที่การมอง จากสมการ (4.45) เราคำนวณหาค่า t= 4/11 และเนื่องจาก t อยู่ในช่วง 0 ถึง 1 ระนาบบนนี้จะมีจุดตัดเกิดขึ้นจากสมการ 4.44 และสามารถคำนวณหา Coordinate ของจุดตัดได้ดังนี้

$$Xe = (4-1)(4/11) + 1 = 23/11$$

$$Ye = (-2-6)(4/11) + 6 = 34/11$$

$$Ze = (5-2)(4/11) + 2 = 34/11$$

แต่เรายังไม่สรุปว่าจุดนี้เป็นจุดตัดจริง เพราะว่าจากโจทย์ เราได้ว่าระนาบ clipping ด้านหน้าอยู่ที่ Ze = 4 และ Ze = 4 34/11 เพราะฉะนั้นเราจึงไม่สามารถเห็นจุดตัดนี้ได้ เราต้องทำการทดสอบต่อไปเพื่อที่จะหาจุดตัดที่แท้จริง ต่อไปเราก็คำนวณหาค่า t ของระนาบที่เหลือต่อไป ซึ่งการคำนวณจะได้ผลตามนี้

Plane	t	Visibility
บน	4/11	NO (Ze = 34/11 4)
ล่าง	-8/5	NO
ขวา	กำหนดค่าไม่ได้	NO
ซ้าย	-1/2	NO
หลัง	8/3	NO
หน้า	2/3	YES (Ze = 4)

เพราะฉะนั้นจุดที่มีการตัดกันจะอยู่ที่ระนาบด้านหน้า ซึ่งคำนวณหาค่า Coordinate ได้ตามนี้

$$Xe = (4-1)(2/3) + 1 = 3$$

$$Ye = (-2-6)(2/3) + 6 = 2/3$$

$$Ze = (5-2)(2/3) + 2 = 4$$

จุด (3,2/3,4) จะแทนจุดที่มีการตัดกันอย่างแท้จริง ก่อนที่จะจบส่วนนี้ เราจะขอให้พิจารณาในกรณีพิเศษคือ เมื่อ $w = 0$ ในสมการ 4.39 ถึง 4.41 ค่าจำกัดต่าง ๆ เหล่านี้จะอยู่ในสภาวะที่ค่าของ w มีค่าเป็นบวกเสมอ ในความจริงนั้นเราสามารถให้ค่าของ w เป็นเท่าใด ๆ ก็ได้ ถ้าค่า w มีค่า = 0 จุดนั้นจะวางอยู่ในระนาบของจุดสังเกตุ (Ze=0) และถ้า w 0 จุดสังเกตุจะอยู่ภายในวัตถุ เนื่องจากรูปภาพที่วาดด้วยจุดสังเกตุหนึ่งจะโปร่งต้องถูกดำเนินกรรมวิธี Clipping Subroutine

4.5.2 Viewport Mapping

ภายหลังที่มีการ clip รูปภาพที่ตัดกับเนื้อที่การมองแล้ว ลำดับขั้นสุดท้ายก็คือ การ Map รูปภาพเหล่านี้ให้เป็น Viewport เพราะว่า Perspection Transformatun จะทำให้ได้ค่า Screen-coordinate แบบ 2 มิติเท่านั้น เราใช้ลำดับขั้นตอนที่กล่าวไว้แล้วในส่วนที่ 3.5 สำหรับการทำ Viewport Mapping ส่วนค่า Screen coordinate ที่แท้จริงนั้นของจอภาพจะหาได้โดยการใช้สมการ (3.36) จนถึงสมการ (3.42)

$$Xs = SCF\left[(d/s)(Xe/Ze)VX + L\right]$$

$$Ys = -(d/s)(Ye/Ze)VY + M \qquad (4.46)$$

เมื่อค่า Parameter ของ Viewing ทั้ง 4 ตัวคือ Center Size, Viewport ตั้งอยู่ที่ตำแหน่งศูนย์กลาง (L,M) ความกว้าง 2VX, ความกว้าง 2VY= สมการ 4.6 ที่ได้นี้เราจะสมมุติว่าภาพที่ปรากฏบนจอภาพนั้นมีขนาดเท่ากันทั้งทางด้าน VX และ VY (โดยเปรียบเทียบสมการ 4.46 กับสมการ 3.42 และสมการ 3.43)

4.6 Screen display OJ 3-D graphic

เราจะจบบทนี้ลงด้วยโปรแกรม computer 2 โปรแกรม ที่แสดงให้เห็นถึงความรู้ที่กล่าวมาแล้วบทก่อน ๆ โดยที่โปรแกรมแรกจะวาดภาพ Perspective ของลูกบาศก์และโปรแกรมที่ 2 จะวาดภาพพื้นผิวที่กำหนดโดยฟังก์ชั่นทางคณิตศาสตร์ที่อยู่ใน form ของ Z = (X,Y)

4.6.1 Perspective view of a cube

เราจะพิจารณาว่าลูกบาศก์นี้ตั้งอยู่ที่จุดกำเนิดของระบบ World Coordinate และจะกำหนดจุดต่าง ๆ ไว้ตามนี้

ลำดับของจุด	World Coordinate (X,Y,Z)
1	(1, 1, -1)
2	(1, 1, 1)
3	(1, -1, 1)
4	(1, -1, -1)
5	(-1, 1, -1)
6	(-1, -1, -1)
7	(-1, -1, 1)
8	(-1, 1, 1)

เราจะมองลูกบาศก์นี้ที่จุดสังเกตุ (12.5, 53.13°, 48.65°) พร้อมกับแกนการมอง Ze ชี้ไปยังจุด Origin ของระบบ World Coordinate เราจะนำรูปภาพที่ได้ใส่ในจอภาพที่มีขนาด $13\frac{1}{2}$" (252 mm) (โดยวัดตามเส้นทะแยงมุม) และจอภาพมีพื้นที่แสดงภาพขนาด 20 x 20 cm ถูกออกแบบมาให้มองที่ระยะห่างจากจอ 20 cm ในที่นี้เราจะสมมุติว่าจอภาพ Computer มีตำแหน่งที่อยู่ของจุด 320 x 200 ตำแหน่ง (Mode Medium Resolution ของเครื่อง IBM PC . เพราะฉะนั้นจะได้ Parameter ต่าง ๆ ดังนี้ D=12.5, O=53.13°, d =40, s =10 (ดูในรูป 4.13 วิธีการกำหนดค่า S)

สำหรับตัวอย่างในการคำนวณในที่นี้จะใช้จุดที่ 1 โดยใช้สมการ 4.28, เราก็สามารถคำนวณหาจุดในระบบ eye coordinate เป็นดังนี้

$$Xe = -1(0.8) + 1(0.6) = -0.2$$

$$Ye = -1(0.6)(0.684) -(1)(0.8)(0.684)+(-1)(0.73)$$

$$= -1.688$$

$$Ze = -1(0.6)(0.73)-(1)(0.8)(0.73)-(-1)(0.684)$$
$$+ 12.5 = 12.162$$

ใช้สมการ 4.31, 4.32 เพื่อกำหนดค่า Screen Coordinate

$$Xs = \frac{40}{10}\left(\frac{-0.2}{12.162}\right) = -0.066$$

$$Ys = \frac{40}{10}\left(\frac{-1.688}{12.162}\right) = -0.555$$

ต้องคำนวณหาค่า Ze และค่าของ Ze ของจุดทั้งหมดของลูกบาศก์จะอยู่ในช่วงระหว่าง 10.784 ถึง 14.205 เพราะฉะนั้นเราอาจจะตั้งค่า K_1 ไว้ที่ 10 และ K_2 ไว้ที่ 15 เพื่อให้แน่ใจว่าจุดทั้งหมดจะอยู่ภายใน Truncate view Pyramid และโดยการใช้สมการ 4.38 เราสามารถคำนวณหาค่า Zs ของจุดที่ 1 ได้ดังนี้

$$Zs = \frac{15(12.162-10)}{(15-10)12.162} = 0.533$$

การหาค่า Screen Coordinate ของจุดที่เหลือสามารถคำนวณได้โดยวิธีเดียวกัน และสมมุติว่าตอนนี้เราหาเสร็จเรียบร้อยแล้ว และเก็บไว้ในตารางที่ 1 เมื่อพิจารณาตาม colume Zs ในตาราง 4.1 จะพบว่าจุดที่ Zs นั้นอยู่ใกล้ผู้สังเกตุมากที่สุด (Zs=0.22) และจุดที่ 6 อยู่ไกลจากผู้สังเกตุมากที่สุด (Zs = 0.88)

ลำดับขั้นตอนสุดท้าย ก็คือการหาค่าCoordinate ที่ใช้กับจอภาพจริงคือจอภาพ IBM PC เราจะวาดภาพลูกบาศก์ลงบนจอภาพด้วย Mode Medium Resolution (320 x 200) ในที่นี้หาค่า Screen Coordinate ของ IBM PC สำหรับจุดที่ 1 โดยการใช้สมการ 4.46 จะได้ค่า Xs และ Ys ดังนี้

$$Xs = 1.2\left((-0.066)(100)+160\right) = 184.08$$

$$Ys = -(-0.555)(100) + 100 = 155.5$$

โปรแกรม 4.1 จะplotภาพ Perspectiveของลูกบาศก์ และก็มีการเปลี่ยนแปลงค่าViewing Parameter ภายในProgram ทำให้ได้รูปแบบของภาพ Perspective

TABLE 4.1 Viewing transformation

Vertex number	World coordinate			Eye coordinate			Screen coordinate			Device coordinate (IBM PC)	
	x	y	z	x_e	y_e	z_e	x_s	y_s	z_s	x_s	y_s
1	1	1	−1	−0.2	−1.688	12.162	−0.066,	−0.555	0.533	184.1	155.5
2	1	1	1	−0.2	−0.228	10.794	−0.075,	−0.085	0.220	183.1	108.5
3	1	−1	1	−1.401	0.866	11.962	−0.469,	0.289	0.492	135.8	71.1
4	1	−1	−1	−1.401	−0.573	13.329	−0.421	−0,178	0.749	141.6	117.8
5	−1	1	−1	−1.4	−0.867	13.037	0.429	−0.266	0.699	243.5	126.6
6	−1	−1	−1	0.199	0.227	14.205	0.056	0.064	0.888	198.8	93.6
7	−1	−1	1	0.199	1.687	12.837	0.062	0.525	0.663	199.5	47.5
8	−1	1	1	1.4	0.592	11.670	0.479	0.203	0.429	249.6	79.7

[SCF = 1.2, $V_x = V_y = 100$, $L = 160$, $M = 100$. [See Eq. (4.46).]

```
100 ' Program 4.1 : Draw a Cube
110 '  NV = Number of Vertices : NS = Number of Surfaces
130 '  NPS(J) = Number of vertices to form surface J
140 '
150 CLS : KEY OFF : SCREEN 1,0 : COLOR 1,2
160       GOSUB 210
170       GOSUB 280
180       GOSUB 530
190       GOSUB 660
200 END
210 '------ DEFINE VIEWING PARAMETERS ------
220       D=12.5: VD=40   : THETA=53.13: PHI=46.85: TX= 160: TY=100 : S=10
230       PI=3.141593     :SCF=1.2      : VX=100    : VY=100
240       THETA=THETA*PI/180 : PHI=PHI*PI/180
250       SN1=SIN(THETA) : SN2=SIN(PHI) :CN1=COS(THETA) :CN2=COS(PHI)
260       NV=8 : K1=10: K2=15: W=K2/(K2-K1)
270 RETURN
280 '------ VERTEX ARRAY  ----------
290       DIM V(8,3), EC(8,3), SC(8,3)
300       FOR J=1 TO NV
310           READ X,Y,Z
320           V(J,1)=X: V(J,2)=Y: V(J,3)=Z
330           GOSUB 410
340           EC(J,1)=XE: EC(J,2)=YE: EC(J,3)=ZE
350           GOSUB 460
360           SC(J,1)= XS   : SC(J,2)= YS   : SC(J,3)=ZS
370       NEXT J
380 RETURN
390       DATA 1,1,-1, 1,1,1, 1,-1,1, 1,-1,-1
400       DATA -1,1,-1, -1,1,1, -1,-1,1, -1,-1,-1
410 '------ EYE COORDINATES  ---------
420       XE= -X*SN1 + Y*CN1
430       YE= -X*CN1*CN2 - Y*SN1*CN2 + Z*SN2
440       ZE= -X*SN2*CN1 - Y*SN2*SN1 - Z*CN2 + D
450 RETURN
460 '-----  SCREEN COORDINATES -------
470       XS=(VD/S)*(XE/ZE)                          ' See Eq.(4.31)
480       YS=(VD/S)*(YE/ZE)                          ' See Eq.(4.32)
490       ZS=W*(1- K1/ZE)                            ' See Eq.(4.38)
500           XS=XS*VX + TX     : XS=SCF*XS          ' PC device coordinates
510           YS=-YS*VY + TY                         ' See Eq.(4.46)
520 RETURN
530 '------  SURFACE ARRAY   ----------
540 DIM SF(6,5),NPS(6)
550 NS=6
560 FOR I=1 TO NS: READ NPS(I): NEXT I
570     FOR J=1 TO NS
580         FOR K=1 TO NPS(J)
590             READ SF(J,K)
600         NEXT K
610     NEXT J
620 RETURN
630 DATA 5,5,5,5,5,5
640 DATA 1,2,3,4,1, 1,6,8,2,1, 5,6,7,8,5, 4,3,7,6,4, 2,8,7,3,2, 1,4,6,5,1
660 '------  PLOT   ---------------
670 FOR J=1 TO NS
680         FOR K=1 TO NPS(J)
690             NN=SF(J,K)
700             A=SC(NN,1) : B= SC(NN,2)
710             IF K=1 THEN PSET (A,B): GOTO 730
720             LINE - (A,B)
730         NEXT K
740 NEXT J
750 RETURN
```

PROGRAM 4.1 Draw a cube

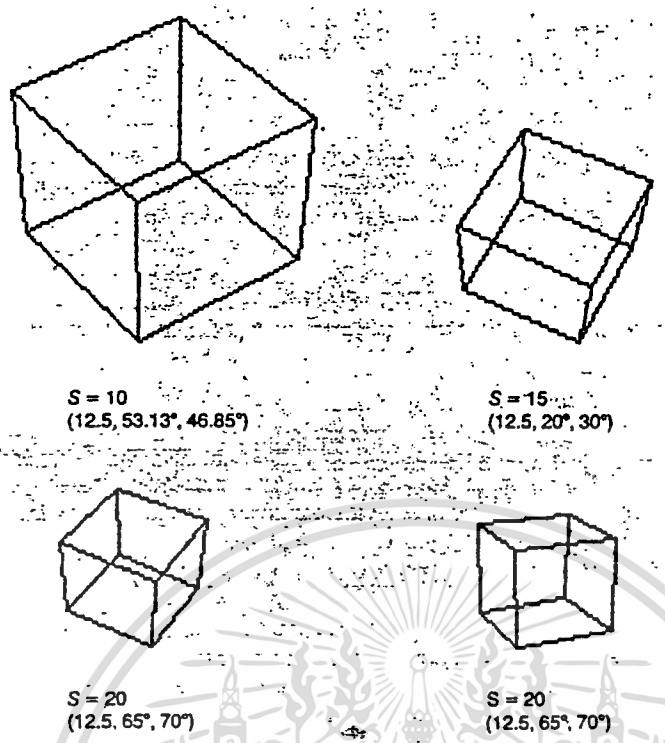


FIGURE 4.16 Perspective projections of the cube at various viewing angles

```
10 ' Program 4.2: Plot of 3-D Functions
20 ' (D,theta,phi) = Viewpoint
30 ' VD = Distance of the Projection Plane from the XY Plane.
40 ' S = Centered screen size. (see Eq.(4.31).
50 CLS: KEY OFF
60 SCREEN 1,0 : COLOR 1,2
70 '
80     GOSUB 110
90     GOSUB 170
100 END
110 '-------- Define Viewing Parameters ---------
120       TX= 160: TY = 100: PI = 3.141593 : SCF= 1.2
130       D= 30: VD=420: THETA=  0: PHI= 70 : S=2
140       THETA=THETA*PI/180 : PHI=PHI*PI/180
150       SN1=SIN(THETA) : SN2=SIN(PHI) : CN1=COS(THETA): CN2=COS(PHI)
160 RETURN
170 '-------- Define the 3-D function to plot ------
180 DEF FNZ(X,Y) = 7*EXP(-.1*(X*X + Y*Y))
190 FOR X=8 TO -8 STEP -1
200       FLAG =0
210       FOR Y=-8 TO 8 STEP .5
220           Z=FNZ(X,Y)
230           GOSUB 290
240           GOSUB 340
250           GOSUB 420
260       NEXT Y
270 NEXT X
280 RETURN
290 '--------- Compute Eye Coordinates ---------------
300       XE= -X*SN1 + Y*CN1
310       YE= -X*CN1*CN2 - Y*SN1*CN2 + Z*SN2
320       ZE= -X*SN2*CN1 - Y*SN2*SN1 - Z*CN2 + D
330 RETURN
340 ' -------- Compute Screen Coordinates ----------
350 ' We assume s1 = s2 = 1 so that both window and viewport coordinates
360 ' will be exactly same. (see Eqs. (3.39) and (3.40).)
370       XS= (VD/S)*(XE/ZE)                          ' Viewport coordinates
380       YS= (VD/S)*(YE/ZE)
390       XS= XS + TX     : XS= SCF*XS                ' PC device coordinates
400       YS= -YS + TY
410 RETURN
420 '---------- Plotting Routine ---------------------
430 IF XS<0 OR XS>319 OR YS<0 OR YS>199 THEN FLAG=0 : GOTO 460
440 IF FLAG=0 THEN PSET (XS,YS): FLAG=1 : GOTO 460
450 LINE - (XS,YS)
460 RETURN
```

PROGRAM 4.2 Plot of 3-D functions

ของลูกบาศก์เปลี่ยนไปตามที่แสดงดังรูป 4.16

ถึงแม้ว่าเราต้องใช้ Clipping routine ในแต่ละจุดของลูกบาศก์ แต่ตามที่ปรากฏในตาราง 4.1 ปรากฏว่าจุดทั้งหมดอยู่ภายใน Truncated viewing Pyramid เพราะฉะนั้น Clipping Algorithm จะยอมรับเส้นทุกเส้นโดยไม่มีการตัดออก

4.6.2 Surface of the form Z = f(X,Y)

กราฟของจุดทั้งหมด (X,Y,Z) ที่เป็นไปตามฟังก์ชัน Z=f(X,Y) จะถูก plot โดยกรรมวิธีที่กล่าวมาแล้วในบทก่อนแต่ไม่เหมือนโปรแกรม 4.1 คือเราต้องเตรียม Coordinate ของจุดต่าง ๆ ให้เป็น DATA และจุดต่าง ๆ เหล่านี้จะเกิดขึ้นภายในส่วนของ Program Program 4.2 เป็นโปรแกรมที่ใช้สร้างภาพแทนพื้นผิวโดยเป็นแบบ curve โครงร่างและโปรแกรมนี้ จะ plot funtion $Z = 7.2^{-1(x^2+y^2)}$ และถ้าต้องการให้ผิวเรียบนิ่งขึ้นก็จะต้องมีการคำนวณจุดให้มากกว่าเดิม แต่ก็ต้องใช้เวลา execute นานขึ้น

รูป 4.17 แสดงให้เห็นวิวต่าง ๆ ของ funtion ที่เลือกด้วยค่า Viewing Parameter ที่ต่าง ๆ กัน ถ้าเราเลือกตำแหน่ง Viewing ที่ต่างกันภาพที่ได้ก็จะยุ่งไปหมดเนื่องจาก curve ที่ได้จะทับกัน program นี้จะสร้าง surface โปร่งใส ซึ่งรูปแบบนี้ทำให้เราเข้าใจผิดว่าเราสามารถมองเห็นพื้นผิวที่อยู่ใกล้ไปยังพื้นผิวที่อยู่ไกลออกไปที่อยู่ทางด้านหลังได้ ถ้าเราสามารถหลีกเลี่ยงการวาดผิวที่ถูกบังเราก็จะได้ภาพที่เป็นธรรมชาติ และในบทต่อไปได้เตรียมวิธีในการแก้ปัญหานี้

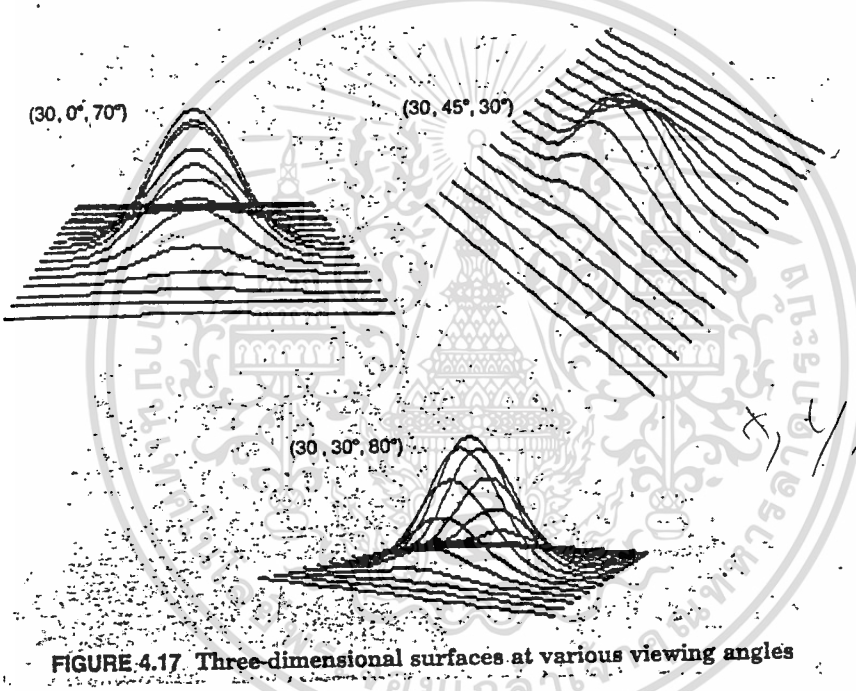


FIGURE 4.17 Three-dimensional surfaces at various viewing angles

บทที่ 5

Hidden line and Surface Removals

ในการสร้างภาพ 3 มิติ ให้เหมือนจริงนั้น บ่อยครั้งที่เราต้องเอาเส้นหรือผิวที่ถูกบังออกและการที่จะทำเช่นนี้ได้เราต้องสร้างวิธีการที่เป็นการกำหนดว่าพื้นผิวหรือเส้นเหล่านั้นอยู่ในสภาวะเช่นไร เช่น มองเห็นหมด, มองเห็นเป็นบางส่วน หรือมองไม่เห็นหมด ต่อจากนั้นเราก็จะแสดงส่วนของวัตถุที่มองเห็นและกำจัดส่วนของวัตถุที่มองไม่เห็นออกไปจากภาพ ในการสร้าง Algorithm ที่จะทำการเอาส่วนของพื้นผิวและเส้นที่มองไม่เห็นออกนั้นจำเป็นต้องใช้เวลาในการประมวลผลและหน่วยความจำเป็นจำนวนมาก องค์ประกอบเหล่านี้เป็นข้อจำกัดของ Microcomputer โดยทั่ว ๆ ไปจึงทำให้ Algorithm ของการกำจัดเส้นหรือผิวที่มองไม่เห็นไม่มีประสิทธิภาพ และถ้าต้องการให้มีประสิทธิภาพดีขึ้นนักเขียนโปรแกรมจำเป็นต้องมีการดัดแปลงเทคนิคบางอย่าง สำหรับเหตุผลนี้เองเราจะจำกัดเนื้อหาของเรื่องดังกล่าวนี้ในรูปแบบค่อนข้างพื้นฐานแต่ก็ใช้ประโยชน์ได้และ Algorithm ที่มีอยู่ในบทนี้ก็สามารถนำมาใช้บนเครื่อง Microcomputer ได้

ก่อนที่เราจะแนะนำทฤษฎีการทำงาน เราจะให้ลักษณะสมมุติที่สำคัญในทางปฏิบัติ 2 อย่าง-สมมุติแรก คือวัตถุประกอบด้วยผิวหน้าแสดงรูปร่าง เราจะใช้เส้นแทนรูปร่างดังกล่าวโดยให้เป็นรูปหลายเหลี่ยม ทั้งนี้เนื่องจากรูปภาพโดยพื้นฐานแล้วจะต้องประกอบด้วยเส้น

-สมมุติที่สอง เราจะถือว่าพื้นผิวของรูปหลายเหลี่ยม จะมีลักษณะแบนหมายความว่าจุดทั้งหมดของรูปหลายเหลี่ยมจะวางอยู่บนระนาบเดียวกัน คุณสมบัติในลักษณะนี้จะทำให้เกิดความสะดวกในการเปรียบเทียบความสัมพันธ์ของตำแหน่งระหว่างรูปหลายเหลี่ยม

5.1 The Visibility of single convex object

ในตอนนี้เราใช้วัตถุเพียงชิ้นเดียวและวัตถุนี้นูน เราสามารถแก้ปัญหาเกี่ยวกับการมองเห็นโดยการตรวจสอบทิศทางของ normals ของแต่ละผิวหน้าของวัตถุ และค่า face normal ถูกกำหนดให้เป็น Vector ตั้งฉากกับผิวหน้า วัตถุนูนคือวัตถุรูปแบบหนึ่งที่มีการเชื่อมโยงเส้นระหว่างจุดต่อภายในวัตถุนั้น ๆ และอยู่ในวัตถุประเภทเหล่านี้ (ลูกบาศก์, ทรงกลม, ปิรามิด) ในทำนองเดียวกันรูปหลายเหลี่ยมนูนก็คือ รูปหลายเหลี่ยมรูปแบบหนึ่งที่มีการเชื่อมโยงจุดต่อภายในรูปหลายเหลี่ยมนั้นและอยู่ในประเภทรูปหลายเหลี่ยมเหล่านี้ (สี่เหลี่ยม, สามเหลี่ยม, วงกลม, หกเหลี่ยม)

5.1.1 Approach to the visibility problem

พิจารณาที่รูปทรงลูกบาศก์ (ในรูป 5.1 a) ลูกบาศก์นี้จะมีจุด 8 จุด มีด้าน 12 ด้าน พื้นผิว 6 ด้านแต่ละพื้นผิวจะมีลักษณะแบนล้อมรอบด้วยด้านทั้ง 4 ถ้าเรามองลูกบาศก์จากจุดสังเกตุ P จะเห็นว่ามีเพียง 2 หรือ 3 พื้นผิวเท่านั้นที่มองเห็นได้ และผิวอื่นที่อยู่ทางด้านหลังจะมองไม่เห็น เพื่อหลีกเลี่ยงการวาดพื้นผิวที่ถูกบัง เราจำเป็นต้องใช้วิธี Visibility test เพื่อตรวจสอบเส้นที่มองเห็นและเส้นที่มองไม่เห็น

ในการสร้าง Visibility test เราจะแนะนำให้ทราบ Vector 2 แบบที่ต้องนำมาเกี่ยวข้องกับแต่ละพื้นผิวก็คือ Surface Normal Vector และ line of sight Vector โดยที่เราจะใช้อักษรตัวพิมพ์หน้าเล็กแทนค่า Vector ทั้ง 2 สองนี้ โดยเราจะกำหนดให้ Surface normal vector เป็น vector ที่ชี้ออกจากผิวของวัตถุ (ตั้งฉาก) และ line of sight vector จะแทนเส้นตรงที่ลากผ่านจุดสังเกตุไปยังจุดเริ่มต้นของ surface normal vector (ดูรูป 5.2)

เราจะคำนวณหามุม ϕ ระหว่าง n และ l ถ้ามุมนี้อยู่ในช่วง 0 ถึง 90 องศา พื้นผิวนั้นจะอยู่ในสภาวะที่มองเห็นต้องนำออกแสดงผลถ้ามุมดังกล่าวมากกว่า 90 องศา พื้นผิวนั้นจะอยู่คนละด้านกับผู้สังเกตุและอยู่ในสภาวะที่มองไม่เห็นจะไม่นำออกแสดงผล ในหน้าถัดไปเราจะแสดงให้เห็นวิธีการคำนวณหา normal

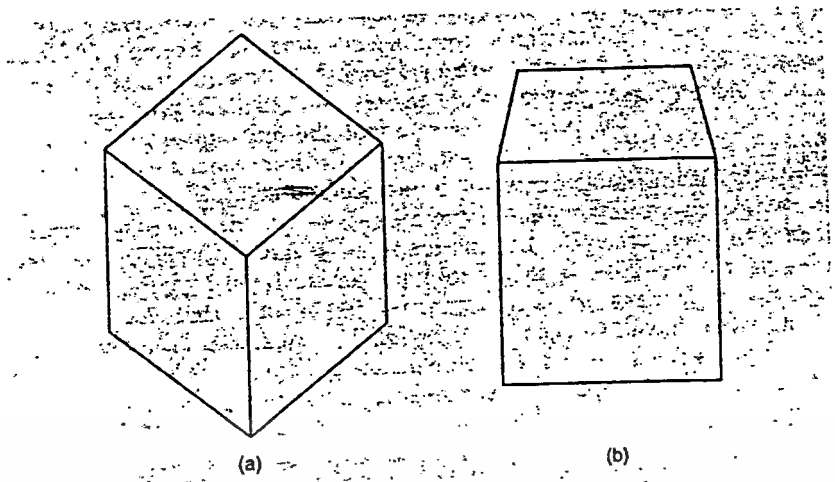


FIGURE 5.1 Number of visible surfaces of a cube: (a) has three sides visible and (b) has only two sides visible.

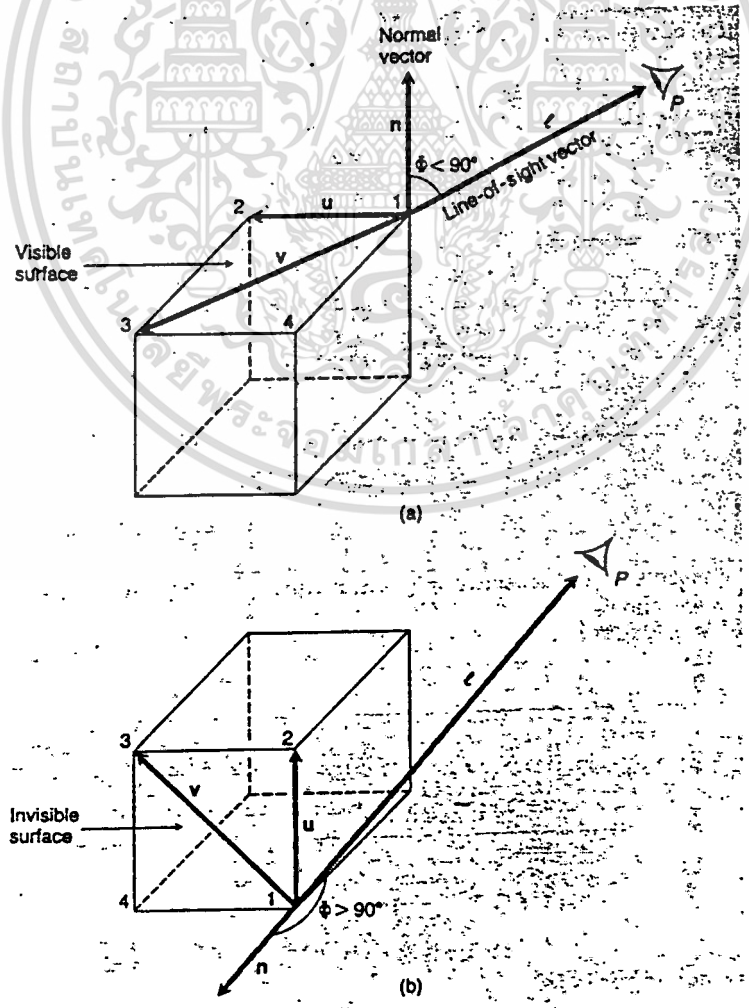


FIGURE 5.2 Normal and line of sight vectors that define the visibility

vector และมุมระหว่าง vector ทั้ง 2

Surface Normal Vector

ในการคำนวณหา normal vector ที่เกี่ยวข้องกับพื้นผิวใด ๆ เรา จะต้องกำหนดการนับจุดของพื้นผิวของลูกบาศก์ในทิศทางทวนเข็มนาฬิกา สำหรับพื้นผิวใด ๆ จุดเริ่มต้นจะเป็นจุดใด ๆ ก็ได้ แต่ข้อสำคัญจุดที่เหลือตามมา จะต้องถูกนับในทิศทางทวนเข็มนาฬิกาโดยที่จะต้องพิจารณาการมองจากภายนอกลูกบาศก์ (ดูรูป 5.2) ภายหลังเมื่อนับจุดดังกล่าวในทิศทางทวนเข็มนาฬิกาแล้ว ขั้นต่อไปคือต้องกำหนด vector U ที่ชี้จากจุดที่ 1 ไปยังจุดที่ 2 และ vector V ชี้จากจุดที่ 1 ไปยังจุดที่ 3 และค่า Cross Product UxV จะทำให้เกิดค่า Surface Normal Vector ที่ชี้ออกจากผิว (ถ้าจุดทั้งหมดถูกกำหนดในทิศทางตามเข็มนาฬิกา, vector นี้จะชี้เข้าหาผิว) ในพีชคณิตเชิงเส้นค่า Cross Product ของทั้ง 2 Vector คือ

$$n = V \times U$$

$$= (bf-ce,\ cd-af,\ ae-bd) \qquad (5.1)$$

เมื่อจุดโดออดิเนตของ $u = (a,b,c)$

$v = (d,e,f)$

สมมุติว่าเรากำหนดจุดของพื้นผิวด้านหน้าของลูกบาศก์หนึ่งในรูป 5.3 เป็นตามนี้

ลำดับจุด	Coordinate (X,Y,Z)
1	1,1,-1
2	1,1,1
3	1,-1,1
4	1,-1,-1

ดังนั้น $u = (1,1,1)-(1,1,-1) = (0,0,2)$

$v = (1,-1,1)-(1,1,-1) = (0,-2,2)$

normal คือ

$n = u \times w = (4,0,0)$

line of sight vector

เราสามารถคำนวณหา line of sight vector โดยการเชื่อมโยงจุดจากจุดสังเกตุ (D,θ,ϕ) ไปยังจุดบนพื้นผิววัตถุ เพื่อให้สะดวกโดยปกติแล้วเราจะเลือกจุดเริ่มต้นของ Surface Normal Vector (จุด 1 ในรูป 5.3) โดยการใช้สมการ (4.1), เรากำหนดจุดสังเกตุในเทอมของ spherical coordinate เพราะฉะนั้น vector 1 คือ

$$1 = (D\sin\phi\cos\theta,\ D\sin\phi\sin\theta,\ D\cos\theta) - (X_1,Y_1,Z_1) \qquad (5.2)$$

เมื่อ (X_1,Y_1,Z_1) คือ coordinate ของจุดที่ 1

และจะสมมุติจุดสังเกตุอยู่ที่ (5.4, 27°, 56°) = (4,2,3) เราจะคำนวณหาค่า Vector 1 ได้ดังนี้

$$1 = (4,2,3)-(1,1,-1) = (3,1,4)$$

Visibility test

เราจะได้ค่า dot Product ของ Vector n และ 1 คือ

$$n.1 = (n_1,n_2,n_3)\cdot(l_1,l_2,l_3)$$
$$= n_1l_1+n_2l_2+n_3l_3 \qquad (5.3)$$

จากพีชคณิตเชิงเส้นเรารู้ว่า dot Product ในสมการ 5.3 จะมีคุณสมบัติดังนี้

$$n.1 = n\ 1\ .\cos\phi$$
$$\phi = \arccos\frac{n.1}{n\ 1} \qquad (5.4)$$

เมื่อ n = ความยาว n

1 = ความยาว 1

ϕ = มุมระหว่าง n และ 1

สำหรับพื้นผิวที่มองเห็น ϕ จะอยู่ในช่วง 0 ถึง 90 นั่นคือ $n.1>0$, สำหรับพื้นผิว, ที่ถูกบังหรือมองไม่เห็น ϕ จะอยู่ในช่วง 90° ถึง 180° นั่นคือ $n.1<0$ และจากตัวอย่าง ค่าของ dot Product จะมีค่า

$$n.1 = (4,0,0)\cdot(3,1,4) = 12>0$$

ดังนั้น $n.1>0$ พื้นผิวนี้จะหมายเห็นจากจุดสังเกตุ (4,2,3)

5.1.2 Development of computer program

เพื่อแสดงให้เห็นวิธีการกำจัดเส้นที่ถูกบัง เราจะใช้ program 5.1 ซึ่งจะทำการวาดลูกบาศก์ซึ่งแสดงให้เห็นในรูป 5.3 ลูกบาศก์นี้มี 6 ผิวหน้า, บน, ล่าง และข้างอีก 4 ผิวหน้า และมี 12 ด้านซึ่งถูกกำหนดโดยจุด 8 จุด เราจะแสดงโปรแกรมเป็น 6 ส่วนย่อย ๆ

ส่วนที่ 1 Vector Array

ลำดับแรกเราต้องกำหนดจุด 8 จุดของลูกบาศก์ในTerm ของระบบพิกัดฉาก ตำแหน่งของแต่ละจุดเป็นดังรูป 5.4 a

ตำแหน่งจุด	Coordinate
1	(1,1,-1)
2	(1,1,1)
3	(1,-1,1)
4	(1,-1,-1)
5	(-1,1-1)
6	(-1,-1,-1)
7	(-1,-1,1)
8	(-1,1,1)

ต่อมาเราจะใช้ array 2 มิติ V(I,J) โดยที่ I คือตำแหน่งจุด (1 ถึง 8) และ คือตัวกำหนด coordinate (1-3) สำหรับตัวอย่าง V (1,1) หมายถึง X coordinate ของจุดที่ 1 ,V (1,2) Y coordinate ของจุดที่ 1 และ V (1,3) คือ Z coordinate ของจุดที่ 1 ส่วนของโปรแกรมที่ 1 จะเป็นตัวกำหนด Viewing parameter ตาง ๆ และ load coordinate ของจุดเข้าไปใน array 2 มิติ ตามค่าตำแหน่งจุดที่ถูกอ่าน สำหรับ screen coordinate ของแต่ละจุด ก็จะถูกคำนวณหาในส่วนนี้ สำหรับแต่ละค่า I;sc(I,1) และ sc(I,2) คือ screen coordinate Xs และYs ของจุด I

ส่วนที่ 2 Surface Array

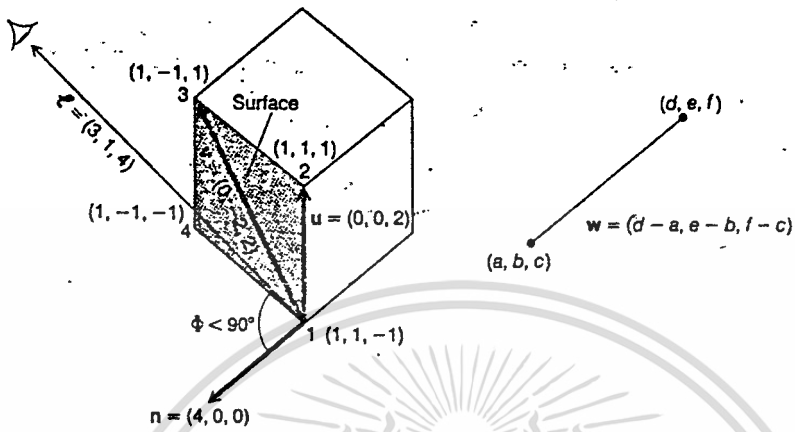


FIGURE 5.3 Visibility test applied to the first surface of the cube

```
10 ' Program 5.1 : Single Object with Hidden Lines Removed
20 CLS : KEY OFF : SCREEN 2
30        GOSUB 110
40        GOSUB 180
50        GOSUB 410
60        GOSUB 540
70        GOSUB 660
80        GOSUB 770
90        GOSUB 940
100 END
110 REM : DEFINE VIEWING PARAMETERS
120       D=5.4: VD=200 : THETA=27 : PHI=56 : TX= 320: TY=100
130       PI=3.141593    : SCF=2.4
140       THETA=THETA*PI/180 : PHI=PHI*PI/180   : S= 2
150       SN1=SIN(THETA) : SN2=SIN(PHI) :CN1=COS(THETA) :CN2=COS(PHI)
160       NV=8 : K1=7: K2=10: W=K2/(K2-K1)
170 RETURN
180 REM : VERTEX ARRAY
190       DIM V(8,3), EC(8,3), SC(8,3)
200       FOR J=1 TO NV
210            READ X,Y,Z
220            V(J,1)=X: V(J,2)=Y: V(J,3)=Z
230            GOSUB 310
240            EC(J,1)=XE: EC(J,2)=YE: EC(J,3)=ZE
250            GOSUB 360
```

PROGRAM 5.1 Single Object with Hidden Lines Removed

```
260           SC(J,1)= XS    : SC(J,2)= YS    : SC(J,3)=ZS
270       NEXT J
280 RETURN
290       DATA 1,1,-1, 1,1,1, 1,-1,1, 1,-1,-1
300       DATA -1,1,-1, -1,-1,-1, -1,-1,1, -1,1,1
310 REM : EYE COORDINATES
320       XE= -X*SN1 + Y*CN1
330       YE= -X*CN1*CN2 - Y*SN1*CN2 + Z*SN2
340       ZE= -X*SN2*CN1 - Y*SN2*SN1 - Z*CN2 + D
350 RETURN
360 REM : SCREEN COORDINATES
370       XS=(VD/S)*(XE/ZE)
380       YS=(VD/S)*(YE/ZE)
390       ZS=W*(1- K1/ZE)
400 RETURN
410 REM : SURFACE ARRAY
420 DIM SF(6,5),NPS(6)
430 NS=6
440 FOR I=1 TO NS: READ NPS(I): NEXT I
450       FOR J=1 TO NS
460           FOR K=1 TO NPS(J)
470               READ SF(J,K)
480           NEXT K
490       NEXT J
500 RETURN
510 DATA 5,5,5,5,5,5
520 DATA 1,2,3,4,1, 1,5,8,2,1, 5,6,7,8,5, 4,3,7,6,4
530 DATA 2,8,7,3,2, 1,4,6,5,1
540 REM : NORMAL ARRAY
550 DIM N(6,3),U(3),W(3)
560       FOR J=1 TO NS
570           FOR K=1 TO 3
580               U(K)= V(SF(J,2),K) - V(SF(J,1),K)
590               W(K)= V(SF(J,3),K) - V(SF(J,1),K)
600           NEXT K
610           N(J,1)= U(2)*W(3) - W(2)*U(3)
620           N(J,2)= U(3)*W(1) - W(3)*U(1)
630           N(J,3)= U(1)*W(2) - W(1)*U(2)
640       NEXT J
650 RETURN
660 REM : VISIBILITY TEST
670 DIM VSF(6)
680 XP= D*SN2*CN1 : YP= D*SN2*SN1 : ZP= D*CN2
690       FOR J=1 TO NS
700           LX=XP - V(SF(J,1),1)
710           LY=YP - V(SF(J,1),2)
720           LZ=ZP - V(SF(J,1),3)
730           T= N(J,1)*LX + N(J,2)*LY + N(J,3)*LZ
740           IF T <= 0 THEN VSF(J)=0 ELSE VSF(J)=1
750       NEXT J
760 RETURN
770 REM: VISIBLE EDGE ARRAY
780       M=1
790       FOR J=1 TO NS
800           IF VSF(J)=0 THEN 920
810           E1= SF(J,1)
820           FOR K=2 TO NPS(J)
830               E2=SF(J,K)
840               FOR L=1 TO M
850                   IF E(L,1)=E2 AND E(L,2)=E1 THEN E(L,3)=2: E(L,4)=J:
                      GOTO 890
860               NEXT L
870               E(M,1)=E1: E(M,2)=E2: E(M,3)=1: E(M,4)=J
880               M=M+1
890               E1=E2
900           NEXT K
910           NVE=M-1
920       NEXT J
930 RETURN
940 REM : PLOT
950       FOR J=1 TO NVE
960           IF E(J,3)=0 THEN 1010
970           F1=E(J,1): F2=E(J,2)
980           A=SC(F1,1):B=SC(F1,2): C=SC(F2,1):D=SC(F2,2)
990           A=SCF*A+TX : B=TY-B : C=SCF*C+TX : D=TY-D
1000          LINE (A,B) - (C,D)
1010      NEXT J
1020 RETURN
```

ตามที่แสดงไว้ในรูป 5.4 (b)จะมีพื้นผิว 6 ผิวในลูกบาศก์นี้ และเรากำหนดพื้นผิวเหล่านี้โดยตำแหน่งจุด (I) เนื่องจากเราวาดพื้นผิวแต่ละผิวโดยการต่อจุดให้เหมาะสม เราจึงใช้ Array SF(I,J) กำหนด surface I และจุดทั้งหมด (J) ที่ประกอบขึ้นเป็นพื้นผิวนั้น สำหรับตัวอย่างพื้นผิว 1 เราต้องต่อตำแหน่งจุด 1 เข้ากับ 2, 2 เข้ากับ 3, 3 เข้ากับ 4, 4 เข้ากับ 1 เราจะ save ค่าเหล่านี้ในลักษณะดังนี้ SF (1,1) = 1, SF(1,2)= 2, SF(1,1)= 3, SF(1,4) =4 และ SF (1,5) = 1 ตามลำดับ สำหรับพื้นผิว 2 เราต้องต่อจุด 1 เข้ากับ 5, จุด 5 เข้ากับ 8, 8 เข้ากับ 2, 2 เข้ากับ 1 พื้นผิวนี้ก็จะถูกกำหนดโดยวิธีการนี้ array ที่เสร็จสิ้นสมบูรณ์ถูกแสดงให้เห็นในตาราง 5.1 ในการใส่ค่าลงใน SF(I,J) สำหรับแต่ละพื้นผิวนั้นจุดต้องถูกนับในทิศทางทวนเข็มนาฬิกาตามที่ปรากฏจากภายนอกลูกบาศก์ การเก็บจุดในลักษณะนี้จะทำให้แน่ใจว่า Normal Vector จะมีทิศทางพุ่งออกมาข้างนอก โปรแกรมส่วนที่ 2 จะเป็นการกำหนดจุดลงใน SF(I,J) ของแต่ละพื้นผิว

<u>ส่วนที่ 3</u> normal vector array

หลังจากได้เก็บค่า coordinat ของจุดทั้งหมดและกำหนดพื้นผิวจากจุดเหล่านี้แล้ว เราจะคำนวณหา normal vector ที่สัมพันธ์กับแต่ละพื้นผิว ภาพที่แสดงในสมการ (5.1) การคำนวณนี้จะใช้พื้นฐานความจริงของค่า cross product ของ 2 vector ก็คือค่า normal ที่ประกอบขึ้นจากระนาบของ vector ทั้งสองในแต่ละพื้นผิวจะมีจุด 4 จุดในลูกบาศก์ เราหา vector U ได้โดยการต่อจุดที่ 1 ไปยังจุดที่ 2 ของจุดเหล่านี้และ vector V ได้โดยการต่อจุดที่ 1 ไปยังจุดที่ 3 เราใช้สมการ (5.1) คำนวณหาค่า cross product UxV และก็เก็บค่า coordinat ของ normal vector Nลงใน normal array N (I,J),N(I,1),N(I,2),N(I,3)คือส่วนประกอบของ X,Y,Z ของ normal vector ของพื้นผิว I array ที่คำนวณเสร็จแล้วถูกแสดงในตาราง 5.2

<u>ส่วนที่ 4</u> lingh-of-sight vector

สำหรับแต่ละ surfad เราจะเลือกค่า line of-sight vector l ตั้งต้นของ surface vector กับจุดสังเกตุ (Xe,Ye,Ze) เราจะแนะนำตัวแปร LX, LY,LZ เพื่อคำนวณ l vector coordinate และกำหนด l=(LX,LY,LZ)

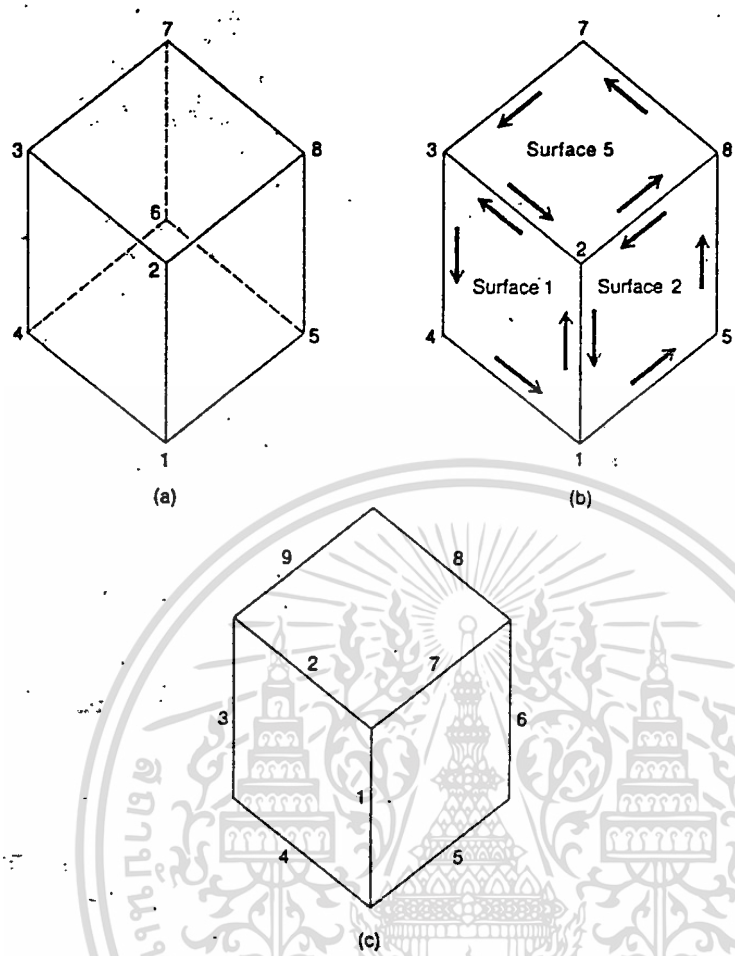


FIGURE 5.4 Definition of (a) vertices, (b) surfaces, and (c) edges for the cube

TABLE 5.1 Surface array

I (Surface)	*J* (Points to be connected)				
	1	2	3	4	5
1	1	2	3	4	1
2	1	5	8	2	1
3	5	6	7	8	5
4	4	3	7	6	4
5	2	8	7	3	2

TABLE 5.2 Normal array $N(I,J)$

I (Surface)	J		
	1 (x)	2 (y)	3 (z)
1	4	0	0
2	0	4	0
3	−4	0	0
4	0	−4	0
5	0	0	4
6	0	0	−4

TABLE 5.3 Calculation of line-of-sight vector array

	J			Eye coordinate		First vertex	
Surface, I	1(x)	2(y)	3(z)	(x_e, y_e, z_e)		(x_1, y_1, z_1)	
1	3	1	4	(4, 2, 3)	–	(1, 1, −1)	
2	3	1	4	(4, 2, 3)	–	(1, 1, −1)	
3	5	1	4	(4, 2, 3)	–	(−1, 1, −1)	
4	3	3	4	(4, 2, 3)	–	(1, −1, −1)	
5	3	2	2	(4, 2, 3)	–	(1, 1, 1)	
6	3	1	4	(4, 2, 3)	–	(1, 1, −1)	

TABLE 5.4 Visibility test results

Surface, I	$N(I,J)$	L	$N(I,J) \cdot L$	Visibility
1	(4, 0, 0)	(3, 1, 4)	12	Visible
2	(0, 4, 0)	(3, 1, 4)	4	Visible
3	(−4, 0, 0)	(5, 1, 4)	−20	Invisible
4	(0, −4, 0)	(3, 3, 4)	−12	Invisible
5	(0, 0, 4)	(3, 2, 2)	8	Visible
6	(0, 0, −4)	(3, 1, 4)	−16	Invisible

TABLE 5.5 Edge array $E(I,J)$

Surface, I	J			
	1	2	3	4
1	1	2	2	1
2	2	3	2	1
3	3	4	1	1
4	4	1	1	1
5	1	5	1	2
6	5	8	1	2
7	8	2	2	5
8	8	7	1	5
9	7	3	1	5

Note: J = 1 and 2 are endpoints;
J = 3 is the edge type;
J = 4 is the surface.

array ที่เสร็จสิ้นสมบูรณ์ถูกแสดงให้เห็นในตาราง 5.3

ส่วนที่ 5 Visibility test

เราจะใช้ visibility test กับแต่ละผิวของลูกบาศก์ โดยคำนวณหาค่า dot product n.l สำหรับแต่ละผิว ผลลัพธ์ที่ได้อยู่ในตาราง 5.4

ส่วนที่ 6 edge array

ตามที่เห็นมาแล้วในส่วน 5 จะมีพื้นผิวที่เห็นเพียง 3 ส่วน และพื้นผิวที่เห็นจะถูกกำหนดโดย11 ด้าน ดูในรูป 5.4(c) อย่างไรก็ตาม ด้านเช่น 3,4,5,6,8, และ 9 เป็นแต่ละขอบเขตสิ้นสุดของพื้นผิวที่มองเห็นด้วยเหตุนี้เราต้องการเพียง 9 ด้านเท่านั้น ในการกำหนดพื้นผิวที่มองเห็น 3 ส่วน ซึ่งมีผลเหมือนกันว่าแต่ละพื้นผิวของลูกบาศก์ใช้ด้าน 4 ด้าน

เราจะแนะนำ edge array E(I,J) สำหรับแต่ละด้าน I เราจะเก็บค่า 4 ตัวเลข ลำดับที่ 1 เราจะเก็บค่าตัวเลขของจุดปลาย 2 จุด ของด้าน I ไว้ใน E(I,1) และ E(I,2) ลำดับที่ 2 เราจะกำหนดจำนวนของพื้นผิวที่มองเห็นซึ่งมีด้าน I เป็นขอบเขต ตัวเลขนี้จะถูกใส่ใน E(I,3) ถ้าด้านนี้เป็นขอบเขตของพื้นผิวที่มองเห็น 2 ด้าน ก็จะใส่ค่า E(I,3) = 2 ลำดับที่ 3 เราจะกำหนดด้านไหนเป็นขอบพื้นผิวใดและเก็บลงใน E (I,4) ถ้าด้านหนึ่งเป็นของพื้นผิวมากกว่า 1 พื้นผิว ก็ให้ค่าพื้นผิวที่พบในครั้งแรกเก็บลงใน E(I,4) ส่วนค่าด้านต่าง ๆ เป็นดังรูป 5.4 (c) ส่วน edge array ที่เสร็จสมบูรณ์แล้วอยู่ในตาราง 5.5 ไม่มีลักษณะลูกบาศก์ใดที่จะแสดงได้มากกว่า 9ด้านถึงแม้ว่าลูกบาศก์นั้นจะประกอบด้วย 12 ด้านก็ตาม

เราสังเกตุว่าด้านที่อยู่ทางด้านนอกของลูกบาศก์ (3,4,5,6,8,9) ทั้งหมดนี้จะมีค่า E(I,3) = 1 และในทำนองเดียวกันด้านที่อยู่ภายในลูกบาศก์ (1,2,7) จะมีค่าE(I,3) =2 ข้อแตกต่างระหว่างด้านภายในไม่เป็นสิ่งจำเป็นเมื่อเราวาดวัตถุเพียงชิ้นเดียว แต่ข้อแตกต่างนี้จะมีความสำคัญมากเมื่อเราวาดวัตถุมากกว่า 1 ชิ้น (ดูในส่วน 5.2)

โปรแกรมในส่วนที่ 6 จะใส่ค่าด้านต่างใน edge array จุดปลายของแต่ละด้านที่ถูกอ่านจาก surface array และเก็บลงใน edge array

และflog จะถูก set เป็น 1 เมื่อด้านนั้นถูกพบเป็นครั้งที่ 2 flag ก็จะถูก set เป็น 2

ส่วนที่ 7 Plotting the object

ขณะนี้เราพร้อมที่จะ plot ด้านที่มองเห็น program ในส่วนนี้จะกำหนดส่วนของด้านที่มองเห็น และอ่านค่าจุดปลายของด้านที่มองเห็นนี้และก็วาดเส้นของด้านนี้ ขณะนี้เราจะได้รูปลูกบาศก์ในลักษณะการมองต่าง ๆ โดยการเปลี่ยนค่า Parameter จุดสังเกตุ ตามที่แสดงไว้ในรูป 5.5

5.2 The Visibility of several object

รูปแบบในหัวข้อที่แล้วจะวาดส่วนที่มองเห็นของวัตถุนูนเพียงอันเดียวเท่านั้น และในส่วนนี้เราจะแสดง Algorithm สำหรับวาดวัตถุที่มากกว่า 1 อัน พร้อมกับมีการนำเอาเส้นที่มองไม่เห็นออกจากภาพด้วย เมื่อเราจะวาดวัตถุมากกว่า 1 รูปขึ้นไป ก็ต้องมีความจำเป็นที่ต้องกำหนดส่วนของวัตถุที่ถูกบังโดยวัตถุที่อยู่ใกล้กว่า และต้องกำหนดส่วนเพิ่มเติมของแต่ละวัตถุที่มองเห็นด้วย

Algorithm สำหรับที่จะนำส่วนที่ถูกบังออกจากภาพของวัตถุสามารถทำได้ในรูปแบบของ object space และ Image space object space Algorithm นั้นจะใช้ความสัมพันธ์ทางเรขาคณิตระหว่างวัตถุแต่ละชิ้นในภาพเพื่อกำหนดส่วนที่ถูกบัง Image space Algorithm ทำงานบนภาพ 2 มิติ และต้องมีการกำหนดส่วนของ pixel ไหนที่มองเห็นทั้ง วิธีนี้ได้ถูกออกแบบมาเพื่อให้ทำงานบน raster scan device เราจะกล่าวถึง image space algorithm ในส่วนที่ 5.5

โดยทั่วไปแล้วทั้ง 2 วิธีจะใช้แยกกันและ algorithm ทั้งสองก็มีลักษณะการทำงานที่แตกต่าง ๆ กัน และ hardware ก็มีแตกต่างกันด้วย วิธีของ object space โดยพื้นฐานแล้วจะนำมาใช้ใน Hidden line algorithm ในขณะที่ image space นำมาใช้ใน hidden surface algorithm

Algorithm ที่สร้างขึ้นในส่วนนี้จะแยกเป็นส่วน ๆ คือ ส่วนที่ทำงานใน mode object space และอีกส่วนจะทำงานใน image space Algorithm ที่ใช้ใน object space จะกำหนดส่วนที่ถูกบังของแต่ละวัตถุ แต่ใช้ image

space ในการกำหนดความสัมพันธ์ระหว่างวัตถุที่มองเห็น รูปแบบทั้ง 2 อย่างนี้สามารถนำมารวมกันได้เพื่อให้ใช้ผลลัพธ์ที่ดียิ่งขึ้นในเครื่องคอมพิวเตอร์

3.2.1 Geometric computation

ลักษณะวิธีการกระทำกับพื้นผิวในส่วน 5.1 คือพื้นฐานของ Hidden line removal ที่จะนำมาสร้างในส่วนนี้ วิธีทางเราขาคณิตใน Hidden line removal ประกอบด้วยส่วนที่สำคัญ 3 ส่วนคือ

ส่วนที่ 1 เราจะใช้ visibility test ตรวจสอบวัตถุแต่ละชิ้นเพื่อที่จะหาพื้นผิวที่ถูกบัง

ส่วนที่ 2 เราจะใช้ลำดับความสัมพันธ์ต่าง ๆ ตรวจสอบเพื่อกำหนดความสัมพันธ์ที่เป็นไปได้ของพื้นผิวที่เห็นหมดในภาพนั้นมีการคัดกันหรือไม่ หรือวัตถุหนึ่งล้อมรอบอีกวัตถุหนึ่งหรือไม่มีการคัดกันเลย ในลำดับแรกนั้นที่ต้องทดสอบก่อนคือminimax test จุดประสงค์ของการทดสอบวิธีนี้คือ ต้องการตรวจสอบว่ารูปหลายเหลี่ยมคู่ใดบ้างที่ไม่ทับกัน

ส่วนที่ 3 ถ้าการทดสอบโดยวิธี minimax ตรวจสอบแล้วว่ารูปหลายเหลี่ยมที่โอกาสทับกัน เราก็ทำการตรวจสอบรูปหลายเหลี่ยมคู่นั้นต่อไปอีกเพื่อที่จะกำหนดว่าด้านของรูปหลายเหลี่ยมคัดกันหรือไม่ หรือรูปหลายเหลี่ยมรูปหนึ่งล้อมรอบรูปหลายเหลี่ยมอีกรูปหนึ่ง โดยการใช้ Intersetion ตรวจสอบเป็นลำดับต่อมาการตรวจเช็คนี้จะทำการตรวจสอบเส้นที่ละเส้นของรูปหลายเหลี่ยมนั่นคือด้านหนึ่งของรูปหลายเหลี่ยมที่ต้องการทดสอบจะต้องอยู่ในสภาวะที่มองเห็นด้วยถ้ามันตัดกับอีกเส้นหนึ่งที่เป็นด้านของรูปหลายเหลี่ยมอีกรูปหนึ่ง เราก็บอกได้ว่ารูปหลายเหลี่ยมคู่นี้ทับกันและต่อมาก็จะใช้ depth test (หรือเรียกอีกชื่อหนึ่งว่า Priority test) เพื่อกำหนดว่ารูปหลายเหลี่ยมไหนจะเห็นหมดและรูปหลายเหลี่ยมไหนจะเห็นเป็นบางส่วนจากจุดสังเกตุนั้น ถ้าไม่มีการตัดกันระหว่างรูปหลายเหลี่ยมทั้งสองแล้วก็มีโอกาสเป็นไปได้อีกกรณีต่อไปนี้คือ รูปหลายเหลี่ยมรูปหนึ่งล้อมรอบรูปหลายเหลี่ยมอีกรูปหนึ่ง หรือรูปหลายเหลี่ยมทั้งสองไม่ทับกันเลย ในการทดสอบว่ารูปหลายเหลี่ยมรูปใดล้อมรอบรูปหลายเหลี่ยมอีกรูปหนึ่ง เราจะใช้ containmant test ถ้าการตรวจสอบปรากฏว่ารูปหลายเหลี่ยมหนึ่ง

ล้อมรอบรูปหลายเหลี่ยมอีกรูปหนึ่ง Polygon ที่ถูกล้อมรอบจะมองไม่เห็น

สิ่งสำคัญที่ต้องจำไว้คือ การกระทำแบบ visibility test จะต้องกระทำใน object space ในขณะที่ความสัมพันธ์ทั้งหลาย เช่น Intersettion test, depth test ถูกกระทำใน image space

รูป 5.6 จะสรุปให้เห็นลำดับขั้นตอนการทดสอบที่เกี่ยวข้องในการสร้าง algorithm ที่จะกำจัดส่วนที่ถูกบังออกไปในส่วนถัดไป เราจะกล่าวถึงการทดสอบด้วยวิธีการเหล่านี้โดยละเอียดยิ่งขึ้น และเราได้กล่าวถึง visibility test ในส่วนที่ 5.1 แล้ว ต่อไปเราจะเริ่มต้นด้วย

Minimax test

พิจารณาที่รูปหลายเหลี่ยม A และ B ในระนาบภาพ ซึ่งได้มาจากการ projection ของรูปหลายเหลี่ยมทั้งสองของวัตถุ 3 มิติ ดูในรูป 5.7 สำหรับในแต่ละรูปหลายเหลี่ยมนั้นเราจะหาค่าสูงสุด และค่าต่ำสุด X และ Y ต่อจากนั้นก็นำค่าสูงสุดและต่ำสุดของรูปหลายเหลี่ยมทั้งสองมาทำการเปรียบเทียบกัน และถ้ามีเงื่อนไขหนึ่งเงื่อนไขใดเป็นจริงตามข้อความที่ให้ไว้ข้างล่างนี้ รูปหลายเหลี่ยมทั้ง 2 จะไม่ทับกัน เงื่อนไขก็คือ

$$\begin{aligned} MAX\ XA &< MIN\ XB \\ MAX\ XB &< MIN\ XA \\ MAX\ YA &< MIN\ YB \\ MAX\ YB &< MIN\ YA \end{aligned} \qquad (5.5)$$

เมื่อ Xj = coordinate X ของรูปหลายเหลี่ยม j (j = A,B)

Yj = coordinate Y ของรูปหลายเหลี่ยม j (j = A,B)

เราจะใช้ Minimax ตรวจสอบแต่ละคู่ของรูปหลายเหลี่ยมดังในรูป 5.8 การทดสอบจะแสดงให้เห็นว่า รูปหลายเหลี่ยมแต่ละคู่มีโอกาสทับกัน อย่างไรก็ตามคู่ของรูปหลายเหลี่ยม 5.8(d) และ 5.8 (e) จะไม่มีการทับกันอย่างแน่นอน เนื่องจากการทดสอบแบบนี้ยังไม่สิ้นสุด เราต้องมีการทดสอบต่อไปอีกเพื่อที่จะกำหนดให้ได้ว่าด้านเหล่านั้นมีการตัดกันหรือไม่หรือรูปหลายเหลี่ยมมีการล้อมรอบหรือไม่ และเนื่องจากการตรวจสอบแบบ Intersettion มีความยุ่งยากมากกว่า ดังนั้นการตรวจสอบแบบนี้จึงกระทำต่อจาก minimax test ที่ทำการ

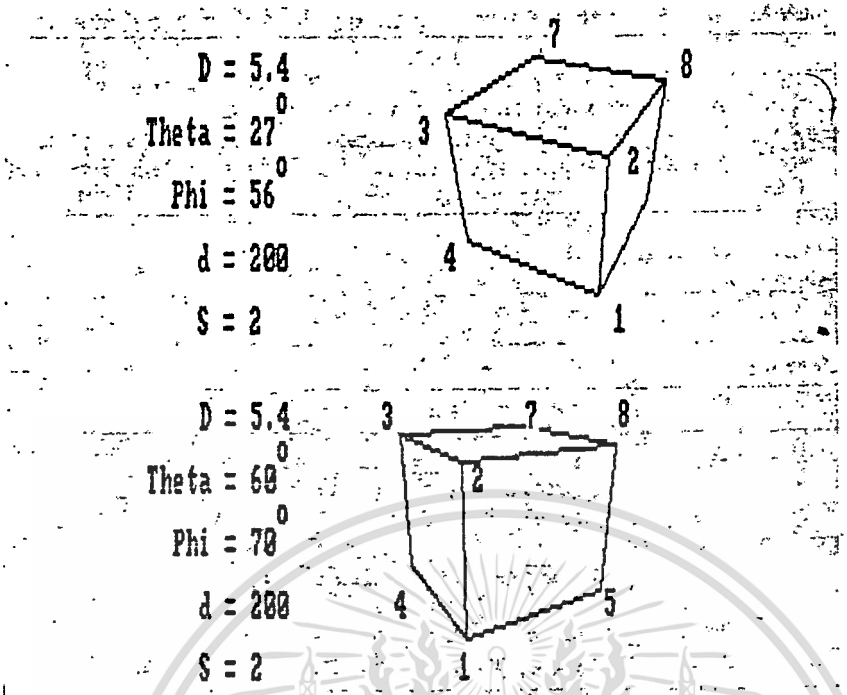


FIGURE 5.5 Two different views of the cube with hidden lines removed

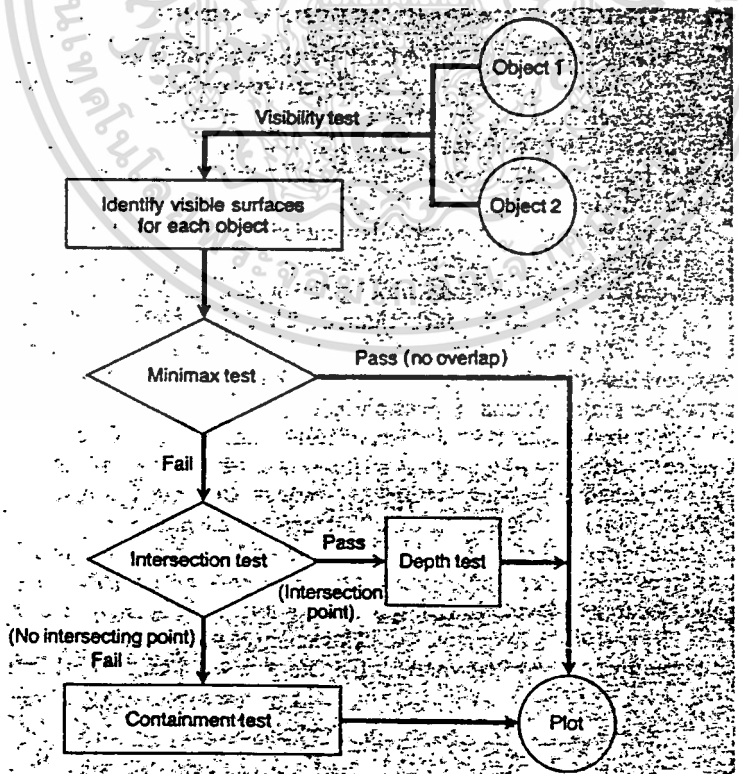


FIGURE 5.6 Logical sequence of tests used in hidden line elimination

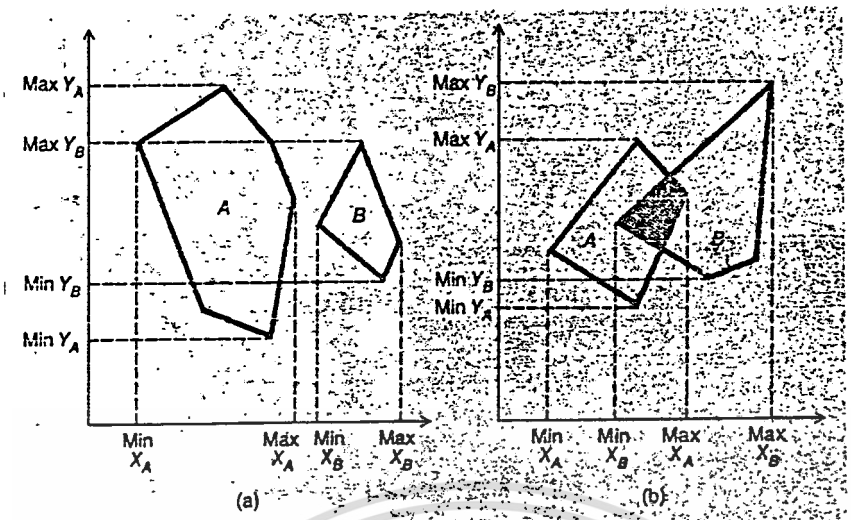


FIGURE 5.7 Minimax test: (a) cannot overlap and (b) overlap possible but not proven

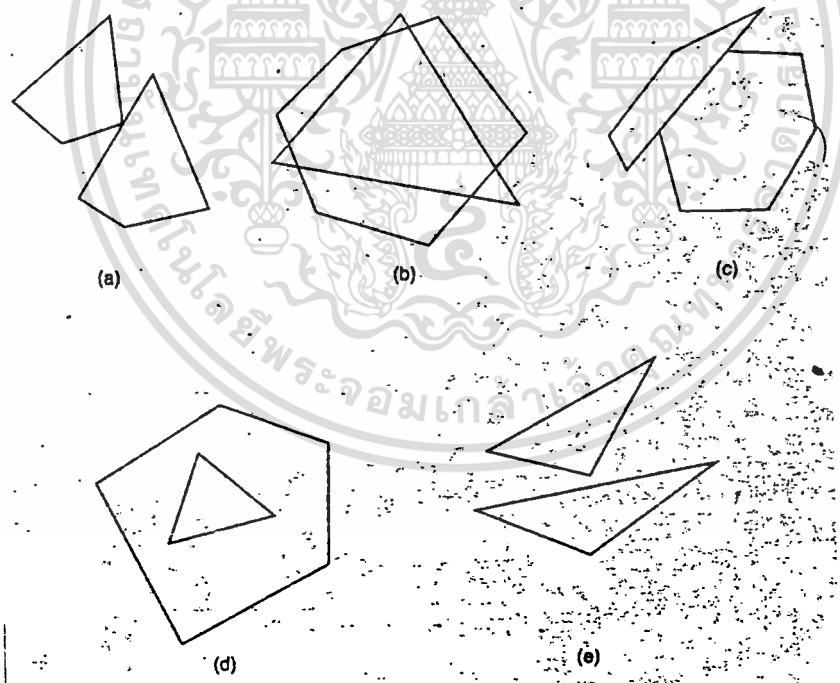


FIGURE 5.8 Possibilities of determining the distinguished point in the intersection of two polygons: (a) unique point, (b) multiple points, (c) infinitely many points, (d) nested set (no point), and (e) no overlap

ตรวจสอบแล้วว่ารูปหลายเหลี่ยมคู่ใดมีโอกาสจะตัดกันหรือล้อมรอบกัน

5.2.3 Intersection test

จากพีชคณิตเชิงเส้นเราสามารถคำนวณจุดตัดระหว่างเส้นตรงทั้งสองได้ และสำหรับการตัดกันของเส้นตรงทั้งสองนี้ เราจะแสดงสูตร 2 สูตร ซึ่งมีประโยชน์ในการตรวจสอบแบบ Intersection test

Intersection of two line สมมุติเรามีสมการเส้นตรง 2 เส้น คือ 1, 1'

$$L_1 = A_1X + B_1X + C_1 = 0 \qquad (A_1, B_1 \text{ต้องไม่เป็น } 0)$$

$$L_2 = A_2X + B_2X + C_2 = 0 \qquad (A_2, B_2 \text{ จะต้องไม่เป็น } 0) \qquad (5.6)$$

สมการทั้ง 2 เส้นนี้จะทำให้เกิดเงื่อนไขที่เป็นไปได้ 3 แบบคือ

1. เส้น 1,1' อาจจะขนานกันซึ่งในกรณีนี้จะไม่เกิดการตัดกัน เกิดขึ้นเงื่อนไขในรูปแบบนี้จะเกิดขึ้นได้ดังนี้

$$\begin{bmatrix} A_1 & B_1 \\ A_2 & B_2 \end{bmatrix} = A_1B_2 - B_1A_2 = 0$$

2. เส้น 1,1' อาจจะทับกันซึ่งในกรณีนี้จะมีจุดตัดเกิดขึ้นอย่างมากมาย เงื่อนไขนี้จะเกิดขึ้นเมื่อ

$$A_1/A_2 = B_1/B_2 = C_1/C_2$$

3. ถ้าไม่ใช่เงื่อนไขในกรณีทั้ง 2 ก็จะต้องมีจุดตัด (Xi, Yi) เกิดขึ้นซึ่ง Xi และ Yi จะหาได้ดังนี้

$$Xi = \frac{B_1C_2 - B_2C_1}{A_1B_2 - A_2B_1}$$

$$Yi = \frac{C_1A_2 - C_2A_1}{A_1B_2 - A_2B_1} \qquad (5.7)$$

Intersection of two point

สมมุติว่าเราจะกำหนดเส้นตรง G กับ G' บนเส้นตรง 1 และ 1' ตามลำดับส่วนของเส้นตรง G จะถูกกำหนดด้วยจุดปลาย 2 จุดคือ (X_1, Y_1) และ (X_2, Y_2) และส่วนของเส้นตรง G' จะถูกกำหนดด้วยจุด (X_1', Y_1') และ

(X'_2, Y'_2) ให้ 1 และ 1' ตัดกันที่จุด i ที่ coordinate(Xi,Yi) ตามที่กล่าวมาแล้วในแต่ก่อน เส้นตรง G กับ G' จะไม่ตัดกันที่จุด i ถ้ามีเงื่อนไขหนึ่งเป็นไปตามเงื่อนไขข้างล่าง

$$Xi < A_1 \ , \ Xi > B_1 \ , \ Xi < C_1, \ Xi > D_1$$
$$Yi < A_1 \ , \ Yi > B_2 \ , \ Xi < C_2, \ Xi > D_2 \qquad (5.8)$$

เมื่อ $A_1 = \min(X_1, X_2)$, $B_1 = \max(X_1, X_2)$, $C_1 = \min(X_1, X_2)$ $D_1 = \max(X_1, X_2)$

$A_2 = \min(Y_1, Y_2)$, $B_2 = \max(Y_1, Y_2)$, $C_2 = \min(Y_1, Y_2)$ $D_2 = \max(Y_1, Y_2)$

ถ้าไม่เข้าเงื่อนไขใดเส้นตรง G และ G' ก็จะตัดกัน ซึ่งแสดงให้เห็นในรูป 5.9 และเราใช้กฎเหล่านี้เป็นเงื่อนไขในการตรวจสอบแบบ Intersection test

Edge intorsection test

หน้าที่ของเราไม่เพียงจะทดสอบว่ามีการตัดกันระหว่างด้านหรือไม่แต่เรายังต้องหาจุดตัดระหว่างด้านของรูปหลายเหลี่ยมนั้น ๆ โดยที่ด้านหนึ่งจะถูกกำหนดโดยจุดปลาย 2 จุด เราจะสร้างสมการของด้านได้โดยการกำหนดจากจุดปลายทั้งสองของด้านนั้น ๆ

ให้จุด (X_1, Y_1) และ (X_2, Y_2) เป็นจุดปลายทั้งสองของด้านหนึ่งของรูปหลายเหลี่ยม A สมการของด้านนี้จะหาได้ดังนี้

$$Y = Y_1 + M(X - X_1)$$
$$M = (Y_2 - Y_1) \ / \ X_2 - X_1 \qquad 5.9 \ (a)$$

หรืออาจจะแสดงในรูปแบบอีกอย่างคือ

$$MX - Y + (Y_1 - MX_1) = 0$$

ในทำนองคล้ายกันจุด (X'_1, Y'_1) และ (X'_2, Y'_2) เป็นจุดปลายของด้านของรูปหลายเหลี่ยม B ดังนั้นสมการของด้านนี้คือ

$$M'X - Y + (Y'_1 - M'X'_1) = 0 \qquad 5.9 \ (b)$$

เมื่อ $M' = (Y'_2 - Y'_1) \ / \ (X'_2 - X'_1)$

ในการหาจุดตัดเราต้องดำเนินขั้นตอนตามนี้คือ

ขั้นตอนที่ 1 คำนวณหา Determinant

$$\begin{bmatrix} M & -1 \\ M' & -1 \end{bmatrix} = M' - M$$

ถ้า $M'-M=0$ ด้านทั้งสองจะขนานกันไปทำขั้นตอนที่ 5 ถ้าไม่เป็นไปตาม เงื่อนไขนี้ทำขั้นตอนที่ 2

ขั้นตอนที่ 2 ถ้า $M/M'=1=(Y_1-MX_1)/(Y_1'-M'X_1')$ ด้านทั้งสองจะทับกันไปทำขั้นตอนที่ 5 ถ้าไม่เป็นไปตามเงื่อนไขนี้ทำขั้นตอนที่ 3

ขั้นตอนที่ 3 คำนวณหาจุดตัด Xi Yi โดยใช้สมการ (5.7)

$$Xi = \frac{(Y_1-Y_1')-(MX_1-M'X_1')}{M'-M} \qquad 5.9\ (c)$$

$$Yi = \frac{(Y_1-MX_1)M'-(Y_1'-M'X')\,M}{M'-M}$$

ขั้นตอนที่ 4 ใช้กลุ่มเงื่อนไขในสมการ 5.8 เพื่อที่จะหาว่าด้านทั้งสองตัดกันที่จุด XiYi จริงหรือไม่

ขั้นตอนที่ 5 เลือกด้านอื่นของรูปหลายเหลี่ยม B และทำตามขั้นตอนที่ 1 ถึง 5 ต่อไปอีก ถ้าไม่มีด้านใดแล้วหยุด

5.2.4 Containment test

เมื่อใช้ Intersection test ตรวจสอบแล้วไม่มีจุดตัดระหว่างรูปหลายเหลี่ยมทั้งสอง ก็อาจเป็นไปได้อีก 2 กรณี คือ

1. รูปหลายเหลี่ยมรูปหนึ่งล้อมรอบอีกรูปหนึ่ง
2. รูปหลายเหลี่ยมทั้งสองไม่ทับกัน

เพื่อที่จะให้รู้ว่าเข้าในกรณีที่ 1 หรือกรณีที่ 2 เราต้องตรวจสอบจุดทั้งหมดของรูปหลายเหลี่ยมรูปหนึ่งดูว่าล้อมรอบรูปหลายเหลี่ยมอีกรูปหนึ่งหรือไม่ โดยเราจะใช้กฎเกณฑ์ทางคณิตศาสตร์ รูปแบบในการทดสอบเป็นดังนี้

Test by Calculating a sun at angle

เราจะคำนวณหาผลรวมของจุดใดจุดหนึ่ง (จุดทดสอบ) ของพื้นผิวหลายเหลี่ยม A โดยใช้ด้านทุกด้านที่เชื่อมโยงจากจุดทุกจุดของรูปหลายเหลี่ยม B (ดูรูป 5.10) ให้ $Pj=(Xj,Yj)$ โดย Pj คือจุดต่างของรูปหลายเหลี่ยม

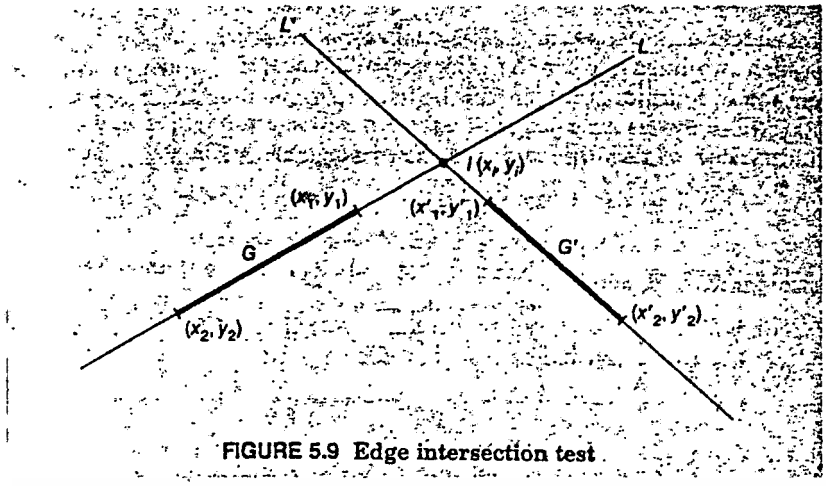


FIGURE 5.9 Edge intersection test

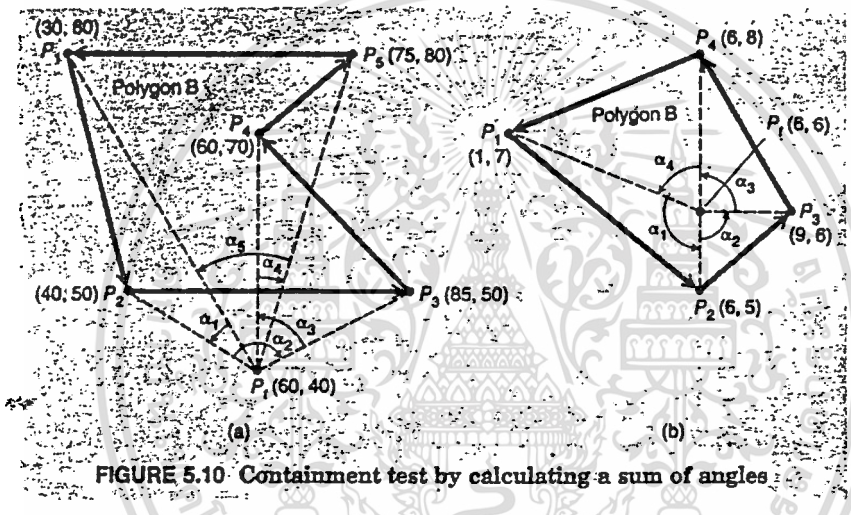


FIGURE 5.10 Containment test by calculating a sum of angles

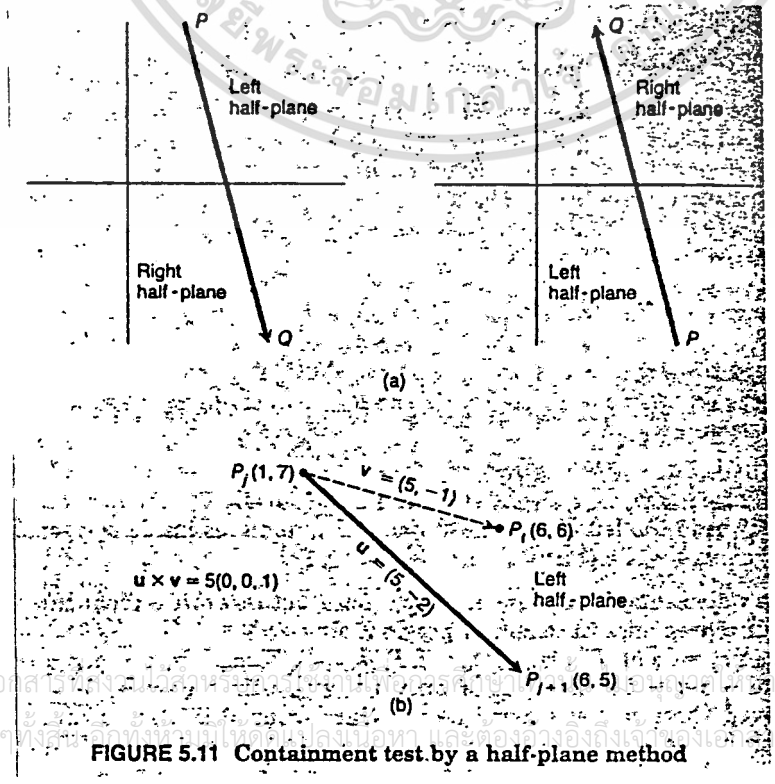


FIGURE 5.11 Containment test by a half-plane method

เมื่อ $j = 1,2,3,\ldots. n$ และ $Pn=P_1$ เราจะนับจุดของรูปหลายเหลี่ยมในทิศทางทวนเข็มนาฬิกา ให้ Pt เป็นจุดที่จะถูกทดสอบด้วยรูปหลายเหลี่ยม

$PtPi$ แสดงส่วนของเส้นตรงที่ต่อระหว่าง Pt กับ Pi ให้ i คือมุมระหว่าง $PtPj$ และ $PtPj+1$ และถูกวัดจาก $PtPj$ ไปยัง $PtPj+1$ เราจะต้องคำนึงถึงเครื่องหมายในการวัด ค่ามุม j เช่นตัวอย่างถ้าเราวัดมุมในทิศทางตามเข็มนาฬิกาเราจะกำหนดให้มีเครื่องหมายเป็นลบ ถ้าเราวัดมุมในทิศทางทวนเข็มนาฬิกา เราจะกำหนดให้ค่ามุมเป็นบวก เราจะคำนวณหาค่า i ของทุกค่า j และต่อจากนั้นก็คำนวณหาผลรวมทั้งหมดของ j ก็เป็นอันสิ้นสุดวิธีการของ Containment test

ถ้า $\Sigma xi = 0$ จุด Pt อยู่ภายนอกรูปหลายเหลี่ยม B

ถ้า $\Sigma xi = 360^\circ$ จุด Pt อยู่ภายในรูปหลายเหลี่ยม B

เพื่อที่จะแสดงให้เห็นขั้นตอนต่าง ๆ ให้เราพิจารณารูป 5.10 (a) เราจะคำนวณหาค่า i สมมุติว่าจุด P_1 (30,80), P_2(40,50), P_t (60,40) ลำดับแรกต้องหา vector ทั้งสอง PtP_1 และ PtP_2 และความยาว PtP_1 และ PtP_2

$$PtP_1 = (30,80) - (60,40) = (-30,40)$$

$$PtP_2 = (40,50) - (60,40) = (-20,10)$$

$$PtP_1 = (-30)^2 + 40^2 = 50$$

$$PtP2 = (.20)^2 + (10)^2 = 22.36$$

ต่อจากนั้นก็หาค่า dot product ของ vector ทั้งสอง

$$PtP1 \cdot PtP2 = (-30,40) \cdot (-20,10) = 100$$

แล้วใช้สมการ (5.4) คำนวณหา $\alpha 1$ ได้ดังนี้

$$\alpha 1 = \cos^{-1} \frac{1000}{50(22.36)} = 26.56^\circ$$

เราหาค่า 2 ในวิธีเดียวกันกับข้างบนนี้ มุม 2 ถูกวัดในทิศทางตามเข็มนาฬิกา เพราะฉะนั้นค่ามุมที่ได้เป็นมุมลบ ซึ่งผลการคำนวณ 2 มีค่า = 131.63° และเราก็ดำเนินกรรมวิธีดังกล่าวนี้กับจุดที่เหลือ j และก็หาค่าผลรวม $j= 0$ ก็หมายความว่าจุดทดสอบ Pt อยู่ภายนอกรูปหลายเหลี่ยม

Containment จะถูกทำซ้ำกับจุดแต่ละจุดของรูปหลายเหลี่ยม A รูปหลายเหลี่ยม A ถูกล้อมรอบด้วยรูปหลายเหลี่ยม B เมื่อจุดทุกจุดของ A อยู่ภาย

ใน B แต่ผลที่ได้นี้ยังไม่สิ้นสุดเมื่อทุกจุดของ A อยู่ภายนอก B อาจจะเป็นได้ที่รูปหลายเหลี่ยม A ล้อมรอบรูปหลายเหลี่ยม B ในกรณีนี้เราจึงต้องมีขั้นตอนที่ต้องทำสลับกันเพื่อตรวจสอบว่า รูปหลายเหลี่ยม A ล้อมรอบรูปหลายเหลี่ยม B หรือไม่

Test by half plane method

วิธีนี้เป็นอีกวิธีหนึ่งที่ต้องกำหนดว่า half plane ไหนของรูปหลายเหลี่ยม B ที่มีจุดทดสอบตั้งอยู่ (เป็นจุดในรูปหลายเหลี่ยม A) จาก Analytic geometry เรารู้ว่าเส้นตรงที่ต่อจากจุด P และ Q จะแบ่งระนาบ XY ออกเป็น 2 ส่วนคือ half plane ที่อยู่ทางด้านซ้ายมือเมื่อมองตามแนว vector Pq จะถูกกำหนดให้เป็น left half plane ถ้าอยู่ทางด้านขวาจะถูกกำหนดให้เป็น right half plane

เราอาจจะมองแต่ละด้านของรูปหลายเหลี่ยมเป็น vector ที่แบ่งระนาบ XY และเราจะใช้จุดทดสอบ Pt ทำการกำหนดว่า half plane ทางด้านไหนที่จุด Pt ตั้งอยู่ ซึ่งสามารถทำได้โดย

ลำดับแรก ต้องกำหนด vectorขึ้นมา 2 vector คือ vector U และ vector V ตามที่แสดงในรูป 5.11 โดยที่ vector V นั้นจะเป็น vector ที่ได้จากการต่อจุด Pj กับ Pt โดยที่ Pt เป็นจุดทดสอบส่วน vectorV คือ vector ที่ต่อจากจุด Pj ไปยัง Pj+1

ลำดับที่ 2 เป็นการกำหนดว่าจุด P อยู่ทาง half plane ใด

- ถ้า Pt อยู่ทาง right half plane ของ vector PjPj+1 (อยู่ด้านนอกรูปหลายเหลี่ยม B) เราจะได้ว่า UxV ทำให้เกิดค่าสัมประสิทธิ์ตัวคูณ (0,0,1) มีค่าเป็นลบ

- ถ้า Pt อยู่ทาง left half plane ของ vector PjPj+1 (อยู่ด้านในรูปหลายเหลี่ยม B) เราจะได้ว่า UxV ทำให้เกิดค่าสัมประสิทธิ์ตัวคูณ (0,0,1) มีค่าเป็นบวก

รูปแบบที่กำหนดนี้เป็นกฎเกณฑ์ที่ถูกต้องตามแบบคณิตศาสตร์ แต่เราจะแสดงรูปแบบของค่าเหล่านี้ในรูปแบบที่แตกต่างจากนี้เพื่อให้เกิดความสะดวกใน Computer Programming

ตัวอย่าง ให้ Pj=(a,b),Pj+1=(e,f) และ Pt=(c,d) จะได้ vector U=(e-a,f-b) และ vector V =(c-a,d-b) แต่ค่า cross product จะถูกนำมาใช้ใน vector 3 มิติเท่านั้น เพราะเหตุนี้เราต้องมีการขยาย vector U และ V ให้อยู่ในรูปแบบ 3D โดยการเพิ่มค่า 0 ใน coordinate Z ในแต่ละ vector และรูปแบบของ 2D vector บนระนาบจะมีค่าเท่ากับ 3D vector ที่มีค่า Z coordinate มีค่าเป็น 0 ทำให้ได้ค่า vector U และ vector V ดังนี้

$$U = (e-a, f-b, 0)$$
$$V = (c-a, d-b, 0)$$

จากสมการ 5.1 ค่า cross product ที่ได้คือ

$$UxV = (0,0,(e-a)(d-b)-(f-b)(c-a)) \qquad (5.10)$$

เงื่อนไขของ cross product ที่มีค่าสัมประสิทธิ์ตัวคูณมีค่าเป็นบวกของค่า (0,0,1) คือ

$$(e-a)(d-b) > (f-b)(c-a)$$

ฉะนั้นจึงสรุปได้ว่า

- ถ้า $(e-a)(d-b) > (f-b)(c-a)$ จะได้ว่าจุด Pt อยู่ทาง left half plane ของ vector PjPj+1
- ถ้า $(e-a)(d-b) < (f-b)(c-a)$ จะได้ว่าจุด Pt อยู่ทาง right half plane vector PjPj+1

Containment มีเนื้อหาเพียงเท่านี้, เราจะต้องดำเนินวิธีนี้ซ้ำอีกโดยใช้ด้านที่เหลือของรูปหลายเหลี่ยม B และด้านของรูปหลายเหลี่ยมจะต้องมีทิศทางทวนเข็มนาฬิกาเท่านั้น เราจะสรุปได้ว่า

- Pt อยู่ภายในรูปหลายเหลี่ยม B ถ้าจุด Pt อยู่ทาง left half plane ของแต่ละด้านของรูปหลายเหลี่ยมนั้น
- Pt อยู่ภายนอกรูปหลายเหลี่ยม B ถ้าจุด Pt อยู่ทาง right half plane อย่างน้อยที่สุด 1 ด้านของพื้นผิวหลายเหลี่ยมนั้น

เพื่อแสดงให้เห็นวิธีคำนวณ โดยจะพิจารณาจากรูป 5.10 b สำหรับ P1P2 และจุดทดสอบ Pt เราสามารถคำนวณหาค่า cross product ของ vector ทั้ง 2 คือ U และ V

$$V = (6-1, 6-7) = (5,-1)$$

$$U = (6-1, 5-7) = (5, -2)$$

และ $UxV = (5,-2,0)x(5,-1,0) = (0,0,5)$

$$= 5(0,0,1)$$

ซึ่งในที่นี้ 5 เป็นสัมประสิทธิตัวคูณของค่า (0,0,1) มีค่าเป็นบวกเราก็สามารถกำหนดรูปแบบจากรูป 5.11 (b) ว่าจุด Pt อยู่ทางด้าน left half plane ต่อไปเราก็ใช้วิธีนี้ตรวจสอบ P2P3, P3P4, P4P1 และจากการตรวจสอบแล้วจุด Pt อยู่ใน left half plane สำหรับแต่ละด้านทดสอบ ฉะนั้นจุด Pt จึงอยู่ภายในรูปหลายเหลี่ยม B

วิธีของ half plane เป็นวิธีที่น่าสนใจมากกว่าการหาผลรวมของมุม โดยเหตุนี้เราจึงรับเอาวิธีนี้มาใช้ใน hidden line algorithm แต่มีข้อควรจำอย่างคือวิธี half plane นี้จะนำมาใช้กับรูปหลายเหลี่ยมนูนเท่านั้น (รูปหลายเหลี่ยมในรูป 5.10 (a) ไม่เป็นรูปหลายเหลี่ยมนูน ดังนั้นวิธี half plane จะไม่สามารถนำมาใช้กับรูปหลายเหลี่ยมดังกล่าวนี้ได้

5.2.5 Depth test

ในส่วนที่ 4.4 ได้แนะนำวิธีการหาค่าความลึกของแต่ละจุดในภาพ Persective แล้วโดยที่ค่าความลึก Zs จะหาได้จากการ Transform plane ในระบบeye coordinate ไปเป็นระบบ screen coordinate

Plane equation

ตามที่ได้เห็นมาแล้วจากสมการ (4.35) และ (4.36), ขั้นตอนแรกใน depth test คือกำหนดหาค่าตัวสัมประสิทธิที่ไม่รู้ค่าของ plane equation ที่กำหนดรูปหลายเหลี่ยมนั้นและในการหาจะทำได้ดังนี้ ให้จุด Pj(xi,yi,zi),j=1,2,3 เป็นจุดแต่ละจุดบนรูปหลายเหลี่ยม เราจะใช้ vector 2 vector คือ U และ V รวมกันเป็นระนาบ (ซึ่งประกอบด้วย

Surface Normal Vector และต้องนับจุดทั้ง 3 ในทิศทางทวนเข็มนาฬิกา)

$$U=P_1P_2=(X_2,Y_2,Z_2)-(X_1,Y_1,Z_1)=(X_2-X_1,Y_2-Y_1,Z_2-Z_1)$$

$$V=P_1P_3=(X_3,Y_3,Z_3)-(X_1,Y_1,Z_1)=(X_3-X_1,Y_3-Y_1,Z_3-Z_1)$$

และรู้จาก Plane analytic geometry ว่า cross product UxVจะทำให้ได้ค่าสัมประสิทธิของระนาบผ่านจุด (X_1,Y_1,Z_1)จากสมการระนาบ

$$AX + BY + CZ + D = 0 \qquad (5.11)$$

ค่าสัมประสิทธิ A,B,C จะหาได้จาก UxV , $(A,B,C) = U\times V$ (5.12)

และใช้สมการ 5.1 เราก็จะหาค่าสัมประสิทธิทั้ง 3 ได้ดังนี้

$$A = Y_1(Z_2-Z_3) + Y_2(Z_3-Z_1) + Y_3(Z_1-Z_2)$$

$$B = Z_1(X_2-X_3) + Z_2(X_3-X_1) + Z_3(X_1-X_2)$$

$$C = X_1(T_2-Y_3) + X_2(Y_3-Y_1) + X_3(Y_1-Y_2)$$

และค่า D จะหาได้ดังนี้

$$D = -AX_1-BY_1-CZ_1 \qquad (5.13)$$

Priority

การคำนวณเพียง 2 ครั้งก็เพียงพอที่จะแสดงว่ารูปหลายเหลี่ยมที่มีการทับกันนั้น รูปหลายเหลี่ยมรูปไหนจะอยู่ใกล้จุดสังเกตุมากกว่ากัน การคำนวณในครั้งแรกจะคำนวณหาจุดตัดของรูปหลายเหลี่ยมนั้น (รูปหลายเหลี่ยมอาจตัดกันมากกว่า 1 จุด แต่เราต้องการเพียงจุดเดียว มาใช้ในการตรวจสอบ โดยวิธี depth test) การคำนวณในครั้งที่ 2 คือการหาค่า Zs ของรูปหลายเหลี่ยมที่มีการตัดกันและการคำนวณจะมีลักษณะตามนี้

- ให้ Xi Yi เป็น screen coordinate ของจุดตัดนั้น
- กำหนด plane equation ที่แสดงถึงรูปหลายเหลี่ยมนั้นในระบบ screen coordinate เช่น

$a_1Xs + b_1Ys + c_1Zs + d = 0$ รูปหลายเหลี่ยมที่ 1

$a_2Xs + b_2Ys + c_2Zs + d = 0$ รูปหลายเหลี่ยมที่ 2

- แทนค่า Xs=Xi และ Ys=Yi เพื่อที่จะหาค่า Z
- เปรียบเทียบค่า Zs โดยที่รูปหลายเหลี่ยมใดที่มีค่า Zs น้อยกว่ารูปหลายเหลี่ยมนั้นจะอยู่ใกล้จุดสังเกตุมากกว่า และรูปหลายเหลี่ยมนั้นจะเห็นโดยตลอด

ในส่วนต่อไป เราจะแสดงลำดับขั้นการทำงานและวิธีการคำนวณของ hidden line algorithm โดยการสร้าง computer program.

5.3 Program development of drawing two convex object

ใน program จะใช้วัตถุ 2 ชิ้นคือลูกบาศก์และปิรามิด ดังรูป 5.12 และแสดงวิธีการคำนวณซึ่งจำเป็นในการสร้าง hidden line algorithm ดังที่กล่าวในส่วน 5.2 โปรแกรมนี้ยาวมาก ดังนั้นเราจะแสดงโปรแกรมนี้เป็นหลาย ๆ ส่วน และอธิบายวัตถุประสงค์ของแต่ละส่วน

5.3.1 The main program

โปรแกรม 5.2 มีโปรแกรมย่อย 12 โปรแกรม และยังมีโปรแกรมย่อยซ้อน 5 โปรแกรม โดยพื้นฐาน main program ควบคุมและกำหนดทิศทางขั้นตอนตามการตรวจสอบในแต่ละกรณี รูป 5.13 คือลักษณะขั้นตอนใน Hidden line algorithm รายละเอียดของโปรแกรมถูกแสดงอยู่ในหน้า 207–210

5.3.2 The viewing parameter

เราจะสังเกตุปิรามิดและลูกบาศก์จากจุดสังเกตุ $(D,\theta,\phi) = (10, 57^\circ, 68^\circ)$ ในระบบพิกัดฉากจุดสังเกตุนี้จะถูกตั้งในตำแหน่ง (5.05, 7.78, 3.75) โดยตั้งตำแหน่งจอภาพที่ $d = 50c$ ส่วนค่า view parameter D,θ,ϕ และ d นั้นใช้ควบคุมตำแหน่งและทิศทางการมองและขนาดของภาพ ใน program จะ plot วัตถุที่มีรูปทรงง่าย ๆ และจะ set window และ view point ให้มีขนาดเท่ากัน (ดูในสมการ 3.39 และ 3.40) และใช้ใน mode high resolution ส่วน Scaling factor (SF) = 2.4

5.3.3 Vertex, eye, and screen coordinate arrays

ตอนแรก กำหนดให้ NV(I) เป็นจำนวนจุดที่มีอยู่ในวัตถุแต่ละอัน ปิรามิดจะมี 4 จุดและลูกบาศก์จะมี 8 จุด ตำแหน่งของจุดเป็นดังรูป 5.12 ต่อมากำหนด parameter K1,K2 โดยจะเลือกค่า K1 และ K2 ให้ครอบคลุมภาพทั้งหมด, ด้วยการคำนวณหาค่า Ze coordinate สำหรับจุดแต่ละจุด และหาค่า

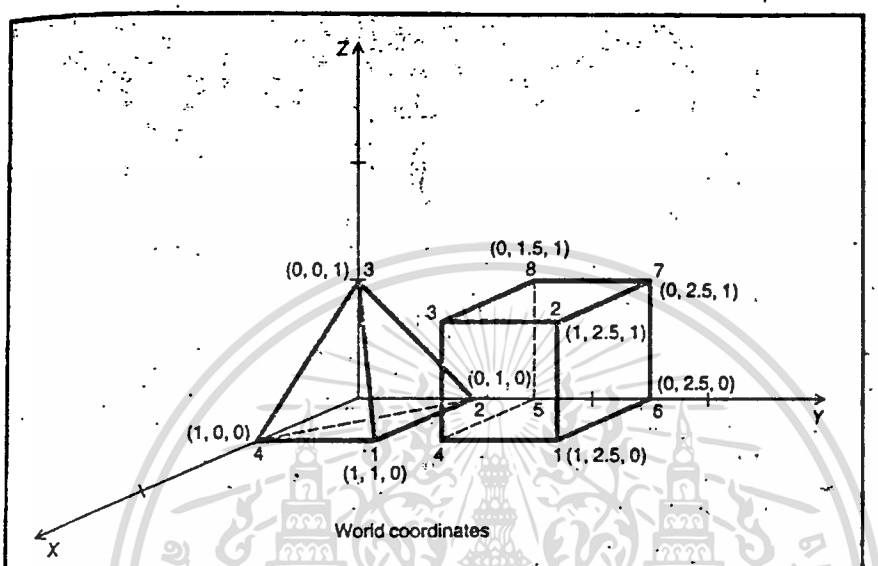


FIGURE 5.12 Two convex objects, pyramid and cube, defined in the world coordinate system

```
10 ' Program 5.2 : Hidden Line Elimination - Two Objects
20 CLEAR
30 CLS : KEY OFF : SCREEN 2
40       GOSUB 170
50       GOSUB 230
60       GOSUB 500
70       GOSUB 670
80       GOSUB 810
90       GOSUB 940
100      GOSUB 1140
110      IF OVERLAP=0 THEN 120 ELSE 130
120      FOR LL=1 TO 2 :GOSUB 1300:NEXT LL: END
130      GOSUB 1390
140      GOSUB 2090
150      GOSUB 2240
160   END
170 REM : DEFINE VIEWING PARAMETERS
180      D=10 : VD=500 : THETA=57 : PHI=68 : TX= 320: TY=100
190      PI=3.141593    :SCF=2.4
200      THETA=THETA*PI/180 : PHI=PHI*PI/180   : S = 1.5
210      SN1=SIN(THETA) : SN2=SIN(PHI) :CN1=COS(THETA) :CN2=COS(PHI)
220 RETURN
230 REM : VERTEX ARRAY
240 NV(1)=4 : NV(2)=8 : K1=7 : K2=10 : W=K2/(K2-K1)
250 DIM V(2,8,3), EC(2,8,3), SC(2,8,3)
260 FOR I=1 TO 2
270     FOR J=1 TO NV(I)
280          READ X,Y,Z
290          V(I,J,1)=X: V(I,J,2)=Y: V(I,J,3)=Z
300          GOSUB 400
```

PROGRAM 5.2 Hidden Line Elimination—Two Objects

```
310           EC(I,J,1)=XE: EC(I,J,2)=YE: EC(I,J,3)=ZE
320           GOSUB 450
330           SC(I,J,1)=  XS    : SC(I,J,2)= YS    : SC(I,J,3)=ZS
340       NEXT J
350 NEXT I
360 RETURN
370 DATA 1,1,0,0,1,0,0,0,1,1,0,0
380 DATA 1.2,5.0, 1.2,5.1, 1.1,5.1, 1.1,5.0
390 DATA 0,1,5.0, 0.2,5.0, 0.2,5.1, 0,1,5.1
400 REM : EYE COORDINATES
410       XE= -X*SN1 + Y*CN1
420       YE= -X*CN1*CN2 - Y*SN1*CN2 + Z*SN2
430       ZE= -X*SN2*CN1 - Y*SN2*SN1 - Z*CN2 + D
440 RETURN
450 REM : SCREEN COORDINATES
460       XS=(VD/S)*(XE/ZE)                          ' See Eq.(4.31)
470       YS=(VD/S)*(YE/ZE)                          ' See Eq.(4.32)
480       ZS=W*(1- K1/ZE)                            ' See Eq.(4.38)
490 RETURN
500 REM : SURFACE ARRAY
510 DIM SF(2,6,5),NPS(2,6)
520 NS(1)=4 :NS(2)=6
530 FOR I=1 TO 2: FOR J=1 TO NS(I): READ NPS(I,J): NEXT J: NEXT I
540 FOR I=1 TO 2
550       FOR J=1 TO NS(I)
560           FOR K=1 TO NPS(I,J)
570               READ SF(I,J,K)
580           NEXT K
590       NEXT J
600 NEXT I
610 RETURN
620 DATA 4,4,4,4
630 DATA 5,5,5,5,5,5
640 DATA 1,2,3,1, 2,4,3,2, 4,1,3,4, 1,4,2,1
650 DATA 1,2,3,4,1, 6,7,2,1,6, 5,8,7,6,5, 4,3,8,5,4
660 DATA 4,5,6,1,4, 7,8,3,2,7
670 REM : NORMAL ARRAY
680 DIM N(2,6,3)
690 FOR I=1 TO 2
700       FOR J=1 TO NS(I)
710           FOR K=1 TO 3
720               U(K)= V(I,SF(I,J,2),K) - V(I,SF(I,J,1),K)
730               W(K)= V(I,SF(I,J,3),K) - V(I,SF(I,J,1),K)
740           NEXT K
750           N(I,J,1)= U(2)*W(3) - W(2)*U(3)
760           N(I,J,2)= U(3)*W(1) - W(3)*U(1)
770           N(I,J,3)= U(1)*W(2) - W(1)*U(2)
780       NEXT J
790 NEXT I
800 RETURN
810 REM : VISIBILITY TEST
820 DIM VSF(2,6),
830 XP= D*SN2*CN1 : YP= D*SN2*SN1 : ZP= D*CN2
840 FOR I=1 TO 2
850       FOR J=1 TO NS(I)
860           LX=XP - V(I,SF(I,J,1),1)
870           LY=YP - V(I,SF(I,J,1),2)
880           LZ=ZP - V(I,SF(I,J,1),3)
890           T= N(I,J,1)*LX + N(I,J,2)*LY + N(I,J,3)*LZ
900           IF T <= 0 THEN VSF(I,J)=0 ELSE VSF(I,J)=1
910       NEXT J
920 NEXT I
930 RETURN
940 REM: VISIBLE EDGE ARRAY
950 DIM E(2,12,4), NVE(2)
960 FOR I=1 TO 2
970       M=1
980       FOR J=1 TO NS(I)
990           IF VSF(I,J)=0 THEN  1110
1000          E1= SF(I,J,1)
1010          FOR K=2 TO NPS(I,J)
1020              E2=SF(I,J,K)
1030              FOR L=1 TO M
1040                  IF E(I,L,1)=E2 AND E(I,L,2)=E1 THEN E(I,L,3)=2:GOTO 1080
1050              NEXT L
1060              E(I,M,1)=E1: E(I,M,2)=E2: E(I,M,3)=1: E(I,M,4)=J
1070              M=M+1
1080              E1=E2
1090          NEXT K
1100          NVE(I)=M-1
1110      NEXT J
1120 NEXT I
1130 RETURN
1140 REM: MINIMAX TEST
1150 FOR I=1 TO 2
1160      XMX(I)=-9999999999: XMN(I)=9999999999
```

PROGRAM 5.2 (continued)

```
1170      YMX(I)=XMX(I) : YMN(I)=XMN(I)
1180      FOR J=1 TO NVE(I)
1190          IF E(I,J,3) <>1 THEN 1260
1200          K=E(I,J,1)
1210          A=SC(I,K,1) : B=SC(I,K,2)
1220          IF A > XMX(I) THEN XMX(I)=A
1230          IF A < XMN(I) THEN XMN(I)=A
1240          IF B > YMX(I) THEN YMX(I)=B
1250          IF B < YMN(I) THEN YMN(I)=B
1260      NEXT J
1270  NEXT I
1280      IF (XMX(1)<XMN(2)) OR (XMX(2)<XMN(1)) OR (YMX(1)<YMN(2))
             OR (YMX(2)<YMN(1))      THEN OVERLAP=0 ELSE OVERLAP=1
1290  RETURN
1300  REM : PLOT
1310      FOR KJ=1 TO NVE(LL)
1320      IF E(LL,KJ,3)=0 THEN 1370
1330      F1=E(LL,KJ,1): F2=E(LL,KJ,2)
1340      A=SC(LL,F1,1):B=SC(LL,F1,2): C=SC(LL,F2,1):D=SC(LL,F2,2)
1350      A=SCF*A + TX:B=TY -B: C=SCF*C + TX : D= TY - D
1360      LINE (A,B) - (C,D)
1370      NEXT KJ
1380  RETURN
1390  REM : DETERMINE OBJECT PRIORITY
1400    I=1 : JJ=2
1410      FOR J=1 TO NVE(I)
1420          A=E(I,J,1): B=E(I,J,2)
1430          FOR K=1 TO NVE(JJ)
1440              IF E(JJ,K,3) <> 1 THEN 1520
1450              C=E(JJ,K,1): D=E(JJ,K,2)
1460              GOSUB 1690
1470              IF NCUT=0 THEN 1520
1480              GOSUB 1880
1490              IF DT1 > DT2 THEN 1500 ELSE 1510
1500              KKK=0:II=1:LL=2: GOSUB 1300: RETURN
1510              KKK=0:II=2:LL=1: GOSUB 1300: RETURN
1520          NEXT K
1530      NEXT J
1540      GOSUB 1590
1550          IF (ZMN(1) < ZMX(2)) AND (YMX(2)>YMX(1)) THEN LL=2:
                 GOSUB 1300: END
1560          IF (ZMX(2) < ZMN(1)) AND (YMX(1)>YMX(2)) THEN LL=1:
                 GOSUB 1300: END
1570          IF (YMX(2)>YMX(1)) THEN KKK=1:II=1:LL=2: GOSUB 1300: RETURN
1580          IF (YMX(1)>YMX(2)) THEN KKK=1:II=2:LL=1: GOSUB 1300: RETURN
1590  REM: CONTAINMENT TEST
1600    DIM ZMX(2),ZMN(2)
1610      FOR I=1 TO 2
1620          ZMX(I)=0: ZMN(I)=1
1630          FOR J=1 TO NV(I)
1640              IF SC(I,J,3) > ZMX(I) THEN ZMX(I)=SC(I,J,3)
1650              IF SC(I,J,3) < ZMN(I) THEN ZMN(I)=SC(I,J,3)
1660          NEXT J
1670      NEXT I
1680      RETURN
1690  REM : INTERSECTION TEST
1700      AX=SC(L,A,1): AY=SC(L,A,2): BX=SC(L,B,1):BY=SC(L,B,2)
1710      CX=SC(L,C,1): CY=SC(L,C,2): DX=SC(L,D,1):DY=SC(L,D,2)
1720      R1=AX-BX: R2=AY-BY: R3=CX-DX:  R4=CY-DY
1730      S1=SGN(R1): S2=SGN(R2): S3=SGN(R3): S4=SGN(R4)
1740      A1=AX: B1=BX: A2=AY: B2=BY: C1=CX: D1=DX: C2=CY: D2=DY
1750        IF S1 > 0 THEN A1=BX: B1=AX
1760        IF S2 > 0 THEN A2=BY: B2=AY
1770        IF S3 > 0 THEN C1=DX: D1=CX
1780        IF S4 > 0 THEN C2=DY: D2=CY
1790  REM - MINMAX TEST FOR EDGE OVERLAPPING
1800      IF (B1<C1) OR (D1<A1) OR (B2<C2) OR (D2<A2) THEN NCUT=0: RETURN
1810      M1=R2/R1 : M2=R4/R3 : M3=M2-M1
1820      IF (M3=0) OR (ABS(M3)<=.0001) THEN NCUT=0: RETURN
1830  REM - COMPUTE INTERSECTION POINT
1840      Z0=AY-CY:Z1=M1*AX-M2*CX : Z2=(AY-M1*AX)*M2: Z3=(CY-M2*CX)*M1
1850      XI=(Z0-Z1)/M3   : YI=(Z2-Z3)/M3
1860      IF (XI<A1) OR (XI>B1) OR (XI<C1) OR (XI>D1) OR
          (YI<A2) OR (YI>B2) OR (YI<C2) OR (YI>D2) THEN NCUT=0 ELSE NCUT=1
1870    RETURN
1880  REM - DEPTH TEST
1890  II=I: IJ=J
1900      A1=AX: A2=AY: A3=SC(I,A,3): O=E(I,J,4)
1910      B1=BX: B2=BY: B3=SC(I,B,3)
1920      GOSUB 1980 : DT1 =DEPTH
1930  II=L: IJ=K
1940      A1=CX: A2=CY: A3=SC(L,C,3): O=E(L,K,4)
1950      B1=DX: B2=DY: B3=SC(L,D,3)
1960      GOSUB 1980 : DT2 =DEPTH
1970  RETURN
1980  REM - COMPUTE ZS (PLANE NORMAL)
```

PROGRAM 5.2 (continued)

```
1990 FOR JJ=1 TO NPS(II,0)
2000      IF SF(II,0,JJ) <> E(II,1,2) THEN 2020
2010      KK=SF(II,0,JJ+1): GOTO 2030
2020 NEXT JJ
2030      C1=SC(II,KK,1): C2=SC(II,KK,2): C3=SC(II,KK,3)
2040      AA=B1-A1: BB=B2-A2: CC=B3-A3: DD=C1-A1: EE=C2-A2: FF=C3-A3
2050      XX=BB*FF-CC*EE: YY=CC*DD-AA*FF: ZZ=AA*EE-BB*DD
2060      T1=XX*A1 + YY*A2 + ZZ*A3
2070      T2=XX*X1 + YY*Y1
2080      DEPTH=(T1-T2)/ZZ :RETURN
2090 REM - VISIBLE ENDPOINT
2100    DIM ENDPOINT(NVE(II))
2110    FOR J=1 TO NVE(II)
2120              A=E(II,J,1)
2130              FOR K=1 TO NVE(LL)
2140                        IF E(LL,K,3) <> 1 THEN 2200
2150                        C=E(LL,K,1): D=E(LL,K,2)
2160                        AX=SC(II,A,1):AY=SC(II,A,2):
                            CX=SC(LL,C,1):CY=SC(LL,C,2):
                            DX=SC(LL,D,1):DY=SC(LL,D,2)
2170                        V1=DX-CX : V2=DY-CY
2180                        U1=AX-CX : U2=AY-CY :TEST=U2*V1-U1*V2
2190                        IF TEST <=0 THEN ENDPOINT(J)=1:GOTO 2220
2200              NEXT K
2210              ENDPOINT(J)=-1
2220    NEXT J
2230    RETURN
2240 REM : PLOT OF LINE SEGMENT
2250   DIM IX(2),IY(2)
2260    I=II: L=LL
2270    FOR J=1 TO NVE(II).
2280              V1=ENDPOINT(J): NCUT=0 :TCUT=0
2290              A=E(II,J,1): B=E(II,J,2)
2300              FOR K=1 TO NVE(LL)
2310                        IF KKK=1 THEN 2330
2320                        IF E(LL,K,3) <> 1 THEN 2390 ELSE 2340
2330                        IF E(LL,K,3) <> 2 THEN 2390 ELSE 2340
2340                        C=E(LL,K,1): D=E(LL,K,2)
2350                        GOSUB 1690
2360                        IF NCUT=0  THEN 2390 ELSE 2370
2370                        TCUT=TCUT+NCUT
2380                        IX(TCUT)=SCF*XI+TX: IY(TCUT)=TY-YI
2390              NEXT K
2400                        AX=SCF*AX + TX : AY=TY - AY :
                            BX=SCF*BX + TX : BY=TY - BY
2410                        GOSUB 2440
2420    NEXT J
2430    RETURN
2440 REM: PLOT OF VISIBLE EDGE SEGMENT
2450    IF TCUT=0 AND V1=1 THEN LINE (AX,AY)-(BX,BY): RETURN
2460    IF TCUT=0 AND V1=-1 THEN RETURN
2470    IF TCUT=1 AND V1=1 THEN LINE (AX,AY)-(IX(1),IY(1)): RETURN
2480    IF TCUT=1 AND V1=-1 THEN LINE(IX(1),IY(1))-(BX,BY): RETURN
2490    IF   (AX<BX) AND (IX(1) < IX(2)) THEN 2510
2500    IF   (AX>BX) AND (IX(2) < IX(1)) THEN 2510 ELSE  2520
2510    LINE (AX,AY)-(IX(1),IY(1)):LINE (IX(2),IY(2))-(BX,BY): RETURN
2520    LINE (AX,AY)-(IX(2),IY(2)):LINE (IX(1),IY(1))-(BX,BY): RETURN
```

PROGRAM 5.2 Hidden Line Elimination—Two Objects

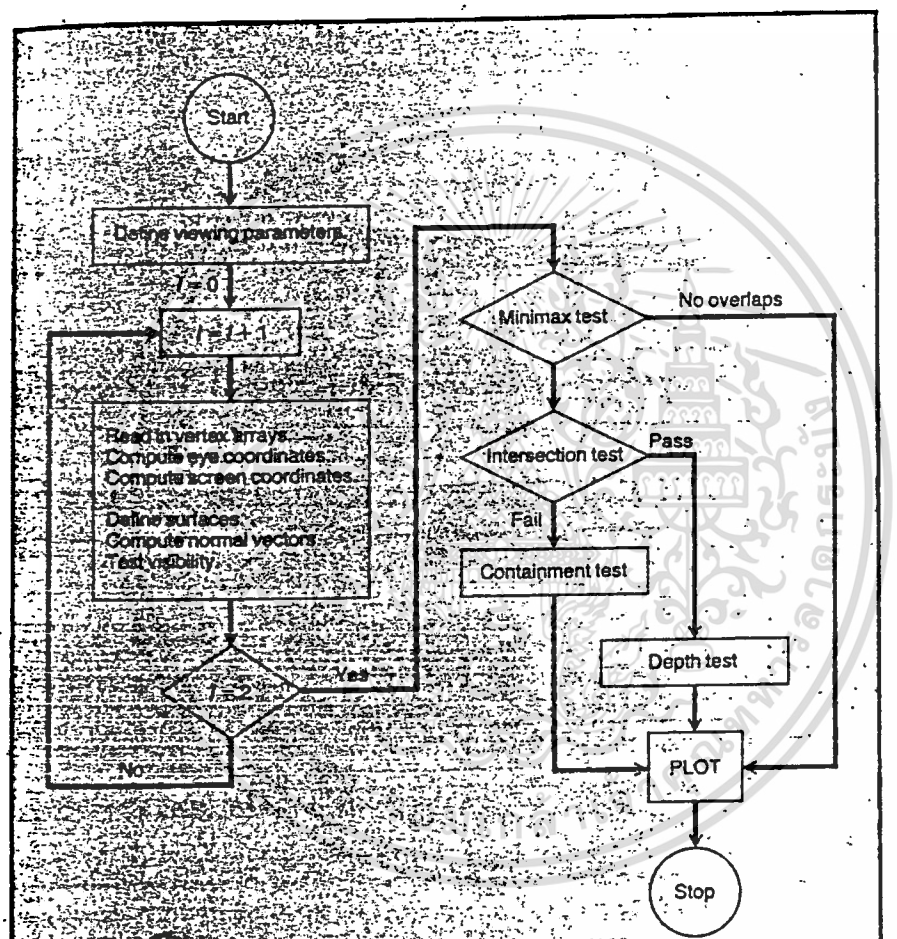


FIGURE 5.13 Programming logic chart

Ze ที่ใหญ่และเล็กที่สุด ในตัวอย่างค่า Ze อยู่ในช่วง 7.19 และ 9.62 เพราะฉะนั้นเราจะตั้งค่า K1 = 7 และ K2 = 10 เพื่อให้แน่ใจว่าจะครอบคลุมเนื้อที่ของภาพทั้งหมด (ดูในส่วน 4.4.2) ในการคำนวณหา eye coordinate ใช้สมการ (4.28) คำนวณ screen coordinate ใช้สมการ (4.31), (4.32), (4.38) สำหรับทุกจุด coordinate เราได้คำนวณ coordinate เหล่านี้แล้วส่วนค่าที่ได้อยู่ในตาราง 5.6 ค่า screen coordinate ในตารางนี้ไม่ใช่ device coordinate ถ้าจะสร้าง device coordinate ให้ใช้ viewing Transformation ในสมการ 3.42

5.3.4 The Surface Array

มี 10 พื้นผิวในภาพนี้ สี่พื้นผิวเป็นของปิรามิด และ 6 พื้นผิวเป็นของลูกบาศก์ Surface Array SF(I,J,K) ทำหน้าที่เหมือนกับในโปรแกรม 5.1 โดยอยู่ในส่วน 5.1 I จะเป็นตัวกำหนดวัตถุ (I =1,2) และ K เป็นตัวกำหนดจำจุดที่ใช้ในการต่อเข้ากับแต่ละพื้นผิว J แต่ละพื้นผิวในปิรามิดจะกำหนดด้วยจุด 3 จุด ในขณะที่แต่ละพื้นผิวในลูกบาศก์จะถูกกำหนดด้วยจุด 4 จุด เราจะกำหนดส่วนภายใน Array SF(I,J,1) โดยผ่านทาง SF(1,J,NPS(I,J) จุดของพื้นผิว J ของวัตถุ I เมื่อ NPS (J,J) คือจำนวนของจุดที่กำหนดพื้นผิว J ของวัตถุ I เป็นสิ่งสำคัญมากที่จะแสดงจุดต่าง ๆ ในทิศทางทวนเข็มนาฬิกา เรียงตามลำดับรอบพื้นผิวตามที่มองจากภายนอกวัตถุ เมื่อทำเสร็จแล้วค่าของ surface array เป็นดังตารางที่ 5.7

5.3.5 The Normal Array

เราคำนวณ normal array N(I,J,K) เหมือนแต่ก่อนโดยที่ N(I,J,1), N(I,J,2) และ N(I,J,3) จะกำหนดส่วนประกอบ 3 ส่วนที่เป็น normal vector พุ่งออกจากพื้นผิว J ของวัตถุ I ค่าต่าง ๆ อยู่ในตาราง 5.8

5.3.6 Visibility test

เราจะใช้ visibility test กับพื้นผิวแต่ละพื้นผิวของวัตถุสำหรับแต่ละพื้นผิวเราจะล็อก lind of sigth vector l โดยการต่อจุดเริ่มต้นของ

TABLE 5.6 Coordinate transformations

Pyramid vertex number	x,	y,	z	x_e,	y_e,	z_e	x_s,	y_s,	z_s
(1)	1,	1,	0	−0.3 ,	−0.52,	8.71	−16.87,	−29.73,	0.656
(2)	0,	1,	0	0.54,	−0.32,	9.22	29.52,	−17.04,	0.803
(3)	0,	0,	1	0 ,	0.92,	9.62	0 ,	48.16,	0.909
(4)	1,	0,	0	−0.84,	−0.21,	9.49	−44.17,	−10.75,	0.875
Cube vertex number									
(1)	1,	2.5,	0	0.52,	−0.99,	7.55	34.62,	−65.52,	0.243
(2)	1,	2.5,	1	0.52,	−0.07,	7.19	36.43,	− 4.34,	0.081
(3)	1,	1.5,	1	−0.03,	0.25,	7.95	− 1.37,	15.83,	0.394
(4)	1,	1.5,	0	−0.03,	−0.68,	8.32	− 1.31,	−40.54,	0.531
(5)	0,	1.5,	0	0.81,	−0.48,	8.83	46.24,	−26.68,	0.691
(6)	0,	2.5,	0	1.36,	−0.79,	8.05	84.50,	−48.75,	0.436
(7)	0,	2.5,	1	1.36,	0.14,	7.68	88.62,	9.22,	0.295
(8)	0,	1.5,	1	0.81,	0.45,	8.45	48.28,	26.94,	0.574

TABLE 5.7 Surface arrays for two objects

Object	J	K	Object	J	K
Pyramid	1	1 2 3 1	Cube	1	1 2 3 4 1
	2	2 4 3 2		2	6 7 2 1 6
	3	4 1 3 4		3	5 8 7 6 5
	4	1 4 2 1		4	4 3 8 5 4
				5	4 5 6 1 4
				6	7 8 3 2 7

TABLE 5.8 Normal array for two objects

Object	J	K	Object	J	K
Pyramid	1	0 1 1	Cube	1	1 0 0
	2	−1 −1 −1		2	0 1 0
	3	1 0 1		3	−1 0 0
	4	0 0 −1		4	0 −1 0
				5	0 0 −1
				6	0 0 1

Surface normal vector กับจุดสังเกตุต่อมาก็คำนวณหาค่า dot product และกำหนดเครื่องหมายของ dot product เราจะเก็บค่าสภาวะดังกล่าวของแต่ละพื้นผิว J ของวัตถุ I ไว้ VSF(I,J) (ถ้ามองเห็น VSF (I,J) =1 ถ้าไม่เห็น VSF (I,J)=0 ค่า visibility test ของปิรามิดในพื้นผิว 1 และ 3 อยู่ในสภาวะมองเห็นค่า visibility test ของลูกบาศก์ ในพื้นผิว 1,2,6 อยู่ในสภาวะมองเห็น แสดงค่าทั้งหมดนี้ในตาราง 5.10

5.3.7 visible edge array

เราจะแนะนำ E (I,J,K) แทน array ของด้าน ส่วนประกอบตัวแรก I จะเป็นตัวกำหนดชนิดวัตถุ สำหรับแต่ละค่า I ส่วนประกอบตัวที่ 2 , J จะเป็นตัวกำหนดจำนวนด้านในวัตถุ สำหรับแต่ละค่า J ส่วนประกอบ K จะเก็บรายละเอียดของข้อมูล 4 แบบ: จุดปลายของด้าน (K=1,2) ชนิดของด้าน (K= 3) และด้านนั้นเป็นของพื้นผิวใด (K = 4)

ด้านที่มองเห็นจะกำหนดโดยส่วนของโปรแกรมเหมือนกับในโปรแกรม 5.1 ค่าต่างของ edge array อยู่ในตาราง 5.11

5.3.8 Minimax test

สำหรับแต่ละวัตถุเราจะหาค่าสูงสุด ต่ำสุด ของ Xและ Y coordinate จากการตรวจเช็ค screen coordinate array

วัตถุที่ 1 pyramid		วัตถุที่ 2 ลูกบาศก์	
max X=29.52	max Y=48.16	maxX =88.62	maxY = 26.94
XMX(1)	YMX(1)	XMX(2)	YMX(2)
minX=44.17	minY = -29.73	minX = -1.37	minY = -65.52
XMN(1)	YMN(1)	XMN(2)	YMN(2)

จากกฎminimax ในสมการ 5.5 แสดงว่าวัตถุ 2 ชิ้นอาจจะทับกัน

5.3.9 Plot Routine

ถ้า minimax test ตรวจสอบแล้วว่าไม่มีการทับกันระหว่างวัตถุ 2 ชิ้น เราก็จะ plot วัตถุทั้ง 2 plot subroutineจะ plotด้านที่มองเห็น

ของวัตถุโดยการอ่านค่าจุดปลายทั้ง 2 ของด้านนั้น โดยตลอดทั้งโปรแกรมจะร้องขอ plot subroutine นี้หลายครั้ง (หมายเหตุ view transform จาก screen coordinate ไปเป็น device coordinate อยู่ในบรรทัด 1350)

5.3.10 object priority

ด้านของวัตถุอาจถูกบังเป็นบางส่วนโดยด้านที่อยู่ใกล้จุดสังเกตุ เพราะฉะนั้นเราจึงต้องมีการกำหนดความสัมพันธ์ระหว่างตำแหน่งของวัตถุและในตัวอย่างนี้ ปิรามิด กับลูกบาศก์ไม่ลอดทะลุซึ่งกันและกัน ฉะนั้นการแก้ปัญหาของเราก็คือการกำหนดว่าวัตถุชิ้นไหนวางอยู่ใกล้จุดสังเกตุ วัตถุที่อยู่ใกล้จุดสังเกตุมากกว่าจะถูกกำหนดให้เป็นวัตถุที่มีความสำคัญลำดับหนึ่ง

Testing Sequence

ในตอนแรกเราตรวจสอบเห็นว่าถ้าด้านหนึ่งของปิรามิดตัดกับด้านใด ๆ ของลูกบาศก์ (บรรทัด 1460) เมื่อมีการตัดกัน เราจะใช้ depth test ตรวจสอบเพื่อกำหนดว่าวัตถุอันไหนอยู่ใกล้ผู้สังเกตุมากกว่าก็จะมองเห็น (บรรทัด 1480–1510)

ด้านที่มองเห็นของวัตถุมีความสำคัญอันดับหนึ่งจะถูกวาดขึ้นก่อนในบรรทัด 1500–1510) เพราะว่าด้านที่สัมพันธ์กับวัตถุที่มีความสำคัญอันดับหนึ่ง จะไม่ถูกบังโดยวัตถุที่มีความสำคัญอันดับสอง

เมื่อไม่มีการตัดกันระหว่างวัตถุทั้งสองเราก็จะทำ containment test (บรรทัด 1540) containment test จะแสดงให้เห็นว่าวัตถุหนึ่งล้อมรอบอีกวัตถุหนึ่ง (บรรทัด 1550–1560) หรือทั้งสองไม่ทับกันเลย (บรรทัด 1570–1580) เราก็ยังมีการกำหนดลำดับความสำคัญด้วย เมื่อมีวัตถุถูกล้อมรอบ เราทำแบบนี้โดยการให้วัตถุเรียงตามลำดับความสำคัญโดยขึ้นอยู่กับค่าที่น้อยที่สุดของ Zs สำหรับจุดใด ๆ ของวัตถุ

เมื่อวาดด้านของวัตถุที่มีความสำคัญอันดับหนึ่งแล้ว ถ้าวัตถุที่มีความสำคัญอันดับสองถูกล้อมรอบโดยวัตถุที่มีความสำคัญอันดับหนึ่ง วัตถุที่สองก็จะไปถูกวาด ถ้าสภาวะไม่เป็นเช่นนี้ ด้านที่อยู่ภายนอกของวัตถุที่มีความสำคัญอันดับสองจะถูกวางเป็นลำดับต่อมา และด้านที่อยู่ภายในก็จะถูกทำการตรวจเช็ค ถ้าผลการตรวจเช็คปรากฏว่าด้านดังกล่าวนี้อยู่ภายในของด้านภายนอกของวัตถุที่มีความสำคัญอันดับหนึ่ง

ส่วนที่มองเห็นบางส่วนของด้านภายในเท่านั้นจะถูกวาด (กรณีนี้จะแสดงให้เห็นในส่วน 5.3.13)

ต่อไปเราจะแสดงให้เห็นส่วนของ computer program ที่ทำหน้าที่ prority test ต่อจากนั้นเราจะแสดง subroutine ซ้อนพร้อมกับตัวอย่าง numerical โดยเฉพาะ ซึ่งจะทำให้เรารู้ว่าด้านไหนที่มีการตัดกันและค่าความลึกจะทำได้อย่างไร

Intersection test

แต่ละด้านของภาพ (ในที่นี้คือปิรามิด) จะถูกเปรียบเทียบกันแต่ละด้านของอีกภาพหนึ่ง (ลูกบาศก์) เพื่อตรวจสอบการตัดกันของด้านเหล่านี้ (บรรทัด 1420 และ 1450) ทันทีที่เรากำหนดคู่ของด้านที่ถูกเปรียบเทียบ โดยจะนำด้านที่กำหนดโดยรูปแบบ screen coordinate(บรรทัด 1700–1710) minimax test ของด้านแต่ละด้านจะช่วยเพิ่มความเร็วในการ process โดยการกำหนดอย่างรวดเร็วว่าด้านทั้งสองไม่มีโอกาสที่จะตัดกันได้ (บรรทัด 1720–1790) ในตัวอย่างของเราเราจะสังเกตุว่ามีการตัดกัน 2 ครั้ง

	ปิรามิด	ลูกบาศก์	Xi Yi
การตัดกันครั้งที่ 1	ด้านที่ 1	ด้านที่ 3	-1.320,-25.47
การตัดกันครั้งที่ 2	ด้านที่ 2	ด้านที่ 9	13.168,19.087

เราจะใช้จุดตัดเพียงจุดเดียวในการกำหนดความสำคัญของวัตถุ ในทันทีที่เราได้จุดของด้านที่มีการตัดกัน เราจะ set ค่า Ncut = 1 และกลับไปที่บรรทัด 1460 (เราจะใช้จุดตัดที่ 1 ใน depth test)

Depth test

สำหรับปิรามิด ด้านที่ 1 เป็นของพื้นผิวที่ 1 ของวัตถุ (ดูรูป 5.14) เพราะว่าจุดที่ 1,2,3 เป็นตัวกำหนดพื้นผิว เราจะใช้ screen coordinate ของ 3 จุดนี้ กำหนดสมการระนาบ (บรรทัด 1900–1910)

TABLE 5.9 Pyramid visibility status

Pyramid surface	n			ℓ			$n \cdot \ell$	Visibility	*VSF(I)*
1	0	1	1	4.05	6.78	3.75	10.53	Visible	1
2	−1	−1	−1	5.05	6.78	3.75	−15.57	Invisible	0
3	1	0	1	4.05	7.78	3.75	7.80	Visible	1
4	0	0	−1	4.05	6.78	3.75	− 3.75	Invisible	0

TABLE 5.10 Cube visibility status

Cube surface	n			ℓ			$n \cdot \ell$	Visibility	*VSF(I)*
1	1	0	0	4.05,	5.28	3.75	4.05	Visibile	1
2	0	1	0	5.05,	5.28	3.75	5.28	Visible	1
3	−1	0	0	5.05	6.28	3.75	−5.05	Invisible	0
4	0	−1	0	4.05	6.28	3.75	−6.28	Invisible	0
5	0	0	−1	4.05	6.28	3.75	−3.75	Invisible	0
6	0	0	1	5.05	5.28	2.75	2.75	Visible	1

TABLE 5.11 Visible edge array for two objects

Object *I*	Edge number (*J*)	Endpoints *K*=1	*K*=2	Edge type *K*=3	Surface number *K*=4
Pyramid	1	1	2	1	1
	2	2	3	1	1
	3	3	1	2	1
	4	4	1	1	3
	5	3	4	1	3
Cube	1	1	2	2	1
	2	2	3	2	1
	3	3	4	1	1
	4	4	1	1	1
	5	6	7	1	2
	6	7	2	2	2
	7	1	6	1	2
	8	7	8	1	6
	9	8	3	1	6

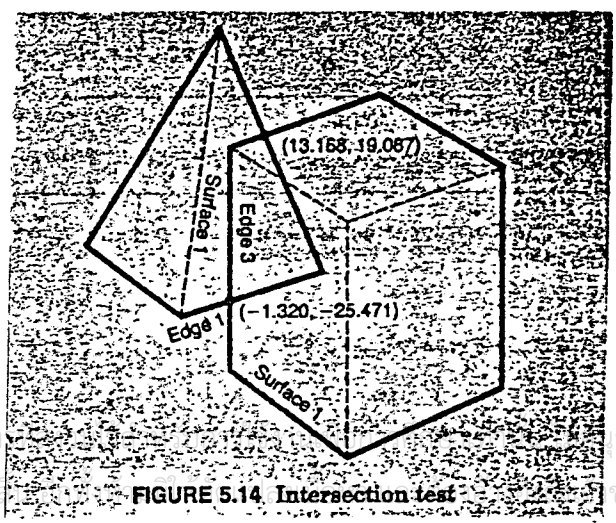


FIGURE 5.14 Intersection test

ตำแหน่งจุด	Xs	Ys	Zs
1	-16.87	-29.73	0.656
2	29.52	-17.04	0.803
3	0	48.16	0.909

ขณะนี้ใช้สมการ (5.13) คำนวณหาค่าสัมประสิทธิของสมการระนาบที่จะแทนสมการ Perspective image ของพื้นผิวที่ 1

$$-8.211Xs-9.242Ys+3399.316Zs-2654.498=0$$

หาค่า Zs ที่ (Xi Yi) จะทำให้ค่าความลึก (บรรทัด 2080)

Xs = Xi = -1.320

Ys = Yi = -25.471

Zs = 0.7058 = DT1

สำหรับตัวอย่าง ด้าน 3 เป็นของพื้นผิว 1 ของลูกบาศก์ ในพื้นผิวนี้จะใช้จุดกำหนด 4 จุดคือ 1,2,3,4 เราจะใช้จุด 3,4, และ 1 (หรือ 2,3,4) หาค่าสมการระนาบ (บรรทัด 1930-1950)

ตำแหน่งจุด	Xs	Ys	Zs
3	-1.37	15.73	0.399
4	-1.31	-40.54	0.531
1	34.62	-65.52	0.243

สมการระนาบก็จะหาได้เหมือนกับตัวอย่างก่อน ในที่นี้มีค่า

$$19.56Xs+4.758Ys+2023.997Zs-857.85=0$$

หาค่า Zs ที่ Xi Yi

Zs = 0.4965 = DT2

เป็นที่แน่ชัดแล้ว ลูกบาศก์นี้จะอยู่ใกล้ผู้สังเกตุมากกว่าปิรามิด เพราะว่าค่าความลึก Zs= 0.4965 มีค่าน้อยกว่าค่าความลึก Zs ของปิรามิด , Zs = 0.7058 เพราะฉะนั้นลูกบาศก์ก็จึงมีความสำคัญอันดับ 1 (บรรทัด 1490) ทันทีที่ได้ค่าลำดับความสำคัญของวัตถุแล้ว เราก็กลับจากโปรแกรมย่อยเพื่อไปสู่โปรแกรมหลัก (บรรทัด 130)

5.3.11 Visibility status of edge enpoint

มีลักษณะด้าน 3 อย่างมีสัมพันธ์กับพื้นผิวของวัตถุที่มีความสำคัญเป็นลำดับสอง ด้านบางด้านจะถูกบังหมดโดยวัตถุที่อยู่ใกล้กว่า, ด้านบางด้านถูกบังเป็นบางส่วน, และด้านบางด้านเห็นได้ตลอด, โปรแกรมบรรทุด 2120 จะเป็นตัวกำหนดจุดปลายจุดแรกของแต่ละด้านของวัตถุที่มีความสำคัญลำดับสอง โปรแกรมบรรทัดที่ 2130-2200 อ้างอิงถึงจุดปลายแต่ละจุดเพื่อแบ่งเป็นพวกคือ พวกที่มองเห็น และพวกที่ถูกบัง เราจะใช้รูป 5.15 แสดงการแบ่งพวกของจุดปลายทำได้อย่างไร ในรูปนี้ด้านที่อยู่ภายนอกของลูกบาศก์จะถูก plot และจุดปลายของด้านแต่ละด้านของปิรามิดก็อยู่ในรูปนี้ด้วย หมายเหตุ จุดปลาย (B) ซึ่งถูกบังด้วยลูกบาศก์ อยู่ภายในขอบเขตด้านนอกของลูกบาศก์ ในขณะที่จุด A อยู่ภายนอก ดังนั้นเราจะกำหนด surface array และเก็บค่ารายละเอียดของ edge array ตามทิศทางทวนเข็มนาฬิกา เพราะฉะนั้นจุด O เป็นจุด inside เพราะวางอยู่ในleft half plane ของแต่ละด้านของขอบเขตสิ้นสุด (ดูส่วน 5.2.4) ในทำนองเดียวกันจุด A เป็นจุด outside เพราะมันอยู่ทาง right half plane ของแต่ละด้านของขอบเขตสิ้นสุดอย่างน้อย 1 ด้าน

เพราะว่าจุดใด ๆ ที่อยู่ในสภาวะมองเห็น ถ้ามันอยู่ทาง right half plane อย่างน้อยที่สุด 1 ด้านของด้านสิ้นสุดนั้น ๆ ลูป 2130-2200 ทำการเปรียบเทียบจุดปลาย A (บรรทัด 2120) กับด้านที่มองเห็นแต่ละด้านของลูกบาศก์

สำหรับด้านนอก CD ของลูกบาศก์ (บรรทัด 2150) ต้องหาค่า 2 vector (ดูในรูป 5.15) โดย vector V ได้มาจากการต่อจุด C กับจุด D และ vector U ได้จากการต่อจุด C กับจุด A ที่ถูกทดสอบในตัวอย่าง

U = D-C=(-1.31, -40.54)-(-1.37,15.83) = (0.06-56.37)

V = A-C=(-96.87,-29.73)-(-1.37,15.83)=(-15.5,-45.56)

จากสมการ (5.10) ค่า cross product มีค่าดังนี้

UxV = (0,0,-876.59) = -876.59 (0,0,1)

เพราะว่า UxV ได้ค่าสัมประสิทธิ์ตัวคูณ (0,0,1) เป็นลบ จุด A จึงอยู่ทางระนาบขวาของ CD (อยู่ภายนอกลูกบาศก์ ในกรณีนี้ program บรรทัด 2190 จะ

กำหนดค่า 1 ลงใน End point (J) และก็ทำการตรวจสอบจุดที่เหลือต่อไป เพื่อทำการแบ่งพวก..

ถ้าจุดปลาย j ที่ทดสอบแล้วอยู่ทาง left half plane ของด้าน นอกแต่ละด้านของลูกบาศก์เราจะกำหนดค่า -1 ลงใน endpoint (J) ถ้าจุดทั้งหมดได้ถูกกำหนดพวกหมดแล้ว ต่อมาเราก็กลับจาก program ย่อย

5.3.12 Plot of visible edge of second priority object

ทันทีที่เราได้ค่าของสภาวะ visibility ของจุดปลายแต่ละจุดแล้วในวัตถุที่มีความสำคัญลำดับสองแล้ว เราก็จะทำการ plot ด้านที่มองเห็น ด้านของวัตถุที่มีความสำคัญเป็นลำดับสองสามารถถูกแบ่งได้เป็นหลาย ๆ พวก

– ถ้าจุดปลายเริ่มต้นของด้าน J มองเห็นและไม่มีด้านใดของวัตถุที่มีความสำคัญลำดับหนึ่งตัดด้าน J นี้แล้ว เราจะ plot ส่วนของด้านนี้ตลอดทั้งเส้น

– ถ้า Endpoint J = -1 และไม่มีด้านใดของวัตถุที่มีความสำคัญลำดับหนึ่งตัดกับด้าน J, ด้าน J นี้ก็จะมองไม่เห็นและเราจะกลับจากโปรแกรมย่อย

– ถ้ามีจุดต่อเนื่องจุดหนึ่งเท่านั้น ด้าน J ก็จะถูกบังเป็นบางส่วนด้วยวัตถุที่มีความสำคัญลำดับหนึ่ง

– ถ้ามีการตัดกัน 2 จุด ด้าน J จะมีส่วน 2 ส่วนที่มองเห็นแยกจากกัน program บรรทัด (2470–2500) จะเป็นตัวกำหนดส่วนของมองเห็นและ plot ส่วนเหล่านี้

รูปหลายเหลี่ยมนูน (รูปหลายเหลี่ยมที่มีมุมภายในทุกมุมน้อยกว่า 180 °) ไม่สามารถมีจุดตัดที่มากกว่า 2 จุดด้วยเส้นตรงเพียงเส้นเดียว ด้วยเหตุนี้กลุ่มของกฎตรวจสอบเหล่านี้ก็เพียงพอที่จะใช้ในตัวอย่างของเรา รูปหลายเหลี่ยมเว้าสามารถมีจุดตัดมากกว่า 2 จุดด้วยเส้นตรงเพียงเส้นเดียว ด้วยเหตุนี้จึงต้องดัดแปลงส่วนที่ทำการ plot ตามลักษณะต่างเหล่านี้

– หาจุดตัดทั้งหมด

– เรียงค่าจุดตัดโดยเริ่มจากลำดับ coordinate X ต่ำไปลำดับ

coordinate X สูง

– ถ้า Endpoint (J) = 1 plot ส่วนของด้านจากจุดเริ่มต้นไปยังจุดที่มีการตัดกันครั้งแรก จากจุดตัดที่ 2 ไปยังจุดตัดที่ 3 และต่อไป

– ถ้า Endpoint (J) =1 plot ส่วนของด้านจากจุดตัดแรกไปยังจุดตัดที่ 2 จุดที่ 3 ไปยังจุดที่ 4 และต่อไป

– ถ้ามีการตัดกันเป็นจำนวนคู่, จุดสุดท้ายของด้านนี้มองเห็น ถ้าเป็นเช่นนี้ plot ส่วนของ segment จุดตัดสุดท้ายและจุดสุดท้ายของด้านนั้น

5.3.13 Graphic Output

เมื่อ Run program โดยตลอด 10–2500) , computer จะทำการ plot รูปภาพดังแสดงในรูป 5.16 เมื่อเปลี่ยนค่า Viewing perameter เช่น 0 และ ∅ ภาพที่ได้ก็มีรูปแบบแตกต่างไป สำหรับตัวอย่าง การเปลี่ยน view angle (บรรทัด 780) และvertex coordinate (บรรทัด 370–390) ภาพที่ได้จะเปลี่ยนไปดังแสดงในรูป 5.17

5.4 The Masking tachnique

ในส่วนที่ 4.6.2 ได้กล่าวถึงรายละเอียดของโปรแกรมในการสร้างภาพจาก funtion Z= f(X,Y) โดยที่ปรากฏนี้สร้างขึ้นมาอย่างรวบรัด และจะวาดพื้นผิวที่เกิดในทันทีแต่โปรแกรมดังกล่าวมีข้อเสียคือไม่สามารถกำจัดเส้นที่ถูกบังได้ และในส่วนนี้เราจะแสดงวิธีการเพิ่มให้ได้พื้นผิวที่ได้มีสภาพเป็นจริงโดยเพิ่มส่วน

5.4.1 Theory of opertion

เราจะใช้ Masking tachnique ที่คิดขึ้นโดย watkin เทคนิคดังกล่าวขึ้นอยู่กับพื้นฐานที่สำคัญ 2 อย่าง

1. เส้นตรงที่เป็น foreground (Positive Z Direction) จะนำมา plot ก่อนเส้นตรงที่เป็น background

2. เส้นตรงหรือส่วนบางส่วนของเส้นตรงอยู่ในสภาพที่ถูกบังถ้าส่วนดังกล่าวนี้วางอยู่ภายในขอบเขตของเส้นที่ plot ไว้ก่อนหน้านี้แล้ว

Masking technique ทำได้โดยการเก็บค่า Visible

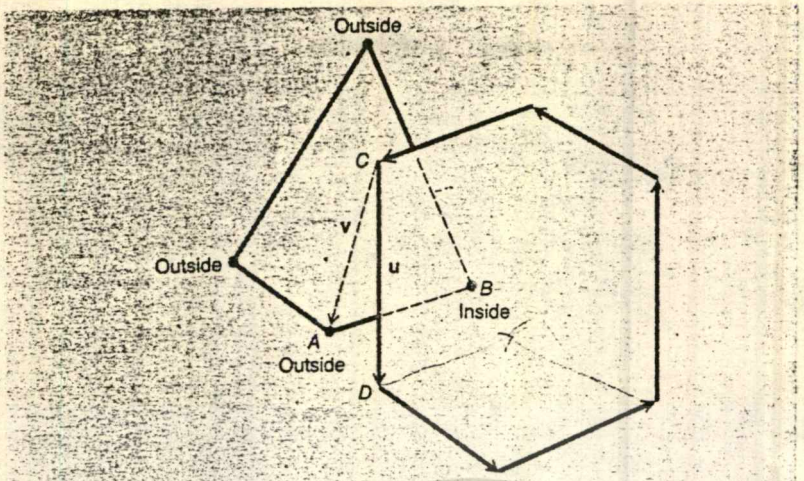


FIGURE 5.15 Endpoint classification for visibility. Outside edges are drawn for the cube.

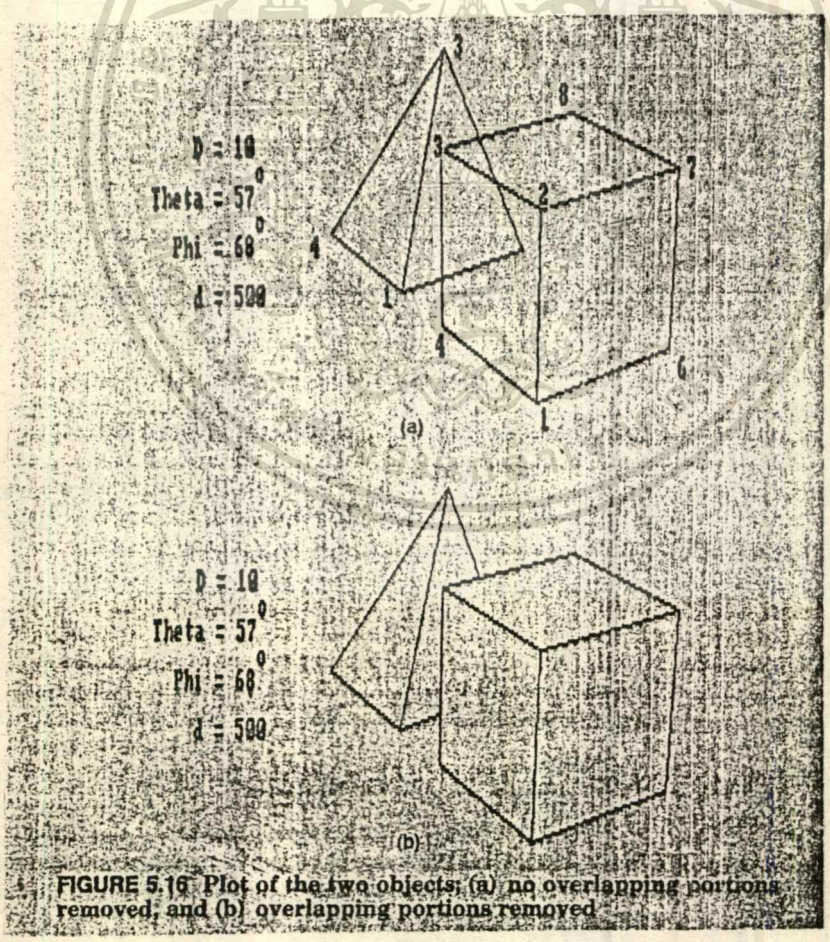


FIGURE 5.16 Plot of the two objects: (a) no overlapping portions removed, and (b) overlapping portions removed

Maximum funtion และ visible minimum funtion ส่วนต่าง ๆ ของเส้นใดถ้ามีค่าตัดอยู่ภายในค่าระหว่าง visible Maximum funtion และ visible minimum funtion ก็จะให้เป็นส่วนที่ถูกบัง ดังรูป 5.18 ในส่วนต่อไปเราจะแสดงวิธีการ Masking technique เปลี่ยนไปเป็น Hidden line elimination routine

Program 5.3 เป็นโปรแกรมที่ขยายเพิ่มเติมมาจาก program 4.2 โดยในโปรแกรมนี้จะมีส่วนของ hidden line routine จุดเริ่มต้นในโปรแกรมก็เหมือนกับโปรแกรม 4.2, โดยเริ่มต้นนั้นจะสร้าง viewing และ screen parameter และ funtion ที่ทำการ plot และในโปรแกรมนี้ยังมีส่วนเพิ่มเติมมาอีก 2 ส่วนคือ Array Ymax(I)และ Ymin (I)Array ทั้ง 2 จะนำมาใช้เหมือนกันคือ นำมาใช้ในแต่ละ hirizontal scan coordinate ทำการเก็บค่าสูงสุดของ vertical screen coordinate ที่พบไว้ใน Ymax(I) และ 2 เก็บค่าต่ำสุดของ Vertical Screen Coordinate ไว้ในY min(I) โดยในสภาวะเริ่มต้นค่าแต่ละค่าของ Ymin(I)จะมีค่าเป็น 0 และค่าแต่ละค่าของ Y min(I) มีค่า =200 (ค่าสูงสุดของ screen) และเมื่อค่า (Xs,Ys) coordinate ถูก plot ค่าของ Ymax (Xs) จะถูก Reset ในแต่ละค่าของ (Xs,Ys) จะถูกพิจารณาในการ plot คือจะต้องนำค่า (Xs,Ys) มาเปรียบเทียบกับ Ymax(Xs) และ Ymin(Xs) ถ้าค่า Ys วางอยู่ภายในระหว่าง Ymax(Xs) และ Ymin(Xs) coordinate (Xs,Ys) จะถูกกำหนดเป็นส่วนที่ถูกบัง ถ้านอกเหนือจากช่วงดังกล่าวก็จะนำ (Xs,Ys)มา plot เมื่อมีการ execute program 5.3 ผลที่ได้เป็นดังรูป 5.19 และเมื่อเปรียบเทียบ 5.19 กับรูป 4.17 รูป 5.19 นี้จะดูว่าเหมือนจริงมากกว่าเพราะได้มีการนำส่วนของเส้นที่ถูกบังออกไป

5.5 The Image space Algorithm

Image space Algorithm มีหลายอย่างที่ได้สร้างแล้วแต่จะกล่าวเพียง 2 อย่างในทั้งหมดที่มีอยู่ และ 2 อย่างที่จะกล่าวนี้คือ depth buffer และ scan line Algorithm Algorithm นี้ถูกออกแบบเพื่อใช้กับ Raster display และ Alporithm นี้จะแก้ไขปัญหาเกี่ยวกับ Hidden Surface

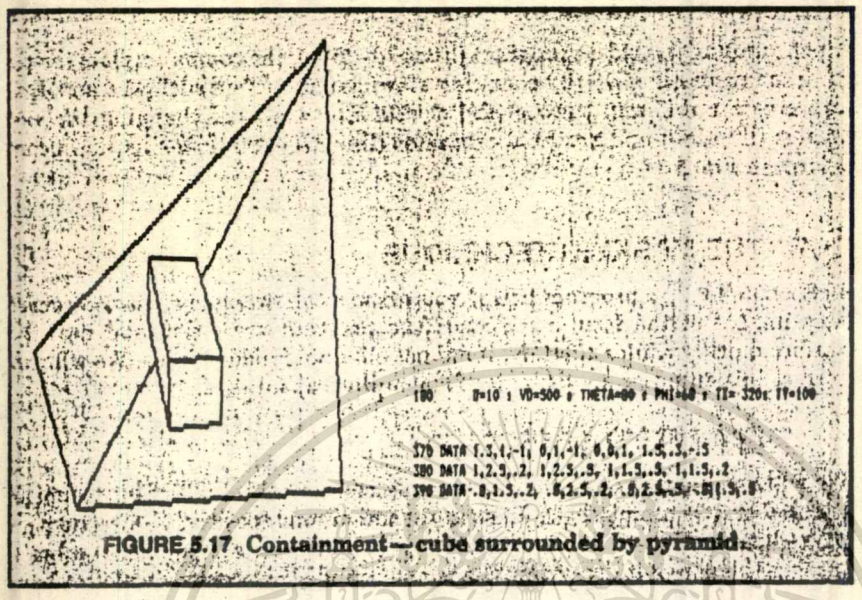


FIGURE 5.17 Containment—cube surrounded by pyramid.

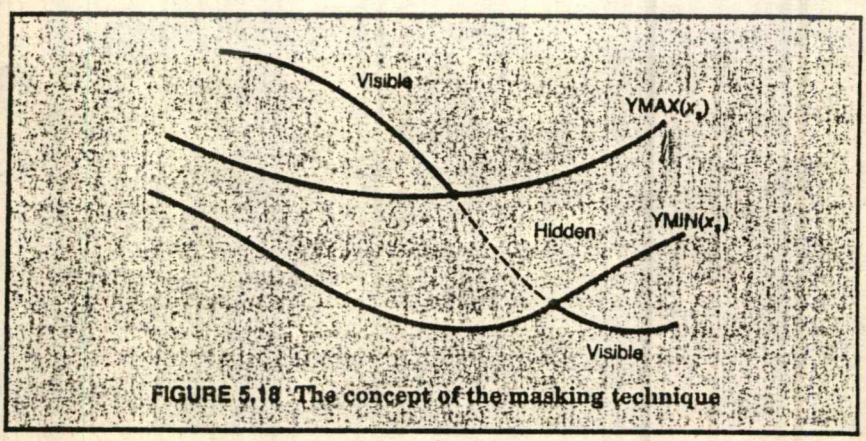


FIGURE 5.18 The concept of the masking technique

```
10 ' Program 5.3: Plot of 3-D Functions with Hidden Lines Removed
20 ' (D,theta,phi) = Viewpoint
30 ' VD = Distance of the Projection Plane from the XY Plane, "d".
40 '  S = Centered screen size. (see Eq.(4.31).
50 ' YMAX(I)= Maximum vertical screen coordinate encountered.
60 ' YMIN(I)= Minimum vertical screen coordinate encountered.
70 ' Flag = The current point is the first of a wire-frame curve (Flag=0).
80 ' Flag1 = The current point is offscreen (Flag1=0).
90 ' Flag2 = The current point is the last referenced point (Flag2=0).
100 '
110 CLS: KEY OFF
120 SCREEN 2
130 DIM YMAX(640), YMIN(640)
140     GOSUB 180
150     GOSUB 240
160     GOSUB 270
170 END
180 '------- Define Viewing Parameters ---------
190       TX= 120: TY = 100: PI = 3.141593 : SCF= 2.4
200       D= 30: VD=420: THETA=30 : PHI= 80 : S=2
210       THETA=THETA*PI/180 : PHI=PHI*PI/180
220       SN1=SIN(THETA) : SN2=SIN(PHI) : CN1=COS(THETA): CN2=COS(PHI)
230 RETURN
240 ' ------- Initialize Ymax and Ymin arrays ------
250 FOR I=1 TO 640 : YMAX(I)=0 : YMIN(I)=199: NEXT I
260 RETURN
270 '------- Define the 3-D function to plot ------
280 DEF FNZ(X,Y) = 7*EXP(-.1*(X*X + Y*Y))
290 FOR X=8 TO -8 STEP -1
300       FLAG =0
310       FOR Y=-8 TO 8 STEP .5
320           Z=FNZ(X,Y)
330           GOSUB 390
340           GOSUB 440
350           GOSUB 520
360       NEXT Y
370 NEXT X
380 RETURN
390 '-------- Compute Eye Coordinates ---------------
400       XE= -X*SN1 + Y*CN1
410       YE= -X*CN1*CN2 - Y*SN1*CN2 + Z*SN2
420       ZE= -X*SN2*CN1 - Y*SN2*SN1 - Z*CN2 + D
430 RETURN
440 ' -------- Compute Screen Coordinates ------------
450 ' We assume s1 = s2 = 1 so that both window and viewport coordinates
460 ' will be exactly same. [see Eqs. (3.39) and (3.40).]
470       XS= (VD/S)*(XE/ZE)                    ' Viewport coordinates
480       YS= (VD/S)*(YE/ZE)
490       XS= XS + TX     : XS= SCF*XS          ' PC device coordinates
500       YS= -YS + TY
510 RETURN
520 '---------- Plotting Routine ----------------------
530 IF FLAG=0 THEN 540 ELSE 570
540       FLAG= 1 : FLAG2=0
550       OLDX=XS : OLDY=YS
560 RETURN
570       DELTAX=OLDX - XS:IF DELTAX=0 THEN DELTAX=1
580       DELTAY= OLDY - YS
590       SLOPE= DELTAY/DELTAX
600       TEMPY=OLDY
610       NEWX= INT(OLDX) + 1
620       FOR TEMPX =NEWX TO XS
630           FLAG1=1
640           TEMPY=TEMPY + SLOPE
650           IF TEMPX <0 OR TEMPX > 639 THEN FLAG1=0 : FLAG2=0 : GOTO 700
660           IF TEMPY <0 OR TEMPY > 199 THEN FLAG1=0 : FLAG2=0
670           IF TEMPY <= YMIN(TEMPX) THEN 730
680           IF TEMPY >= YMAX(TEMPX) THEN 780
690           FLAG2=0
700       NEXT TEMPX
710       OLDX=XS: OLDY=YS
720 RETURN
730       YMIN(TEMPX)=TEMPY
740       IF FLAG1=0 THEN 770
750       IF FLAG2=0 THEN PSET (TEMPX,TEMPY): FLAG2=1
760       LINE -(TEMPX,TEMPY)
770       IF TEMPY < YMAX(TEMPX) THEN 700
780       YMAX(TEMPX)=TEMPY
790       IF FLAG1=0 THEN 700
800       IF FLAG2=0 THEN PSET (TEMPX,TEMPY): FLAG2=1
810       LINE -(TEMPX,TEMPY)
820       GOTO 700
```

PROGRAM 5.3 Plot of 3-D Functions with Hidden Lines Removed

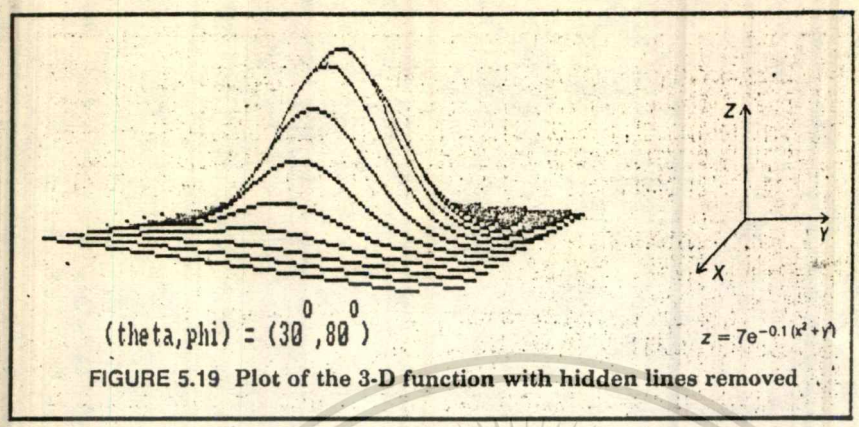


FIGURE 5.19 Plot of the 3-D function with hidden lines removed

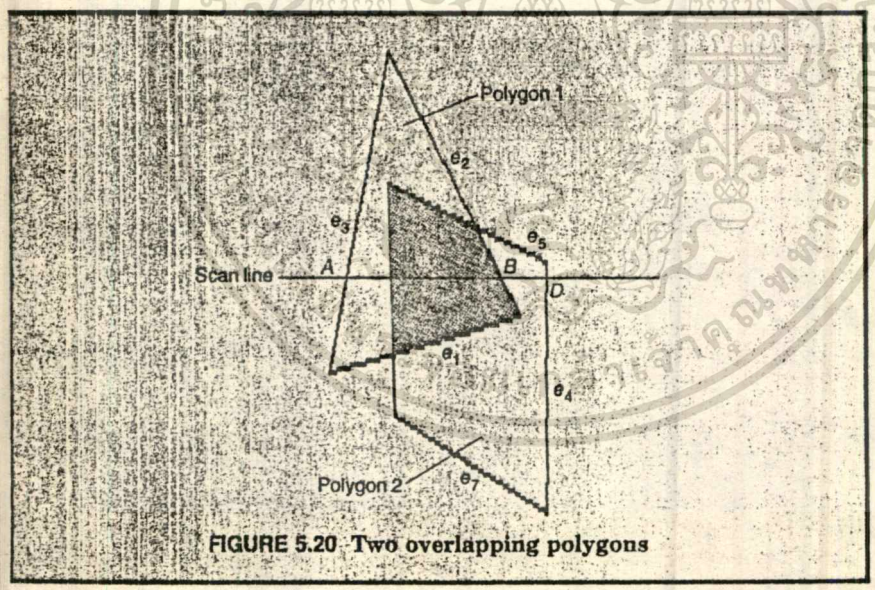


FIGURE 5.20 Two overlapping polygons

ในแต่ละ scan line ที่เวลาหนึ่ง การ process เส้น scan line นี้จะทำจากด้านบนจอภาพไปจนถึงด้านล่างของจอภาพ Algorithm จะทำการตรวจสอบ window บนจอภาพไปตามลำดับต่อเนื่อง แต่ละ window จะประกอบด้วย 1 เส้น scan line high และ scan line wide ของ screen เราจะใช้เทอมของ screen conversion เพื่อแทนกรรมวิธีการหารูปแบบของ dot ที่เหมาะสมกับรายละเอียดของภาพ

5.5.1 The depth buffer algorithm

·Depth buffer Algorithm คือรูปแบบของ hidden surface algorithm ที่ง่ายที่สุดและเป็นวิธีตรงไปตรงมา วัตถุประสงค์ก็คือให้คำนวณหา Intensity (color) ของแต่ละ pixel สำหรับ hidden surface display ค่า Intensity มีพื้นฐานขึ้นอยู่กับสีความใกล้ไกลของรูปหลายเหลี่ยมกับผู้สังเกตุ

·Algorithm

โดยพื้นฐาน Algorithm นี้จะประกอบด้วย Array 2 Array คือ Intensity (X,Y) และ depth (X,Y) ของแต่ละ pixel (X,Y) ที่อยู่บน Screen Intensity Array ก็คือ buffer ที่เก็บค่าของสีแต่ละ pixel depth array ประกอบไปด้วยค่า Zs ของจุดแต่ละจุดที่ใกล้กับผู้สังเกตุที่สุดในแต่ละตำแหน่ง Pixel Intensity Array ในสภาวะเริ่มต้นนั้นจะถูก set ให้เป็นสีของ back ground และ depth array จะถูก set ค่าให้เป็น 1 ซึ่งเป็นค่าที่มากที่สุดของ Zs (ดูในสมการ 4.41)

สำหรับรูปหลายเหลี่ยมแต่ละรูป เราจะทำจากขั้นตอนที่ 1 จนถึงขั้นตอนที่ 4 ของจุด (X,Y) ทั้งหมดที่มีอยู่ภายในรูปหลายเหลี่ยมนั้น

ขั้นตอนที่ 1 กำหนด Plane equation ของรูปหลายเหลี่ยมแต่ละรูป โดยการนำจุดของรูปหลายเหลี่ยมที่ถูกให้ค่าไว้ในเทอมของ screen coordinate (Xs, Ys, Zs)จุดเหล่านี้จะทำให้ได้สมการระนาบดังนี้

$$AXs + BYs + CYs + D = 0$$

ต่อมาก็ทำการ solve สมการระนาบเพื่อหาค่า Zs, $Zs = (-D-AXs-BYs)/C$ (5.14)

ขั้นตอนที่ 2 คำนวณหาความลึก Z(s) ของ polygow ที่ตำแหน่ง pixel นั้น(X,Y) โดยปกติแล้ว Pixel coordinate (device coordinate) แตกต่างไปจาก screen coordinate ดังนั้นเราจำเป็นต้องมีการเปลี่ยน screen coordinate ไปเป็นรูปแบบ device coordinate สำหรับตัวอย่างนี้เรา Transform coordinate ให้อยู่ในรูปแบบ device coordinate โดยการใช้สมการ 3.43

สำหรับในmode (320x 200) สมการcoordinate คือ

$$X = 1.2Xs * 160$$

$$Y = 100-Ys$$

เพราะฉะนั้นการคำนวณหาค่าความลึกที่ pixel ตำแหน่ง coordinate ต่าง ๆ (X,Y) ของสมการ (5 .14) จะหาได้โดยที่

$$Xs = (X-160)/1.2$$

$$Ys = 100-Y$$

ขั้นตอนที่ 3 ถ้า Ys depth (X,Y) ก็ทำการ set ค่า (X,Y) = Zs และ Intensity (X,Y) เป็นสีของรูปหลายเหลี่ยมนั้น ๆ ถ้า Zs depth (X,Y) ก็ไม่ต้องเปลี่ยนค่า depth และ Intensity ภายหลังรูปหลายเหลี่ยมได้ถูกกระทำกรรมวิธีหมดแล้ว ก็จะได้ค่าทั้งหมดของจอภาพอยู่ภายใน Intensity Array และเป็นผลลัพธ์ที่จะนำไปใส่ไว้ใน display buffer

เราไม่จำเป็นต้องหาค่า Zs ของแต่ละจุดบนเส้นสแกน line โดยที่เราต้องดัดแปลงโดยอาศัยประโยชน์จากความจริงว่า รูปหลายเหลี่ยมเป็นรูปพื้นผิวแบนราบดังนั้น ถ้าเราคำนวณหาค่า Zs ที่ตำแหน่ง (X,Y) ได้แล้วเราก็คำนวณหาค่า (X+1,Y) ได้โดยมีค่า

$$Z's = Zs - \frac{A}{C}(\Delta Xs) \qquad (5.15)$$

และค่า (ΔXs) ที่ตำแหน่ง (X+1,Y) มีค่าดังนี้

$$Xs = \frac{((X+1)-160)}{1.2} - \frac{(X-160)}{1.2} = \frac{1}{1.2}$$

ตัวอย่าง รูป 5.20 เป็นรูปของการ Projection รูปหลายเหลี่ยมลงบนระนาบ X,Y ส่วนที่ถูกบังแสดงให้เห็นด้วย dash line รูปหลายเหลี่ยมทั้งสองนี้คือ ภาพฉายของพื้นผิวที่ 1 ของปิรามิด ของพื้นผิวที่ 2 ของลูกบาศก์ในรูป 5.14 ซึ่งเราได้คำนวณหาค่าสมการระนาบทั้ง 2 ของรูปหลายเหลี่ยมนี้ในระบบ Screen Coordinate ในส่วน 5.3.10 ซึ่งระนาบทั้ง 2 มีค่าดังนี้

รูปหลายเหลี่ยม 1 = -8.211Xs + 9.242Ys +3399.316Zs - 2645.498

= 0 (5.16 a)

รูปหลายเหลี่ยม 2 = 19.56Xs + 4.758Ys + 2023.997Zs - 857.85

= 0 (5.16 b)

โดยสมมุติให้ Back ground มีสีดำ และสีในรูปหลายเหลี่ยม 1 เป็นสีเขียว และรูปหลายเหลี่ยม 2 เป็นสีแดง และกำหนด code สีดำนี้ สีดำ= 0, สีเขียว = 1, สีแดง = 2, ต่อมาก็กำหนดสภาวะเริ่มต้นของ Pixel ทั้งหมดบนจอภาพ

Intensity (X,Y) = 0

Depth (X,Y) = 1

ขณะนี้เราก็พร้อมที่จะทำ Scan convert รูปหลายเหลี่ยม 1 และทำการ Solving สมการระนาบเพื่อหาค่า Zs ของสมการระนาบที่กล่าวมาแล้ว

$$Zs = \frac{2645.498+8.211Xs+9.242Ys}{3399.316} \qquad (5.16c)$$

ตามที่แสดงในรูป 5.21(a) สำหรับรูปหลายเหลี่ยม 1 Ys Coordinate อยู่ในช่วง -29.73 จนถึง 48.16 และ Xc Coordinate อยู่ในช่วง -16.87 ถึง 29.52 ถ้าแทนในเทอมของ Pc device coordinate (Mode 320x200) จะได้ค่า Y อยู่ในช่วง 52 ถึง 130 และค่า X อยู่ในช่วง 140 ถึง 195 เพราะฉะนั้น Scan conversion Process จะเริ่มต้นที่ scan line Y ที่ 52 และ scan line ที่ 52 จะสัมผัสกับจุด 3 (160,52) ของรูปหลายเหลี่ยมทำให้ได้ค่า Zs = 0.909 เพราะว่า Zs depth (160,52) = 1, จะได้ค่า Array (160,52) มีค่า = 0.909 และ Inversity (160,52) = 1 (green) ค่าเหล่านี้เป็นเพียงค่าสภาวะของจุดที่ตำแหน่ง (160,52) ของเส้น scan line นี้

ต่อไปเราก็เพิ่มค่า Y อีก 1, Y = 53 และก็คำเนินกรรมวิธี scan conversion ต่อไป

เวลาต่อมา Algorithm ได้ทำการคำเนินกรรมวิธีลงมาจนถึง scan line Y ที่ 110 หรือ (Ys = -10) scan line นี้ตัดรูปหลายเหลี่ยม 2 ครั้ง ครั้งแรกที่ A และอีกครั้งที่ B ในการหาจุดตัดของด้านของรูปหลายเหลี่ยม จะใช้สมการ (5.9) ค่า Xs coordinate ของจุดที่มีการตัดกันเมื่อ Ys = -10 คือจุด -12.60 ต่อจากนั้นก็ใช้สมการ (5.16 c) หาค่า Zs ที่ตำแหน่ง (-12.60, -10) จะได้ค่า Zs= 0.720 Screen Coordinate ของจุดตัด B และการหาค่าความลึกดังกล่าวก็หาได้โดยวิธีคล้าย ๆ กัน และหาได้ค่าดังนี้ (Xs, Ys, Zs) = (26.33, -10, 0.814)

ต่อไปเราก็Transform ค่า Screen Coordinate เหล่านี้ให้อยู่ในรูปแบบของ device Coordinate

จุด A = (-12.60,-10) (144,110)

จุด B = (26.33,-10) (191,110)

เราสังเกตุได้ว่ามีจุดทั้งหมด 47 จุดที่ต้องหาค่าบนเส้น scan line (191-144 =47) โดยการใช้สมการ (5.15) และจะหาค่าความลึกของ Pixel เหล่านี้ได้ สำหรับตัวอย่างค่า Zs ที่จุด (144,110) คือ 0.720, และค่า Zs ที่ตำแหน่ง(145,110) ก็จะได้ค่าดังนี้

$$Zs = 0.720 -(-8.211/3399.316)(1/1.2) = 0.720 + 0.002 = 0.722$$

เพราะว่า Zs depth (145,110) ซึ่งมีค่า = 1 ดังนั้นจะ set ค่า depth (45,110) ให้มีค่าเท่ากับ 0.722 และ Intensity (145,110) ให้มีค่า = 1 กรรมวิธีดังกล่าวนี้จะทำซ้ำกัน Pixel ที่เหลืออยู่บนเส้น scan line และจะสิ้นสุดภายหลังการ Process ที่เส้นสแกน line เส้นสุดท้ายคือที่ Y =130, ก็เป็นอันสิ้นสุดกรรมวิธีการ scan conversion ของรูปหลายเหลี่ยมรูปที่ 1

สำหรับรูปหลายเหลี่ยมที่ 2 ค่า Ys Coordinate จะอยู่ในช่วง -65.53 จนถึง 15.83 และ Xs อยู่ในช่วง -1.37 ถึง 36.43 ในเทอม Pc device Coordinate ค่า Y ของรูปหลายเหลี่ยมนี้จะอยู่ในช่วง 84 ถึง 166 และ X อยู่ในช่วง 158 ถึง 204 เพราะฉะนั้นเส้น scan line จะเริ่มต้นที่

Y = 84 และทำการ scan คิดต่อกันไปเหมือนรูปหลายเหลี่ยมที่ 1 สำหรับ scan line Y 110 เราจะทำการตรวจสอบว่าค่าของ Array มีการเพิ่มขึ้นและเปลี่ยนแปลงอย่างไรและบันทึกไว้ scan line Y 110 คิดค้านรูปหลายเหลี่ยมที่ 2: 2 ครั้ง คือที่จุด C และ D ต่อมากรรมวิธี scan conversion จะเริ่มต้นที่ตำแหน่ง C Screen Coordinate ของจุดตัดและค่าความลึกจะมีค่าดังนี้

(Xs,Ys,Zs) = (1.34,-10,0.460) (X,Y) = 158, 110

ภายหลังการ Process ของรูปหลายเหลี่ยมที่ 1 ค่าสภาวะของ Array (158,110) ของจุด C มีค่าดังนี้

Intensity (158, 110) = 1

Depth (158, 110) = 0.748

สำหรับรูปหลายเหลี่ยม 2 โดยใช้สมการ (5.16 b) หาค่า Zs ที่จุด C มีค่า 0.466 เพราะว่า Zs depth (158,110)= 0.748 เราจะทำการ set ค่า depth (158,110) เป็น 0.460 และ set Intensity (158,110) ให้มีค่า = 2 (สีแดง) นี่ย่อมหมายความว่ารูปหลายเหลี่ยม 2 จะอยู่ใกล้จุดสังเกตุมากกว่ารูปหลายเหลี่ยมที่ 1 รูป 5.22 แสดงให้เห็นค่าของ Array ที่เปลี่ยนแปลงไปตามที่ Process ของจุดที่เหลือบนเส้น scan line นั้น

ในการเลื่อนจุดจากจุดหนึ่งบนเส้น scan line ไปยังอีกจุดหนึ่งบนเส้น scan line เราสามารถหา Coordinate ของ X ใหม่ได้ดังนี้

$$Xi + 1 = Xi + \frac{1}{M} \qquad (5.17)$$

เมื่อ M คือค่า slop ของค้านและมีค่า $=\Delta Y/\Delta X$ ในที่นี้ $\Delta Y = 1$ ดังนั้น Invert slope มีค่า $= 1/M$ ซึ่งมีค่า $=\Delta X$

depth buffer algorithm ไม่นิยมใช้ในทางปฏิบัติเพราะจะต้องใช้ขนาด Array ของ depth และ Intensity ใหญ่มาก การสร้างภาพ Raster ที่มีขนาด 320 x 200 จะต้องใช้ Array depth และ Intensity มีขนาดถึง 6400 เหมือนกับ display buffer จะต้องเตรียมหน่วยความจำเพื่อใช้ใน Intensity Array (มีขนาด 16 K ในเครื่อง IBM PC), depth array ที่เหลือก็ยังคงมีขนาดใหญ่มาก วิธี scan line

coherence ซึ่งจะกล่าวถึงในส่วนต่อไปจะช่วยลดขนาด memory ที่ใช้ในการสร้างภาพนั้น ๆ

Penetrating Polygon

ถึงตอนนี้เรายังไม่มีเทคนิควิธีใดที่จัดการเกี่ยวกับรูปหลายเหลี่ยมที่ทะลุถึงกัน ขณะนี้เราได้ขยายเพิ่มเติม Algorithm ในการจัดการเกี่ยวกับรูปหลายเหลี่ยมที่ทะลุถึงกัน เช่นในรูป 5.23 ในการทำแบบนี้เราจะต้องแบ่งรูปหลายเหลี่ยมหนึ่ง ๆ ออกเป็น 2 ส่วน โดยการสร้างด้าน ls ขึ้นมา ต่อมาเราต้องดัดแปลง Algorithm ในการคืนหาจุด Penetration บนเส้น scan line ที่เราต้องดำเนินกรรมวิธี

Program 5.4 แสดงให้เห็นการเปลี่ยนแปลง Algorithm ใน computer program ได้อย่างไรใน program นี้สามารถดำเนินกรรมวิธีได้ทั้ง non penetration และ Penetration (ไม่ทะลุถึงกันกับทะลุถึงกัน) พร้อมกับDATA ที่ให้ไว้ในบรรทัด 610-630, เราจะได้ Screen output ดังแสดงในรูป 5.24

5.5.2 Scan line coherence algorithm

วิธีนี้คือวิธีที่ขยายเพิ่มเติมมาจาก depth buffer algorithm ความแตกต่างระหว่าง algorithm ทั้งสองคือว่า เราจะไม่จับต้องเพียงรูปหลายเหลี่ยมเดียว แต่เราจะจับต้องกับรูปหลายเหลี่ยมทั้งหมดที่กำหนดเป็นวัตถุนั้น ในทำนองอื่น Algorithm scan line convert รูปหลายเหลี่ยมทั้งหมดในภาพนั้นที่ scan line หนึ่ง ๆ ในเวลาหนึ่ง Algorithm นี้จะดำเนินการไปตามนี้ คือแต่ละเส้น scan line จะดำเนินตามขั้นตอน step ที่ 1 จนถึง step ที่ 3

Step ที่ 1 ให้ค่า Pixel ทั้งหมดบนเส้นสแกน lineหนึ่งมีค่า depth (X) = ∞ และ Intensity (X) = สี Blackground หมายเหตุ Array ของ depth และ Intensity มีขนาดมิติเดียว

Step ที่ 2 สำหรับแต่ละรูปหลายเหลี่ยมในภาพนั้น, หาค่า Pixel X ทั้งหมดบนเส้นสแกน line Y ที่อยู่ภายในรูปหลายเหลี่ยม สำหรับแต่ละ Pixel ให้ดำเนินกรรมวิธีตามนี้

Pixel	$[x, y]$	[144, 110]	[158, 110]	[191, 110]	[204, 110]
Initial value	Depth	1	1	1	1
	intensity	0	0	0	0
After processing polygon 1	Depth	0.721	0.748	0.814	1
	intensity	1	1	1	0
After processing polygon 2	Depth	0.721	0.460	0.190	0.097
	intensity	1	2	2	2

Polygon 1 ⟷ Polygon 2

FIGURE 5.22 Array values after each scan conversion process

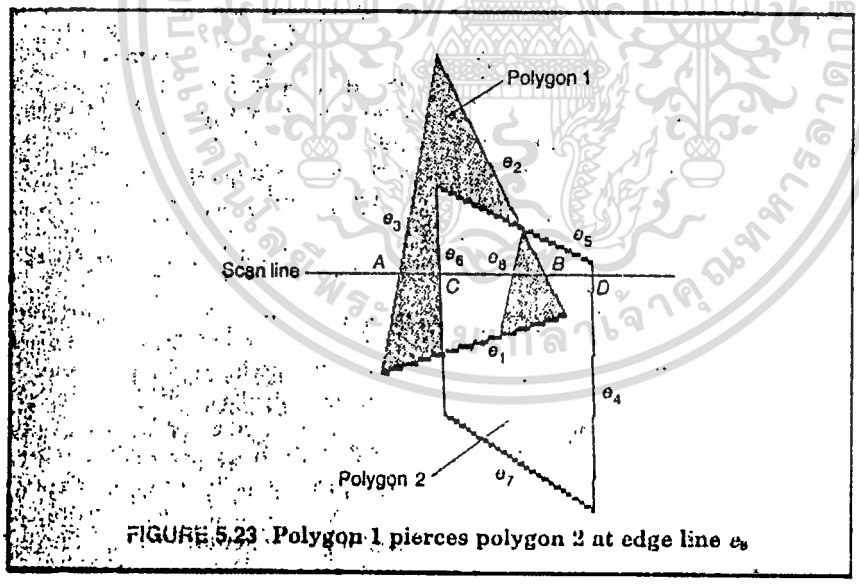


FIGURE 5.23 Polygon 1 pierces polygon 2 at edge line e_8

```
100 '   Program 5.4: Depth-Buffer Algorithm
110 '   Routine to process either penetrating or nonpenetrating polygons
120 '         BACKGROUND COLOR          = BLUE
130 '         POLYGON 1                 = GREEN
140 '         POLYGON 2                 = RED
150 '      -------- MAIN ROUTINE --------
160       CLS: KEY OFF:  SCREEN 0, 1:  WIDTH 80:    COLOR 2, 0
170       PRINT TAB(20) " Specify the types of polygons"
180       PRINT: PRINT TAB(20) "1.  Solid Polygons"
190       PRINT: PRINT TAB(20) "2.  Penetrating Polygons"
200       PRINT: PRINT TAB(20) "Enter Choice ==> "
210       POKE 106, 0
220       CMD$ = INKEY$:  IF CMD$ <> "" THEN 220
230       CMD$ = INKEY$:  IF CMD$ = "" THEN 230
240       IF MID$(CMD$, 1, 1) = "1" THEN SS = 1: GOTO 270
250       IF MID$(CMD$, 1, 1) = "2" THEN SS = 2: GOTO 270
260       GOTO 210
270       CLS:  SCREEN+1, 0:  OUT &H3D9, 1
280       GOSUB 350
290       GOSUB 410
300       GOSUB 750
310       GOSUB 910
320       GOSUB 1060
330       GOSUB 1250
340   END
350 '     -------- DEFINE VIEWING PARAMETERS --------
360       D = 10:  VD = 500:  THETA = 57:  PHI = 68:  TX = 160:  TY = 100
370       PI = 3.141593:  SCF = 1.2
380       THETA = THETA * PI / 180:  PHI = PHI * PI / 180:  S = 1
390       SN1 = SIN(THETA):  SN2 = SIN(PHI):  CN1 = COS(THETA):  CN2 = COS(PHI)
400   RETURN
410 '     -------- VERTEX ARRAY --------
420   NV(1) = 3:  NV(2) = 4:  NV(3) = 4:  K1 = 7:  K2 = 10:  W = K2 / (K2 - K1)
430   DIM V(3, 4, 3), EC(3, 4, 3), SC(3, 4, 3), SO(3, 4, 2)
440   FOR I = 1 TO 3
450       FOR J = 1 TO NV(I)
460           READ X, Y, Z
470           V(I, J, 1) = X:  V(I, J, 2) = Y:  V(I, J, 3) = Z
480           GOSUB 640
490           EC(I, J, 1) = XE:  EC(I, J, 2) = YE:  EC(I, J, 3) = ZE
500           GOSUB 690
510           SC(I, J, 1) = XS:  SC(I, J, 2) = YS:  SC(I, J, 3) = ZS
520           SO(I, J, 1) = XS1:  SO(I, J, 2) = YS1
530       NEXT J
540   NEXT I
550   IF SS = 1 THEN LOCATE 2, 15: PRINT "SOLID POLYGONS":  RETURN
560   LOCATE 2, 12: PRINT "PENETRATING POLYGONS"
570   FOR J = 1 TO NV(2)
580       SC(2, J, 1) = SC(3, J, 1): SC(2, J, 2) = SC(3, J, 2):
          SC(2, J, 3) = SC(3, J, 3): SO(2, J, 1) = SO(3, J, 1):
          SO(2, J, 2) = SO(3, J, 2)
590   NEXT J
600   RETURN
610   DATA  1,1,0,  0,1,0,  0,0,1
620   DATA  1,2.5,0,  1,2.5,1,  1,1.5,1,  1,1.5,0
630   DATA  1,1.5,0.5,  0,1.5,0.5,  0,0,0.5,  1,0,0.5
640 '     -------- EYE COORDINATES --------
650       XE = -X * SN1 + Y * CN1
660       YE = -X * CN1 * CN2 - Y * SN1 * CN2 + Z * SN2
670       ZE = -X * SN2 * CN1 - Y * SN2 * SN1 - Z * CN2 + D
680   RETURN
690 '     -------- SCREEN COORDINATES --------
700       XS1 = (VD / S) * (XE / ZE)
710       YS1 = (VD / S) * (YE / ZE)
720       ZS = W * (1 - K1 / ZE)
730       XS = XS1 * SCF + TX:"  YS = TY - YS1
740   RETURN
750 '     -------- SURFACE ARRAY --------
```

PROGRAM 5.4 Depth-buffer Algorithm

```
760     DIM SF(2, 1, 5), NPS(2, 1)
770     NS(1) = 1: NS(2) = 1
780     FOR I = 1 TO 2:  FOR J = 1 TO NS(I):  READ NPS(I, J):  NEXT J:  NEXT I
790     FOR I = 1 TO 2
800         FOR J = 1 TO NS(I)
810             FOR K = 1 TO NPS(I, J)
820                 READ SF(I, J, K)
830             NEXT K
840         NEXT J
850     NEXT I
860 RETURN
870 DATA  4
880 DATA  5
890 DATA  1, 2, 3, 1
900 DATA  1, 2, 3, 4, 1
910 '     --------  VISIBLE EDGE ARRAY  --------
920 DIM E(2, 4, 2), VNE(2)
930 FOR I = 1 TO 2
940     M = 1
950     FOR J = 1 TO NS(I)
960         E1 = SF(I, J, 1)
970         FOR K = 2 TO NPS(I, J)
980             E2 = SF(I, J, K)
990             E(I, M, 1) = E1:  E(I, M, 2) = E2
1000            E1 = E2: M = M + 1
1010        NEXT K
1020        NVE(I) = M - 1
1030    NEXT J
1040 NEXT I
1050 RETURN
1060 '     --------  Ymin AND Ymax FOR BOTH POLYGONS  --------
1070    DIM YMIN(2), YMAX(2), XMIN(2), XMAX(2)
1080    FOR I = 1 TO 2
1090        XMIN(I) = SC(I, 1, 1): XMAX(I) = SC(I, 1, 1)
1100        YMIN(I) = SC(I, 1, 2): YMAX(I) = SC(I, 1, 2)
1110        FOR J = 2 TO NV(I)
1120            IF XMIN(I) > SC(I, J, 1) THEN XMIN(I) = SC(I, J, 1)
1130            IF XMAX(I) < SC(I, J, 1) THEN XMAX(I) = SC(I, J, 1)
1140            IF YMIN(I) > SC(I, J, 2) THEN YMIN(I) = SC(I, J, 2)
1150            IF YMAX(I) < SC(I, J, 2) THEN YMAX(I) = SC(I, J, 2)
1160        NEXT J
1170        XMIN(I) = INT(XMIN(I)): XMAX(I) = INT(XMAX(I))
1180        YMIN(I) = INT(YMIN(I)): YMAX(I) = INT(YMAX(I))
1190    NEXT I
1200    IF XMIN(1) > XMIN(2) THEN XL = XMIN(2) ELSE XL = XMIN(1)
1210    IF XMAX(1) > XMAX(2) THEN XU = XMAX(1) ELSE XU = XMAX(2)
1220    IF YMIN(1) > YMIN(2) THEN YL = YMIN(2) ELSE YL = YMIN(1)
1230    IF YMAX(1) > YMAX(2) THEN YU = YMAX(1) ELSE YU = YMAX(2)
1240 RETURN
1250 '     --------  BOUNDARY COORDINATES OF BOTH POLYGONS  --------
1260 DIM ISP(2, 2), DX(2, 2), DEPTH(320), INTENSITY(320)
1270    L3 = 0: LX = XL \ 4: LX1 = LX * 4 + 1
1280    UX = XU \ 4 + 1: DE = 80 - (UX - LX):  UX1 = UX * 4
1290    FOR Y = YL TO YU
1300        C = 0
1310        FOR I = XL TO XU:  DEPTH(I) = 1:  INTENSITY(I) = C: NEXT I
1320        FOR I = 1 TO 2
1330            IF (Y < YMIN(I)) OR (Y > YMAX(I)) THEN 1450
1340            KK = 0
1350            FOR J = 1 TO NVE(I)
1360                A = E(I, J, 1): B = E(I, J, 2)
1370                GOSUB 1490
1380                IF NCUT <> 1 THEN 1410
1390                KK = KK + 1: IF I = 1 THEN C = 1 ELSE C = 2
1400                ISP(I, KK) = INT(XI): DX(I, KK) = (XI - TX) / SCF
1410            NEXT J
1420            IF ISP(I, 1) > ISP(I, 2) THEN SWAP ISP(I, 1), ISP(I, 2):
                                                SWAP DX(I, 1), DX(I, 2)
1430            IF (KK = 0) AND (Y = LB) THEN 1450
1440            GOSUB 1680
1450        NEXT I
1460        GOSUB 1910
1470    NEXT Y
1480 RETURN
1490 '     --------  INTERSECTION TEST  --------
1500    AX = SC(I, A, 1): AY = SC(I, A, 2): BX = SC(I, B, 1): BY = SC(I, B, 2)
1510    CX = 0: CY = Y: DX = 319: DY = Y
```

PROGRAM 5.4 (continued)

```
1520    R1 = AX - BX:    R2 = AY - BY:    R3 = CX - DX:    R4 = CY - DY
1530    S1 = SGN(R1):    S2 = SGN(R2):    S3 = SGN(R3):    S4 = SGN(R4)
1540    A1 = AX: B1 = BX: A2 = AY: B2 = BY: C1 = CX: D1 = DX: C2 = CY: D2 = DY
1550    IF S1 > 0 THEN A1 = BX:  B1 = AX
1560    IF S2 > 0 THEN A2 = BY:  B2 = AY
1570    IF S3 > 0 THEN C1 = DX:  D1 = CX
1580    IF S4 > 0 THEN C2 = DY:  D2 = CY
1590 '     --------  MINMAX TEST FOR EDGE OVERLAPPING  --------
1600    IF (B1 < C1) OR (D1 < A1) OR (B2 < C2) OR (D2 < A2) THEN NCUT = 0:
                                                              RETURN
1610    M1 = R2 / R1:  M2 = R4 / R3:  M3 = M2 - M1
1620    IF (M3 = 0) OR (ABS(M3) <= .0001) THEN NCUT = 0: RETURN
1630 '     --------  COMPUTE INTERSECTION POINT  --------
1640    Z0 = AY - CY:                    Z1 = M1 * AX - M2 * CX:
        Z2 = (AY - M1 * AX) * M2:    Z3 = (CY - M2 * CX) * M1
1650    XI = (Z0 - Z1) / M3:             YI = (Z2 - Z3) / M3
1660    IF (XI < A1) OR (XI > B1) OR (XI < C1) OR (XI > D1) OR (YI < A2) OR
           (YI > B2) THEN NCUT = 0 ELSE NCUT = 1
1670 RETURN
1680 '     --------  UPDATE DEPTH & INTENSITY  --------
1690    XI = DX(I, 1):  YI = TY - Y
1700    GOSUB 1810
1710    X0 = ISP(I, 1):    X1 = ISP(I, 2)
1720    IF ZDEPTH >= DEPTH(X0) THEN 1740
1730       INTENSITY(X0) = C:  DEPTH(X0) = ZDEPTH
1740    X0 = X0 + 1:  RATIO = XX / ZZ
1750    FOR K = X0 TO X1
1760       ZDEPTH = ZDEPTH - RATIO
1770       IF ZDEPTH > DEPTH(K) THEN 1790
1780          DEPTH(K) = ZDEPTH:  INTENSITY(K) = C
1790    NEXT K
1800 RETURN
1810 '     --------  COMPUTE DEPTH  --------
1820    A1 = SO(I, 1, 1):  A2 = SO(I, 1, 2):  A3 = SC(I, 1, 3)
1830    B1 = SO(I, 2, 1):  B2 = SO(I, 2, 2):  B3 = SC(I, 2, 3)
1840    C1 = SO(I, 3, 1):  C2 = SO(I, 3, 2):  C3 = SC(I, 3, 3)
1850    AA = B1 - A1:  BB = B2 - A2:  CC = B3 - A3:
        DD = C1 - A1:  EE = C2 - A2:  FF = C3 - A3
1860    XX = BB * FF - CC * EE: YY = CC * DD - AA * FF: ZZ = AA * EE - BB * DD
1870    T1 = XX * A1 + YY * A2 + ZZ * A3
1880    T2 = XX * XI + YY * YI
1890    ZDEPTH = (T1 - T2) / ZZ
1900 RETURN
1910 '     --------  PLOT  --------
1920    DEF SEG = &HB800
1930    R = Y MOD 2
1940    IF L3 <> 0 THEN 1960
1950    IF R = 1 THEN IO = &H2000 + 80 * (Y-1)/2 + LX :  IE = 80 * (Y+1)/2 + LX
                 ELSE IO = &H2000 + 80 * Y / 2 + LX:     IE = 80 * Y / 2 + LX
1960    FOR L1 = LX1 TO UX1 STEP 4
1970       CC = 0:  EX = 64
1980       FOR L2 = 1 TO 4
1990          CC = CC + INTENSITY(L1 + L2 - 1) * EX
2000          EX = EX / 4
2010       NEXT L2
2020       IF R = 1 THEN POKE IO, CC: IO = IO + 1:
                    ELSE POKE IE, CC: IE = IE + 1
2030    L3 = L3 + 1
2040    NEXT L1
2050    IF R = 1 THEN IO = IO + DE ELSE IE = IE + DE
2060 RETURN
```

PROGRAM 5.4 Depth-buffer Algorithm

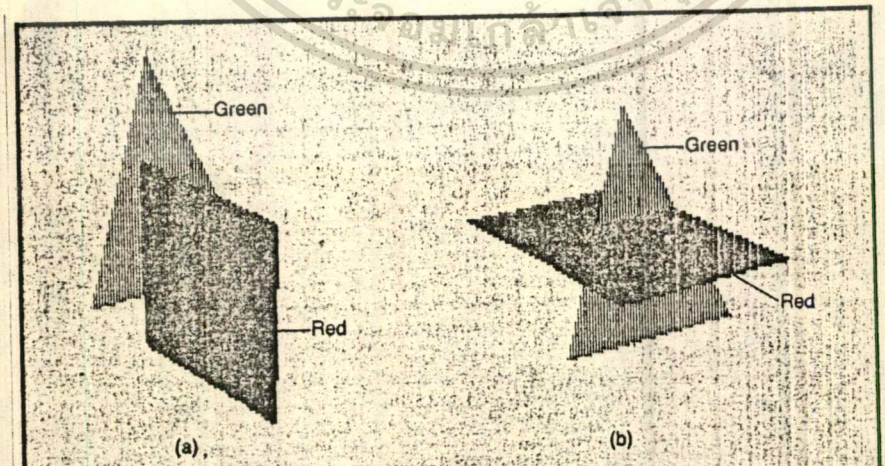


FIGURE 5.24(a) Scan conversion results of nonpenetrating polygons on the IBM PC (320 × 200) and (b) scan conversion results of penetrating polygons on the IBM PC (320 × 200 mode)

1. หาค่าความลึก Zs ของรูปหลายเหลี่ยมที่จุด (X,Y)

2. ถ้าค่า Zs < depth (X), set depth (X) hi = Zs Intensity (X) = สีที่กำหนดให้แก่รูปหลายเหลี่ยมนั้น

.step ที่ 3 ภายหลังที่รูปหลายเหลี่ยมทั้งหมดบนเส้นสแกน line ได้ถูกคำนินการมวิธีเรียบร้อยแล้วค่าที่อยู่ใน Intensity (X) จะแสดงถึงผลลัพธ์ที่ได้และนำออกแสดงผลสู่จอภาพ

.Algorithm ที่กล่าวมาข้างบนนี้จะทำงานได้แต่มันไม่มีประสิทธิภาพ เพราะว่ามันไม่มีขนาดและตำแหน่งที่สัมพันธ์กับพื้นผิวนั้น แต่มันขึ้นอยู่กับภายใน เรียกว่า coherence

.Visility ของด้านหนึ่งเปลี่ยนแปลงเมื่อมันตัดกับด้านอื่น ๆ ตามที่แสดงไว้ในรูป 5.20 เราสามารถนำคุณสมบัตินี้มาสร้างรายละเอียดของแต่ละด้านที่ไม่มีการตัดกัน ต่อมาเราก็ต้องตรวจสอบจุดของแต่ละ segment สำหรับแต่ละ scan line เราจะทำกันเพียงแต่กลุ่มของด้านที่มีการตัดกันเท่านั้น และเราจะกำหนดด้านดังกล่าวนี้ว่า active edge ในการช่วยสนับสนุน edge coherence Algorithm นี้เราจะต้องสร้างตาราง 3 ตารางคือ edge tchl (Et) และ Acive edge table (AET) และ Polygon table (PT)

Edge table เราจะสร้าง edge table ที่ประกอบไปด้วยด้านทั้งหมด โดยการเรียงจาก Ys ที่มี coordinate ต่ำสุด, Et ถูกสร้างขึ้นโดยวิธี bucket sort และจะมี bucket จำนวนมากเมื่อมีเส้น scan line มาก ภายในแต่ละ Bucket ด้านจะถูกเก็บไว้ตามลำดับจากจุดปลาย X ตำแหน่งและเรียงเพิ่มขึ้นภายใน Et แต่ละส่วนจะประกอบด้วย

1. Xs Coordinate ของจุดปลายที่มีค่า Ys Coordinate น้อยที่สุด (X.min)

2. Ys coordinate ของจุดปลายด้านอื่น ๆ(Ys.max)

3. การเพิ่มค่า Xs ที่ใช้ในแต่ละ step จาก 1 scan line ไปยังตำแหน่งต่อไป (1/M)

Polygon 1		Pixel coordinates
Vertex number	$x_s,\ y_s,\ z_s$	$x,\ y$
(1)	−16.87, −29.73, 0.656	140, 130
(2)	29.52, −17.04, 0.803	195, 117
(3)	0, 48.16, 0.909	160, 52
Intersection		
A	−12.60, −10.00, 0.720	144, 110
B	26.33, −10.00, 0.814	191, 110

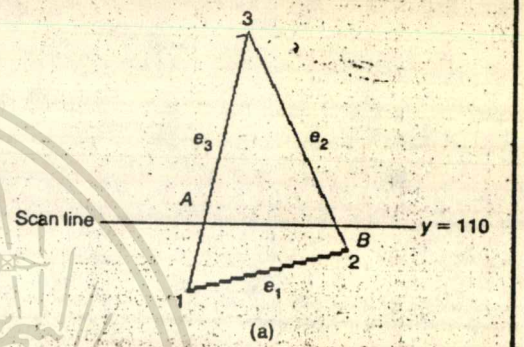


FIGURE 5.21(a) Intersections of polygon 1 with $y=110$ ($y_s = -10$)

Polygon 2		Pixel coordinates
Vertex number	$x_s,\ y_s,\ z_s$	$x,\ y$
(1)	34.62, −65.52, 0.243	202, 166
(2)	36.43, −4.34, 0.081	204, 104
(3)	−1.37, 15.83, 0.394	158, 84
(4)	−1.31, −40.54, 0.531	158, 140
Intersection		
C	−1.34, −10.00, 0.460	158, 110
D	36.26, −10.00, 0.097	204, 110

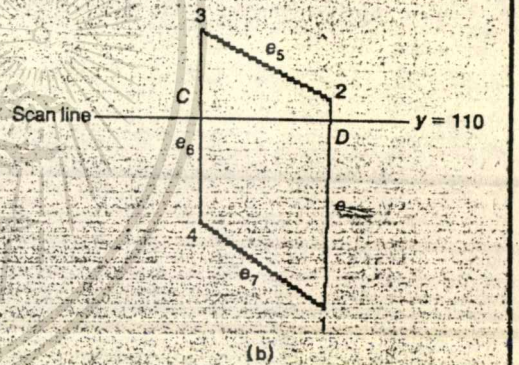


FIGURE 5.21(b) Intersections of polygon 2 with $y = 110$ ($y_s = -10$)

4. ชื่อของรูปหลายเหลี่ยม ๆ โดยแสดงถึงว่าด้านนี้เป็นของรูปหลายเหลี่ยมใด

ตัวอย่างสมมุติ ด้าน 13 ในรูป 5.20 ค่าที่อยู่ภายใน Et จะเป็นดังนี้

ด้าน 13	48.16	-16.87	1/4.62	1
	Ymax	Xsmin	1/M	Polygon number

.Active edge table

AET สามารถสร้างขึ้นในรูปแบบดังนี้ คือ เมื่อเราได้เคลื่อนย้ายไปยังเส้น scan line เส้นต่อไปจะต้องคำนวณหาจุดตัด Xs ใหม่โดยการใช้สมการ (5.17) ด้านตัดใหม่ที่ได้ด้วยเส้น scan line นี้จะถูกเพิ่มเข้าไปใน AET และด้าน AET ที่ไม่ถูกตัดนั้นก็จะเอาออกจาก AET

.Polygon table

ภายใน Pt ประกอบด้วยรายละเอียดเหล่านี้ของรูปหลายเหลี่ยมแต่ละรูป

1. สัมประสิทธิของสมการระนาบ
2. Sitading หรือข้อมูลของสีของรูปหลายเหลี่ยมแต่ละรูป
3. flag ที่แสดงตำแหน่ง Poirler ที่สัมพันธ์กับรูปหลายเหลี่ยม (ซึ่ง In/out bodean flag นี้จะถูก set ค่าเริ่มต้นให้เป็น false (อยู่ภายนอกรูปหลายเหลี่ยม) โดย flag นี้จะถูกนำมาใช้ในระหว่าง scan line process

ตัวอย่าง

เราจะใช้ Algorithm นี้กับตัวอย่าง ในรูป 5.20 Sort Et ของรูปนี้จะประกอบด้วย 17 14 13 11 12 15 ตามลำดับ และ PT มีส่วนประกอบของรูปหลายเหลี่ยม 1 และรูปหลายเหลี่ยม 2

.AET จะเก็บค่าตามตำแหน่งของ Xs ที่เพิ่มขึ้น โดยแต่ละเวลาที่ Algorithm ได้ดำเนินไปจนถึงเส้น scan line Y 110(Ys=-10) AET จะประกอบไปด้วย 13,16,12 และ 14 ตามลำดับ

ด้านเหล่านี้จะถูกดำเนินกรรมวิธีจากซ้ายไปขวา ในการดำเนินกรรมวิธี 13, ลำดับแรกเราต้อง set flag ของรูปหลายเหลี่ยม 1 เป็นจริงเพราะฉะนั้นการ scan จะอยู่ภายในรูปหลายเหลี่ยมนี้และรูปหลายเหลี่ยมนี้จะถูกพิจารณาเพราะเรา scan อยู่ในรูปหลายเหลี่ยมเดียวเท่านั้น รูปหลายเหลี่ยมนี้ต้องมองเห็นโดยตลอด โดยเหตุนี้เราจะระบายสี จากช่วง A ไปยังด้านต่อไปใน AET ด้าน 26 ที่จุดนี้ flag ของรูปหลายเหลี่ยม 2 ก็เป็นจริงเราก็ scan อยู่ภายในทั้ง 2 Polygon เราต้องตัดสินใจว่ารูปหลายเหลี่ยม 1 หรือรูปหลายเหลี่ยม 2 นี้ว่ารูปหลายเหลี่ยมใดอยู่ใกล้จุดสังเกตุมากกว่า โดยเรากำหนดค่าใกล้ไกลนี้ได้จากการหาสมการระนาบของรูปหลายเหลี่ยมทั้ง 2 เพื่อหาค่า Zs ที่ Y = 110 (Ys = -10. เมื่อ Xs มีค่าเท่ากับ การตัดกันของ Y = -10 กับด้าน 16 ค่าของจุดตัดนี้คือ (-1.34,-10) และจะอยู่ภายใน AET โดยตลอดค่า E6 การหาค่าสมการระนาบ (สมการ 5.16 a และ 5.16b) ที่ตำแหน่ง (1.36,-10) จะได้ค่า Zs มีค่าดังนี้

Zs = 0.74 ของรูปหลายเหลี่ยมที่ 1

Zs = 0.46 ของรูปหลายเหลี่ยมที่ 2

เพราะว่า 0.46 0.74 รูปหลายเหลี่ยมที่ 2 จึงอยู่ใกล้จุดสังเกตุมากกว่ารูปหลายเหลี่ยมที่ 1 โดยเหตุนี้สีของรูปหลายเหลี่ยมที่ 2 จึงถูกนำมาใช้ในช่วงของ 12 ดังกล่าวและที่จุดนี้ flag รูปหลายเหลี่ยมที่ 1 กลายเป็น 0 ดังนั้น scan ก็จะกลับมาอยู่ในรูปหลายเหลี่ยมเพียงรูปเดียวเท่านั้น คิดรูปหลายเหลี่ยมที่ 2 และสีก็เป็นสีของรูปหลายเหลี่ยมที่ 2 ต่อไปตลอดจนถึงด้าน 14 ที่จุดนี้กรรมวิธีใน การ scan conversion สิ้นสุดลงและก็จะเลื่อนไปทำการ scan เส้นใหม่ต่อไป

ลักษณะโครงงาน

เป็น Software ที่พัฒนาขึ้นมาใช้สำหรับเครื่อง IBM โดยใช้ภาษา PASCAL ในการเขียนโปรแกรมซึ่งสามารถเขียนเป็นผังงานอย่างคร่าว ๆ ได้ในหน้าถัดไป ส่วนรายละเอียดของแต่ละส่วนในผังงานจะได้กล่าวถึงต่อไป ในโปรแกรมจริงๆนั้นได้อาศัยกราฟฟิคทูลบล๊อกของบริษัทบอร์แลนช่วยด้วยเพื่อสร้าง Help และ Menu และกำหนดการแสดงภาพให้ดียิ่งขึ้น เราจะเริ่มต้นด้วยการออกแบบวัตถุขึ้นโดยวัตถุนี้จะประกอบด้วยจุดและพื้นผิวต่าง ๆ โดยจุด coordinate เหล่านี้จะเป็นแบบ World coordinate ที่อยู่ในระบบคาร์ทีเซียนและถูกนำมาเก็บไว้ใน File ชื่อ Vertex.Dta ส่วนระนาบต่างๆจะนำมาเก็บไว้ใน File ชื่อ Surface.DTA

จาก BLOCK แรกในผังงานจะทำการอ่าน File Vertex.DTA และ Surface.DTA เข้ามาโดยใช้ Procedure read_data_file ต่อมาจะทำการกำหนดจุดมอง(มุมมอง) และค่า Parameter ต่าง ๆ ที่เกี่ยวข้องเพื่อที่จะนำมาใช้เปลี่ยนจุดในระบบ World coordinate ของแต่ละจุดให้เป็นจุด coordinate ในระบบ Eye coordinate และจุดในระบบ Screen coordinate ต่อไปโดยการใช้ Precedure vertex_Array หลังจากสิ้นสุด Procedure นี้ต่อไปก็จะทำการคำนวณหา Normal vecter ของแต่ละพื้นผิวของทุกวัตถุเพื่อจะนำไปคำนวณหาด้านที่มองไม่เห็นและด้านที่มองเห็น ซึ่งในการคำนวณหา Normal vecter นี้จะต้องเรียกใช้ Procedure Normal_Array ลำดับต่อมาก็จะคำนวณหาด้านที่มองเห็นของวัตถุทุกชิ้นโดยใช้ Procedure Visibility_test ซึ่ง Precedure นี้จะทำการหาด้านที่มองเห็นและด้านที่มองไม่เห็นไว้ก่อนจากนั้นจะทำการคำนวณหาเส้นขอบของวัตถุแต่ละชิ้นที่สามารถมองเห็นได้ และในการหาด้านที่ถูกบังโดยวัตถุอื่นจะทำได้โดยการเรียกใช้ Procedure Visible_Edge_Array เมื่อได้ค่าสภาวะต่าง ๆ แล้วต่อจากนั้นก็ต้องมีการจัดลำดับความลึกของวัตถุโดยการทดสอบวัตถุทีละ 1 คู่เพื่อหาว่าวัตถุนั้นมีวัตถุชิ้นใดอยู่ข้างหน้ามันบ้างและจะนำวัตถุที่อยู่หน้ามันมาเก็บไว้เพื่ออ้างอิงต่อไป

สำหรับขั้นตอนในการจัดลำดับความลึกของวัตถุจะประกอบด้วย Procedure ต่าง ๆ ดังนี้คือ Procedure Minimax_test โดย Procedure นี้จะทำหน้าที่ตรวจสอบว่าวัตถุทั้ง 2 เหลื่อมกันหรือไม่ ในกรณีที่มีการเหลื่อมกันเกิดขึ้น (overlap=true) ก็จะนำวัตถุทั้ง2 นั้นมาหาต่อไปว่าวัตถุชิ้นใดอยู่ใกล้ตาเรามากกว่ากันโดยใช้ Procedure Determine_object_priority โดยใน Procedure นี้จะประกอบด้วยProcedure ย่อยหลาย Procedure ดังเช่นว่า Procedure Intersection_test และ Procedure Dept_test และ Procedure Containment_test โดย Procedure Determine_object_priority จะทำการเรียกใช้ Procedure Intersection_test,Procedure

Depth_test และ Procedure Containment_test จนกระทั่งสามารถกำหนดลักษณะต่าง ๆที่เกี่ยวกับด้าน, จุดตัดของวัตถุ, และการล้อมรอบของวัตถุหนึ่งกับอีกวัตถุหนึ่งและก็จะทำการเก็บค่าสถาวะต่างที่ได้นี้ในตัวแปรชื่อ Link ซึ่งในตัวแปรนี้จะเก็บเฉพาะวัตถุที่อยู่ข้างหน้ามันและบังมันเท่านั้นจากนั้นก็จะทำการทดสอบในลักษณะนี้อีกกับวัตถุชิ้นต่อไปเพื่อหาว่ายังมีวัตถุชิ้นอื่นอีกหรือไม่ที่จะมาบังมันอีกถ้าหากมีก็จะนำมาเก็บไว้ในตัวแปรสชื่อ Link อีกโดยที่ตัวแปรแบบ Link นี้จะเป็นแบบ Array

เมื่อได้ลำดับของวัตถุต่างเก็บไว้ใน Link หมดแล้วต่อไปก็จะนำเอาวัตถุที่ถูกบังมา Plot ให้หมดก่อน โดยมีการ Loop เพื่อหาวัตถุที่ถูกบังเหล่านั้นก่อนโดยที่วัตถุที่บังตัวมันจะดูได้จากตัวแปร Link ของวัตถุชิ้นนั้น จากนั้นจะทำการ Plot เฉพาะส่วนที่มองเห็นลงใน Vertual screen ในการ plot นี้จะเรียกใช้ Procedure Plot_of_line_segment โดยส่วนนี้จะทำการ Plot ส่วนของด้านที่มองเห็นเท่านั้นส่วนด้านที่มองไม่เห็นจะตัดทิ้งไป ซึ่งในที่นี้จะต้องมีการกำหนดว่าส่วนไหนของด้านมองเห็นและส่วนไหนของด้านที่มองไม่เห็นโดยต้องมีการหาจุดตัดของด้านนั้นกับด้านอื่น โดยการเรียกใช้ Procedure Intersection_test และก็ใช้ Procedure อื่น ๆ อีกในการกำหนดให้แน่ชัดไปเลยว่าส่วนนี้ของด้านนี้มองเห็นส่วนนั้นของด้านนี้มองไม่เห็น หลังจากนั้น Procedure Plot_of_line_segment จะเรียกใช้ Procedure Visible_endpoint อีกทีหนึ่งเพื่อให้ทำหน้าที่ Plot ส่วนที่มองเห็นเหล่านี้เมื่อ Plot เสร็จแล้วก็จะกลับมาดูที่ตัวแปร Link อีกว่ามีวัตถุอื่นบังมันอีกหรือไม่ถ้ามีก็จะทำการคำนาณหาส่วนที่ถูกบังโดย Procedure Unplot_of_line_segment และนำไปลบออกจาก Vertual Screen โดยใช้ Procedure Unplot_of_line_segment

หลังจากนั้นตัว Loop จะเพิ่มค่าขึ้นและทำเช่นนี้อีกจนกว่าจะหมด Loop เมื่อหมด Loop แล้วนั่นคือขณะนี้ใน Vertual screen จะมีภาพของวัตถุที่ถูกบังทุกชิ้นและภาพที่ถูกบังนี้ได้มีการตัดส่วนที่ถูกบังออกไปจนหมดแล้วและต่อมาก็จะทำการ Loop เพื่อหาวัตถุที่ไม่มีอะไรมาบังเลย Plot อีก โดย Procedure Display และในขณะนี้ที่หน้าจุดยังไม่เห็นภาพเพราะว่าภาพเหล่านั้นอยู่ใน Vertual Screen เราจะต้องทำการสลับข้อมูลกันระหว่าง Vertual Screen และ Active screen เสียก่อนโดยใช้คำสั่ง Swapscreen (เป็นคำสั่งเทอร์โบทูลบล๊อก) ที่มีอยู่ใน Main Program เมื่อแสดงภาพที่ Plot เรียบร้อยแล้วต่อจากนั้นจะตรวจสอบสภานะควบคุมโดยใช้ตัวแปร StatusKBD ถ้าหากเป็น "J" แสดงว่าเป็นการใช้ Joystic ควบคุมถ้าหากเป็น "K" ก็แสดงว่าเป็นการควบคุมโดย Keyboard ถ้าเป็น "J" จะอ่านค่าการโยก Joystic และส่งค่าที่ได้เข้ามาเก็บไว้ในตัวแปรของ Joystic 2 ตัวคือ Joystatus และ Joycommand

เพื่อนำค่า Joystatus และ Joycommand มาทำการหมุนหรือย้ายภาพต่อไปแต่ถ้าหากว่า StatusKBD = K ก็จะทำการอ่าน Keyboard เพื่อควบคุมการหมุนหรือย้ายภาพต่อไป ส่วนสภานะและความหมายที่ใช้ควบคุมขบานการทั้ง Keyboard และJoystic จะเป็นดังนี้

Joystic

ความหมาย	Joycommand	Joystatus
อ้า Clipper	Ø1	18
หุบ Clipper	Ø2	18
หมุน Clipper ขวา	Ø3	17
หมุน Clipper ซ้าย	Ø4	17
ยก Clipper ขึ้น	Ø5	16
กด Clipper ลง	Ø6	16
ยก Arm 1 ขึ้น	Ø7	15
กด Arm 1 ลง	Ø8	15
ยก Arm 2 ขึ้น	Ø9	14
กด Arm 2 ลง	10	14
หมุน Base ขวา	11	13
หมุน Base ซ้าย	12	13

Keyboard

ความหมาย	ค่าในตัวแปร ch
อ้า Clipper	G
หุบ Clipper	F
หมุน Clipper ขวา	N

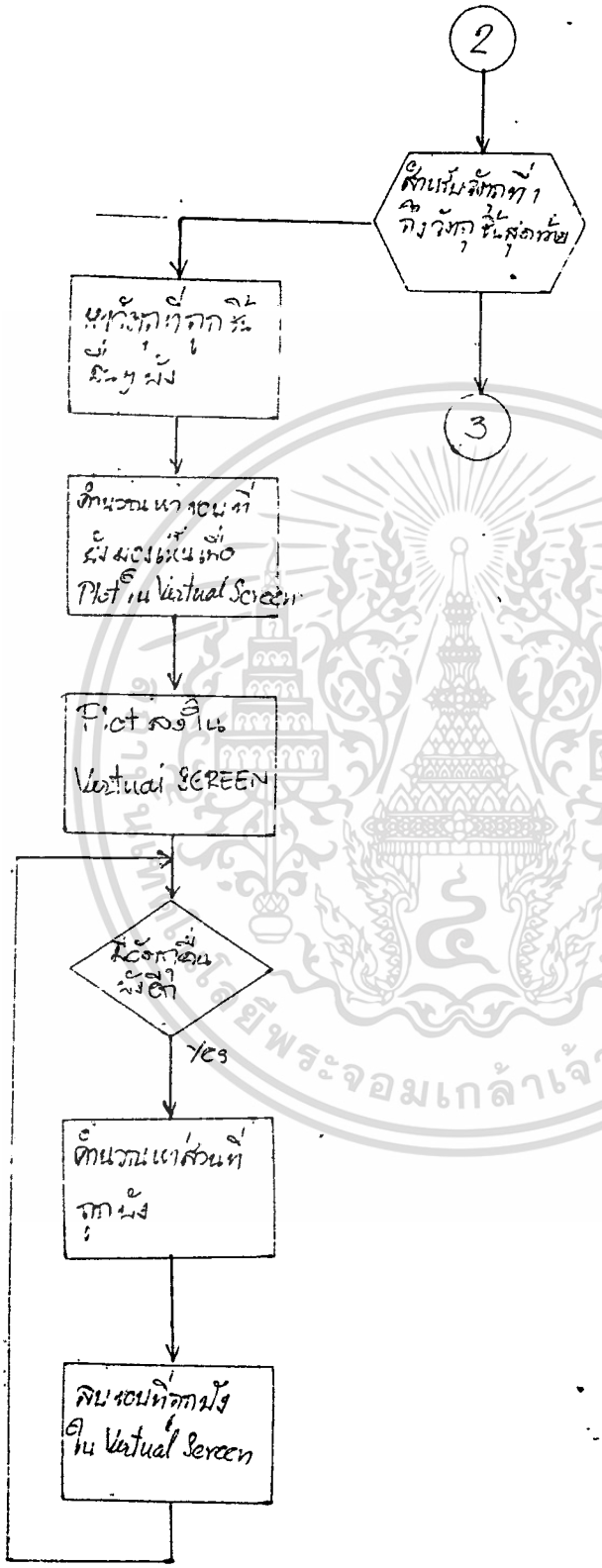
Keyboard(ต่อ)

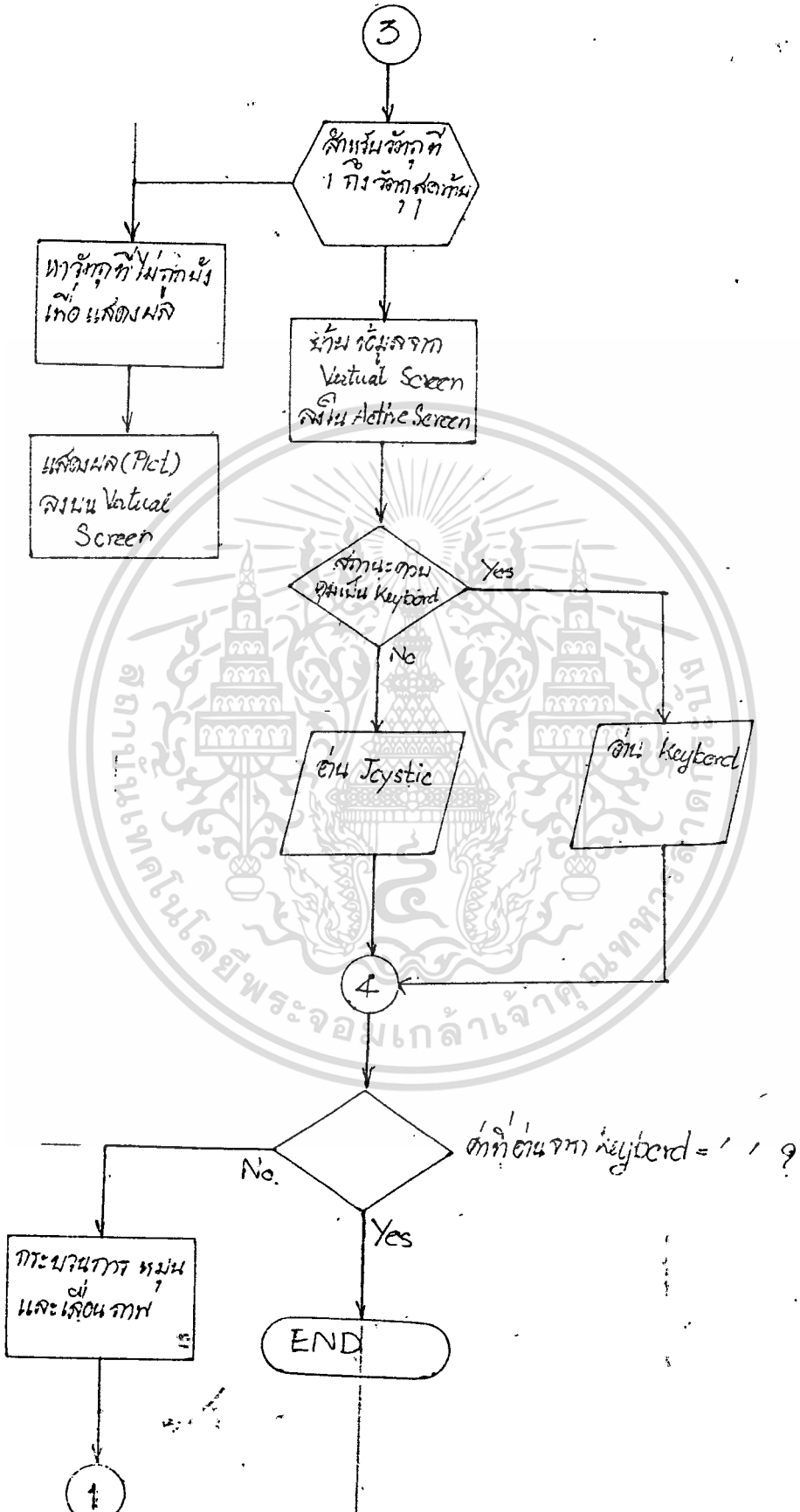
ความหมาย	ค่าในตัวแปร ch
หมุน Clipper ซ้าย	M
ยก Clipper ขึ้น	D
กด Clipper ลง	C
ยก Arm 1 ขึ้น	S
กด Arm 1 ลง	X
ยก Arm 2 ขึ้น	A
กด Arm 2 ลง	Z
หมุน Base ขวา	B
หมุน Base ซ้าย	V

เมื่ออ่านค่าสภานะต่าง ๆ เรียบร้อยแล้วก็จะมีการนำสภานะเหล่านั้นไปทำการหมุนภาพ, เลื่อนภาพ, ย้ายภาพ โดยขึ้นอยู่กับสภาวะที่ได้รับในขณะนั้นแล้วก็จะกลับไปที่ Node1 อีกครั้งหนึ่งและก็จะดำเนินตามกรรมวิธีที่กล่าวไว้แล้วตั้งแต่ต้น

หมายเหตุ ในโปรแกรมนี้ถ้าใช้ Joystic จะไม่สามารถหลุดออกจากโปรแกรมได้ แต่ถ้าหากอยู่ใน Mode Keyboard จะสามารถออกจากโปรแกรมได้ซึ่งการอ่าน Keyboard ถ้าหากอ่านได้รหัสว่าง(' ')จะหมายถึงการจบโปรแกรม

ส่วนผังงานและตัวโปรแกรมทั้งหมดจะอยู่ในหน้าถัดไป





```
{*************************************************************
 *                                                           *
 *            Program Robot Arm Simulate                     *
 *            --------------------------                     *
 *      Progarmmer  : Mr.Witthayakorn Chinyo                 *
 *                  : Mr.Manop       Saekhoo                 *
 *                                                           *
 *************************************************************}

Program ROBOT_ARM_SIMULATE;

{$I e:typedef.sys}
{$I e:graphix.sys}
{$I e:kernel.sys}
{$I e:windows.sys}

Const     Object = 7;
          MaxSur = 12;
          Maxdot = 26;
          MaxdotJump = 11;
          MaxEdge    = 40;

Type
          Nver         = Array[1..Object] of Integer;
          Vertex       = Array[1..Object,1..Maxdot,1..3] of Real;
          Surface      = Array[1..Object,1..MaxSur,1..MaxdotJump] of Integer;
          Nsur         = Array[1..Object,1..MaxSur] of Integer;
          Normal       = Array[1..Object,1..MaxSur,1..3] of Integer;
          Edge         = Array[1..Object,1..MaxEdge,1..4] of Integer;
          Minmax       = Array[1..Object] of Real;
          EndPoint1    = Array[1..300] of Integer;
          Matrix1X4    = Array[1..4] of Real;
          Matrix4X4    = Array[1..4,1..4] of Real;


          Statuss      = Record
                         anglex:real;
                         angley:real;
                         anglez:real;
                         Tz    :real;
                         end;{record}

          Check        = (INF);


Var
                    {** Variable Array **}
          NV,NS,NVE,Nvex    : Nver          ; { Integer }
          V,EC,SC,SCx       : Vertex        ; {   Real  }
          SF                : Surface       ; { Integer }
          NPS,VSF           : Nsur          ; { Integer }
          N                 : Normal        ; { Integer }
          E,Ex              : Edge          ; { Integer }
          XMX,XMN           : Minmax        ; {   Real  }
          YMX,YMN           : Minmax        ; {   Real  }
          ZMX,ZMN           : Minmax        ; {   Real  }
          ix,iy             : array[1..100] of real; {   Real  }
          endpoint          : EndPoint1     ; { Integer }
          endpointx         : array[1..7] of endpoint1      ; { Integer }
          Status1           : Statuss;
          Status2           : Statuss;
          Status3           : Statuss;
```

```
        Status4          : Statuss;
        Status5          : Statuss;
        Status6          : Statuss;
        Status7          : Statuss;
        index            : integer;
        JoyStatus        : integer;
        JoyCommand       : integer;
        StatusKBD        : Char;
        StringStatus     : array[1..8] of String[8];
        String1Status    : array[1..8] of String[8];

        Link             : array[1..7,Check,1..20] of integer;

                {** File(s) Variable **}
        VFile            : File OF Real;
        SF_NPS_File      : File OF Integer;


                  {Math Variable}

        MT               : Matrix4X4; {Matrix for Translation}
        MRx              : Matrix4X4; {Matrix for Rotation x }
        MRy              : Matrix4X4; {Matrix for Rotation y }
        MRz              : Matrix4X4; {Matrix for Rotation z }
        Msc              : Matrix4X4; {Matrix for Scaling    }
        RT               : Matrix4X4; {Matrix for Rotation and Translation}


                {variable in main program}

                i,j,k,o      : integer;
                overlap,stop : boolean;
                kkk          : integer; {kkk is status containment}
                obj,referance: integer;
                ch           : char;
                kk           : array[1..7,1..7] of integer;
                d,vd,phi,tx,ty,scf,s,theta,
                sn1,sn2,cn1,cn2,k1,k2,w : real;

Procedure SetMatrix;
begin
  MT[1,1]:=1; MT[1,2]:=0; MT[1,3]:=0; MT[1,4]:=0;
  MT[2,1]:=0; MT[2,2]:=1; MT[2,3]:=0; MT[2,4]:=0;
  MT[3,1]:=0; MT[3,2]:=0; MT[3,3]:=1; MT[3,4]:=0;
  {variable}  {variable}  {variable}  MT[4,4]:=1;

  MRx[1,1]:=1; MRx[1,2]:=0; MRx[1,3]:=0; MRx[1,4]:=0;
  MRx[2,1]:=0;  {variable}   {variable}  MRx[2,4]:=0;
  MRx[3,1]:=0;  {variable}   {variable}  MRx[3,4]:=0;
  MRx[4,1]:=0; MRx[4,2]:=0; MRx[4,3]:=0; MRx[4,4]:=1;

   {variable}  MRy[1,2]:=0;  {variable}  MRy[1,4]:=0;
  MRy[2,1]:=0; MRy[2,2]:=1; MRy[2,3]:=0; MRy[2,4]:=0;
   {variable}  MRy[3,2]:=0;  {variable}  MRy[3,4]:=0;
  MRy[4,1]:=0; MRy[4,2]:=0; MRy[4,3]:=0; MRy[4,4]:=1;

   {variable}   {variable}  MRz[1,3]:=0; MRz[1,4]:=0;
   {variable}   {variable}  MRz[2,3]:=0; MRz[2,4]:=0;
  MRz[3,1]:=0; MRz[3,2]:=0; MRz[3,3]:=1; MRz[3,4]:=0;
```

```
  Msc[2,1]:=0;  {variable}  Msc[2,3]:=0; Msc[2,4]:=0;
  Msc[3,1]:=0; Msc[3,2]:=0;  {variable}  Msc[3,4]:=0;
  Msc[4,1]:=0; Msc[4,2]:=0; Msc[4,3]:=0; Msc[4,4]:=1;
end;


Procedure Math(Var A:Matrix1X4;B:Matrix4X4);
Var i,k : Integer; C:Matrix1X4;
begin
  for i :=  1 to 4 do
    begin
      C[i] := 0;
      for k := 1 to 4 do
        C[i]:=C[i]+A[k]*B[k,i];
    end;
    A:=C;
end;

Procedure Translation(Object1,Object2:Integer;Tx,Ty,Tz:Real);
var again : integer;
    I,J:Integer;
    A  : Matrix1X4;
begin
    MT[4,1]:=Tx; MT[4,2]:=Ty; MT[4,3]:=Tz;
    for again := object1 to object2 do
    begin
    for i:= 1 to NV[again] do
      begin
        for j:= 1 to 3 do
          A[j]:=V[again,i,j];
          A[4]:=1;
          MATH(A,MT);
          for j := 1 to 3 do
          V[again,i,j]:=A[j];
      end;
    end;
end;

Procedure Rotation(Object:Integer;CH:Char;Angle1:real);
Var A:matrix1X4;B:matrix4X4;Angle:real;
    i,j : integer;
 procedure setmrx;
    begin
      MRx[2,2]:=cos(angle); MRx[2,3]:=sin(angle);
      MRx[3,2]:=-MRx[2,3] ; MRx[3,3]:=MRx[2,2]  ;
      B:=MRx;
    end;
 procedure setmry;
    begin
      MRy[1,1]:=cos(angle); MRy[3,1]:=sin(angle);
      MRy[1,3]:=-MRy[3,1] ; MRy[3,3]:=MRy[1,1]  ;
      B:=MRy;
    end;
 procedure setmrz;
    begin
      MRz[1,1]:=cos(angle); MRz[1,2]:=sin(angle);
      MRz[2,1]:=-MRz[1,2] ; MRz[2,2]:=MRz[1,1]  ;
      B:=MRz;
    end;
begin
```

```
    'x' : Setmrx;
    'y' : Setmry;
    'z' : Setmrz;
    End;{Case}
    for i:= 1 to NV[Object] do
      begin
        for j := 1 to 3 do
          A[j]:=V[Qbject,i,j];
          A[4]:=1;
          MATH(A,B);
          for j := 1 to 3 do
            V[Object,i,j]:=A[j];
      end;
  end;

Procedure Scaling(Object:Integer;Sx,Sy,Sz:real);
Var A:Matrix1X4;i,j:integer;
begin
  Msc[1,1]:=Sx; Msc[2,2]:=Sy; Msc[3,3]:=Sz;
  for i:= 1 to NV[Object] do
    begin
    for j := 1 to 3 do
      A[j]:=V[Object,i,j];
      A[4]:=1;
      MATH(A,Msc);
      for j := 1 to 3 do
        V[Object,i,j]:=A[j];
    end;
end{Scaling};

Procedure Read_DATA_File;
var i,j,k : integer;
  Begin
    NV[1]:=12;NS[1]:=7;
    NV[2]:=26;NS[2]:=12;
    NV[3]:=26;NS[3]:=12;
    NV[4]:=18;NS[4]:=9;
    NV[5]:=10;NS[5]:=6;
    NV[6]:=7;NS[6]:=5;
    NV[7]:=7;NS[7]:=5;

    Assign(Vfile,'VERTEX.DTA');
    Reset(Vfile);
    WHILE NOT EOF(Vfile) Do
    begin
    for i := 1 to Object do
       for j := 1 to Nv[i] do
         for k := 1 to 3 do
          Read(Vfile,V[i,j,k]);
     end;
    Close(Vfile);
    Assign(SF_NPS_FILE,'SURFACE.DTA');
    Reset(SF_NPS_FILE);
    WHILE NOT EOF(SF_NPS_FILE) DO
    begin
      for i := 1 to Object do
        for j := 1 to NS[i] do
          Read(SF_NPS_FILE,NPS[i,j]);
            for i := 1 to Object do
              for j := 1 to NS[i] do
                for k := 1 to NPS[i,j] do
                  read(SF_NPS_FILE,SF[i,j,k]);
```

```
      end;
      Close(SF_NPS_FILE);
    End;

  Procedure VerTex_Array(object:Integer);
    Procedure Eye_Screen_Coordinate;
    Var Xe,Ye,Ze,X,Y,Z,Xs,Ys,Zs : Real;
        i,j,k : integer;
    Begin {Eye}
      i:=object;
         for j := 1 to Nv[i] do begin
             X :=V[i,j,1]; Y:=V[i,j,2]; Z:=V[i,j,3];
             Xe := -X*Sn1 + Y*Cn1;
             Ye := -X*Cn1*Cn2 - Y*Sn1*Cn2 + Z*Sn2;
             Ze := -X*Sn2*Cn1 - Y*Sn2*Sn1 - Z*Cn2 + D;
             Xs := (Vd/S)*(Xe/Ze);
             Ys := (Vd/S)*(Ye/Ze);
             Zs := W*(1-K1/Ze);
             EC[i,j,1]:=Xe; EC[i,j,2]:=Ye; EC[i,j,3]:=Ze;
             SC[i,j,1]:=Xs; SC[i,j,2]:=Ys; SC[i,j,3]:=Zs;
          end;{Next j}
    End;
  Begin
    Eye_Screen_Coordinate;
  End;{Vertex}

  Procedure Normal_Array(object:integer);
  Var U,W : Array[1..3] of Real;
      i,j,k : integer;
  Begin
     i := object;
        for j := 1 to Ns[i] do begin
            for k := 1 to 3 do begin
                U[k]:=V[i,SF[i,j,2],k] - V[i,SF[i,j,1],k];
                W[k]:=V[i,SF[i,j,3],k] - V[i,SF[i,j,1],k];
            end;{Next k}
            N[i,j,1]:=Trunc(U[2]*W[3] - W[2]*U[3]);
            N[i,j,2]:=Trunc(U[3]*W[1] - W[3]*U[1]);
            N[i,j,3]:=Trunc(U[1]*W[2] - W[1]*U[2]);
        end;{Next j}
  End;{Return}


  Procedure Visibility_Test(object:integer);
  Var Xp,Yp,Zp,Lx,Ly,Lz,T : Real;
      i,j,k ; integer;
  Begin
    Xp:=D*Sn2*Cn1; Yp:=D*Sn2*Sn1; Zp:=D*Cn2;
        i := object;
            for j := 1 to NS[i] do
              begin
                  Lx := Xp - V[i,SF[i,j,1],1];
                  Ly := Yp - V[i,SF[i,j,1],2];
                  Lz := Zp - V[i,SF[i,j,1],3];
                  T  := N[i,j,1]*Lx + N[i,j,2]*Ly + N[i,j,3]*Lz;
                  IF T <=0 Then VSF[i,j]:=0 Else VSF[i,j]:=1;
              end;{Next j}
End;{Return}

Procedure Visible_Edge_Array(object:integer);
Label Exit;
Var M,E1,E2,L : Integer;
```

```
    i,j,k     : integer;
  Begin
      i := object;
      M := 1;
      for j := 1 to Ns[i] do begin
        If VSF[i,j]=0 Then begin end
           Else begin
              E1:=SF[i,j,1];
              for k := 2 to NPS[i,j] do begin
                 E2:=SF[i,j,k];
                 for L := 1 to M do begin
                     IF(E[i,1,1]=E2)AND(E[i,1,2]=E1) Then
                     begin E[i,1,3]:=2;Goto Exit;end
                     Else
                 end; {Next L}
                 E[i,m,1]:=E1;E[i,m,2]:=E2;E[i,m,3]:=1;E[i,m,4]:=J;
                 M:=M+1;
                 Exit:
                 E1:=E2;
              end;{Next k}
              NVE[i]:=M-1;
           end{Else};
     end;{Next J}
  End;{Return}


  Procedure Display(ObjectNumber:Integer);
  Var KJ,F1,F2,LL : Integer;
      a,b,c,d     : real;
  Begin
  LL := ObjectNumber;
  NVEx:=NVE;SCx:=SC;Ex:=E;
      for KJ:=1 to NVE[LL] do begin
          IF E[LL,kj,3]=0 Then begin end {Next KJ}
          Else begin
               F1:=E[LL,KJ,1]; F2:=E[LL,KJ,2];
               A:=SC[LL,F1,1]; B:=SC[LL,F1,2];
               C:=SC[LL,F2,1]; D:=SC[LL,F2,2];
               A:=SCF*A+Tx; B:=Ty-B; C:=SCF*C+Tx; D:=Ty-D;
               drawline(Trunc(A),Trunc(B),Trunc(C),Trunc(D));
          end{Else};
      end;{Next KJ}
  End;{Return}

  Procedure minimax_test(obj1,obj2:integer;var overlap:boolean);
  var i,j,k : integer;
      a,b   : real;
  begin
    i := obj1;
    xmx[i]:=-9E+9 ; xmn[i]:=9E+9;
    ymx[i]:=xmx[i]; ymn[i]:=xmn[i];
    for j := 1 to nve[i] do
      begin
        if e[i,j,3] <> 1 then begin end
        else
          begin
            k:=e[i,j,1];
            a:=sc[i,k,1]; b:=sc[i,k,2];
            if a > xmx[i] then xmx[i]:=a;
            if a < xmn[i] then xmn[i]:=a;
            if b > ymx[i] then ymx[i]:=b;
            if b < ymn[i] then ymn[i]:=b;
```

```
        end;
      end;
    i := obj2;
    xmx[i]:=-9E+9 ; xmn[i]:=9E+9;
    ymx[i]:=xmx[i]; ymn[i]:=xmn[i];
    for j := 1 to nve[i] do
      begin
        if e[i,j,3] <> 1 then begin end
        else
          begin
            k:=e[i,j,1];
            a:=sc[i,k,1]; b:=sc[i,k,2];
            if a > xmx[i] then xmx[i]:=a;
            if a < xmn[i] then xmn[i]:=a;
            if b > ymx[i] then ymx[i]:=b;
            if b < ymn[i] then ymn[i]:=b;
          end;
      end;
    if(xmx[obj1]<xmn[obj2])or(xmx[obj2]<xmn[obj1])
    or(ymx[obj1]<ymn[obj2])or(ymx[obj2]<ymn[obj1])
    then overlap := false
    else overlap := true;
end;

Function sgn(X:real):integer;
begin
  if x=0 then sgn:=0
  else
    if x>0 then sgn:=1 else sgn:=-1;
end;

Procedure intersection_test(i,l,a,b,c,d:integer;
var ax,ay,bx,by,cx,cy,dx,dy,xi,yi:real;var ncut:boolean);
var r1,r2,r3,r4,a1,a2,b1,b2,c1,c2,d1,d2:real;
    s1,s2,s3,s4 : integer;
    m1,m2,m3    : real;
    z0,z1,z2,z3 : real;
begin
  ax:=sc[i,a,1]; ay:=sc[i,a,2]; bx:=sc[i,b,1]; by:=sc[i,b,2];
  cx:=sc[l,c,1]; cy:=sc[l,c,2]; dx:=sc[l,d,1]; dy:=sc[l,d,2];
  r1:=ax-bx;     r2:=ay-by;     r3:=cx-dx;     r4:=cy-dy;
  s1:=sgn(r1);   s2:=sgn(r2);   s3:=sgn(r3);   s4:=sgn(r4);
  a1:=ax; b1:=bx; a2:=ay; b2:=by; c1:=cx; d1:=dx; c2:=cy; d2:=dy;
  if s1>0 then begin a1:=bx; b1:=ax; end;
  if s2>0 then begin a2:=by; b2:=ay; end;
  if s3>0 then begin c1:=dx; d1:=cx; end;
  if s4>0 then begin c2:=dy; d2:=cy; end;
  {minmax for edge overlapping}
  if(b1<c1)or(d1<a1)or(b2<c2)or(d2<a2)then ncut:=false
  else
    begin
      m1:=r2/r1; m2:=r4/r3; m3:=m2-m1;
      if(m3=0)or(abs(m3)<=0.0001)then ncut:=false
      else
        begin
          {compute intersection point}
          z0:=ay-cy; z1:=m1*ax-m2*cx; z2:=(ay-m1*ax)*m2; z3:=(cy-m2*cx)*m1;
          xi:=(z0-z1)/m3; yi:=(z2-z3)/m3;
          if(xi<a1)or(xi>b1)or(xi<c1)or(xi>d1)or
          (yi<a2)or(yi>b2)or(yi<c2)or(yi>d2) then ncut:=false
          else ncut:=true;
        end;
```

```
    end;
  end;

  Procedure compute_zs(ii,ij,q:integer;a1,a2,a3,b1,b2,b3,xi,yi:real;var dt:real);
  label exit;
  var jj,kk : integer;
      c1,c2,c3,aa,bb,cc,dd,ee,ff,xx,yy,zz,t1,t2,depth:real;
  begin
    for jj := 1 to nps[ii,q] do
      begin
        if sf[ii,q,jj]<>e[ii,ij,2] then begin end
        else
        begin
        kk:=sf[ii,q,jj+1];
        goto exit;
        end;
      end;{next jj}
  exit: c1:=sc[ii,kk,1]; c2:=sc[ii,kk,2]; c3:=sc[ii,kk,3];
        aa:=b1-a1; bb:=b2-a2; cc:=b3-a3; dd:=c1-a1; ee:=c2-a2; ff:=c3-a3;
        xx:=bb*ff-cc*ee; yy:=cc*dd-aa*ff; zz:=aa*ee-bb*dd;
        t1:=xx*a1 + yy*a2 + zz*a3;
        t2:=xx*xi + yy*yi;
        depth:=(t1-t2)/zz;
        dt:=depth;
  end;

  Procedure depth_test(i,l,j,k,a,b,c,d:integer;
  ax,ay,bx,by,cx,cy,dx,dy,xi,yi:real;var dt1,dt2:real);
  var ii,ij,q : integer;
      a1,a2,a3,b1,b2,b3 : real;
  begin
    ii:=i; ij:=j;
    a1:=ax; a2:=ay; a3:=sc[i,a,3]; q:=e[i,j,4];
    b1:=bx; b2:=by; b3:=sc[i,b,3];
    compute_zs(ii,ij,q,a1,a2,a3,b1,b2,b3,xi,yi,dt1);
    ii:=l; ij:=k;
    a1:=cx; a2:=cy; a3:=sc[l,c,3]; q:=e[l,k,4];
    b1:=dx; b2:=dy; b3:=sc[l,d,3];
    compute_zs(ii,ij,q,a1,a2,a3,b1,b2,b3,xi,yi,dt2);
  end;

  Procedure containment_test(obj1,obj2:integer);
  var i,j : integer;
  begin
    i:=obj1;
    zmx[i]:=0; zmn[i]:=1;
    for j := 1 to nv[i] do
      begin
        if sc[i,j,3]>zmx[i] then zmx[i]:=sc[i,j,3];
        if sc[i,j,3]<zmn[i] then zmn[i]:=sc[i,j,3];
      end;
    i:=obj2;
    zmx[i]:=0; zmn[i]:=1;
    for j := 1 to nv[i] do
      begin
        if sc[i,j,3]>zmx[i] then zmx[i]:=sc[i,j,3];
        if sc[i,j,3]<zmn[i] then zmn[i]:=sc[i,j,3];
      end;
  end;
```

```
var i,j,k,l : integer;
    ncut    : boolean;
    dt1,dt2 : real;
    a,b,c,d : integer;
    ax,ay,bx,by,cx,cy,dx,dy,xi,yi : real;
    ii,ll  : integer;
begin
  i := obj1; l:=obj2;
  for j := 1 to nve[i] do                 {Number edge to see of object[i] }
    begin
      a:=e[i,j,1]; b:=e[i,j,2];
      for k := 1 to nve[l] do             {Number edge to see of object[l] }
       begin
         if e[l,k,3] <> 1 then begin end {e[l,k,3] = 1;edge of one surface}
         else                            {e[l,k,3] = 2;edge of two surface}
           begin
             c:=e[l,k,1]; d:=e[l,k,2];
             intersection_test(i,l,a,b,c,d,ax,ay,bx,by,cx,cy,dx,dy,xi,yi,ncut);
             if ncut = false then begin end
             else
               begin
                 depth_test(i,l,j,k,a,b,c,d,ax,ay,bx,by,cx,cy,dx,dy,xi,yi,dt1,dt2);
                 if dt1>dt2 then
                 begin
                   kkk := 0; ii:=obj1; ll:=obj2; {display(ll);}
                   obj:=ii;  referance:=ll;  goto exit;
                 end
                 else
                 begin
                   kkk := 0; ii:=obj2; ll:=obj1; {display(ll);}
                   obj:=ii;referance:=ll;   goto exit;
                 end;
               end;
           end;
      end;
  end;
  containment_test(obj1,obj2);
  if(zmn[obj1]<zmx[obj2])and(ymx[obj2]>ymx[obj1])then
  begin
    ll:=obj2; stop:=true; referance:=ll; obj:= -1;
    end
    else
    begin
      if(zmx[obj2]<zmn[obj1])and(ymx[obj1]>ymx[obj2])then
      begin
        ll:=obj1; stop:=true; referance:=ll; obj := -1;
      end
        else
        begin
          if(ymx[obj2]>ymx[obj1])then
          begin
              kkk:=1 ;ii:=obj1; ll:=obj2; {display(ll);}
              obj:=ii; referance:=ll;   goto exit;
           end
           else
           begin
             if(ymx[obj1]>ymx[obj2])then
             begin
               kkk:=1; ii:=obj2; ll:=obj1; {display(ll);}
               obj:=ii; referance:=ll;  goto exit;
        end;
      end;
```

```
      end;
    end;
exit:
end;

Procedure visible_endpoint(Obj,referance:integer);
label exit;
var j,k,ii,ll : integer;
    a,c,d     : integer;
    ax,ay,cx,cy,dx,dy : real;
    v1,v2,u1,u2,test  : real;
begin
  ii:=obj; ll:=referance;
  for j := 1 to nve[ii] do  {nve is number of edge to see}
  begin
    a:=e[ii,j,1];
    for k := 1 to nve[ll] do
    begin
      if e[ll,k,3]<>1 then begin end
      else
      begin
        c:=e[ll,k,1]; d:=e[ll,k,2];
        ax:=sc[ii,a,1]; ay:=sc[ii,a,2];
        cx:=sc[ll,c,1]; cy:=sc[ll,c,2];
        dx:=sc[ll,d,1]; dy:=sc[ll,d,2];
        v1:=dx-cx; v2:=dy-cy;
        u1:=ax-cx; u2:=ay-cy;
        test:=u2*v1-u1*v2;
        if test <= 0 then
        begin
          endpoint[j]:=1;
          goto exit;
        end;
      end;
    end;{next k}
    endpoint[j]:=-1;
exit:
  end;{next j}
end;

Procedure plot_of_visible_edge_segment(vi,tcut,ax,ay,bx,by:integer);
begin
  if(tcut=0)and(vi=1 )then drawline(ax,ay,bx,by)
  else
    if(tcut=0)and(vi=-1)then begin end
    else
      if(tcut=1)and(vi=1 )then drawline(ax,ay,trunc(ix[1]),trunc(iy[1]))
      else
        if(tcut=1)and(vi=-1)then drawline(trunc(ix[1]),trunc(iy[1]),bx,by)
        else
          if (tcut=2) then
          begin
            if((ax<bx)and(trunc(ix[1])<trunc(ix[2]))and(ax>bx)and(trunc(ix[2])<trunc(ix[1])))then
              begin
                drawline(ax,ay,trunc(ix[1]),trunc(iy[1]));
                drawline(trunc(ix[2]),trunc(iy[2]),bx,by);
              end
              else
              begin
                drawline(ax,ay,trunc(ix[2]),trunc(iy[2]));
                drawline(trunc(ix[1]),trunc(iy[1]),bx,by);
              end;
```

```
    else drawtextw(100,500,1,' WIDE');   {'Up'}
    invertwindow;

    temz  := status5.anglez;
    temx  := status5.anglex;
    temy  := status5.angley;

    if temz <> 0 then
    for o := 5 to 7 do
    rotation(o,'z',-temz);        {Step 1}

    if temx <> 0 then
    for o := 5 to 7 do
    rotation(o,'x',-temx);        {Step 2}

    if temy <> 0 then
    for o := 5 to 7 do
    rotation(o,'y',-temy);        {Step 3}

    teml[1]:=v[5,9,1];teml[2]:=v[5,9,2];teml[3]:=v[5,9,3];
    translation(5,7,-teml[1],-teml[2],-teml[3]);
    translation(6,6,0,0,step);
    translation(7,7,0,0,-step);

    status6.Tz:=status6.Tz+step;
    status7.Tz:=status7.Tz-step;

    translation(5,7,teml[1],teml[2],teml[3]);
    if temy <> 0 then
    for o := 5 to 7 do
    rotation(o,'y',temy);          {Inv Step 3}

    if temx <> 0 then
    for o := 5 to 7 do
    rotation(o,'x',temx);          {Inv Step 2}

    if temz <> 0 then
    for o := 5 to 7 do
    rotation(o,'z',temz);          {Inv Step 1}
    Selectwindow(6);
    invertwindow;
    Selectwindow(10);
    Setbackground(0);
    drawborder;
end;

Procedure Clipper(angle:real);
var temz,temx  : real;
    teml : array[1..3] of real;
begin
    Selectwindow(6);       {Clipper}
    invertwindow;
    Selectwindow(7);
    invertwindow;
    if angle < 0 then
    begin
    Selectwindow(9);
    invertwindow;
    end
    else
```

```
    end;
    temz  := status5.anglez;
    temx  := status5.anglex;

    if temz <> 0 then
    for o := 5 to 7 do
    rotation(o,'z',-temz);          {Step 1}
    if temx <> 0 then
    for o := 5 to 7 do
    rotation(o,'x',-temx);          {Step 2}

    tem1[1]:=v[5,9,1];tem1[2]:=v[5,9,2];tem1[3]:=v[5,9,3];
    translation(5,7,-tem1[1],-tem1[2],-tem1[3]);
    for o := 5 to 7 do
    rotation(o,'y',angle);

    status5.angley:=status5.angley+angle;
    status6.angley:=status6.angley+angle;
    status7.angley:=status7.angley+angle;

    translation(5,7,tem1[1],tem1[2],tem1[3]);
    if temx <> 0 then
    for o := 5 to 7 do
    rotation(o,'x',temx);
    if temz <> 0 then
    for o := 5 to 7 do
    rotation(o,'z',temz);
    Selectwindow(6);
    invertwindow;
    Selectwindow(7);
    invertwindow;
    if angle < 0 then
    begin
    Selectwindow(9);
    invertwindow;
    end
    else
    begin
    Selectwindow(8);
    invertwindow;
    end;
end;

Procedure H2(angle:real);
var tem  : real;
    tem1 : array[1..3] of real;
  begin
    SelectWindow(6);
    invertwindow;
    selectwindow(10);
    setbackground(0);
    drawborder;
    if angle < 0 then
    drawtextw(100,500,1,' DOWN')       {'Down'}
    else drawtextw(100,500,1,' UP');   {'Up'}
    invertwindow;
    tem  := status4.anglez;
    if tem = 0 then begin end
    else
    for o := 4 to 7 do
    rotation(o,'z',-tem);                        {step 1}
    tem1[1]:= v[4,17,1]; tem1[2]:= v[4,17,2]; tem1[3]:= v[4,17,3];
```

```
    translation(4,7,-tem1[1],-tem1[2],-tem1[3]);     {step 2}
    for o := 4 to 7 do
    rotation(o,'x',angle);

    status4.anglex:=status4.anglex+angle;
    status5.anglex:=status5.anglex+angle;

    translation(4,7,tem1[1];tem1[2],tem1[3]);       { INV step 2}
    if tem <> 0 then
    for o := 4 to 7 do
    rotation(o,'z',tem);                        { INV step 1}
    Selectwindow(6);
    invertwindow;
    Selectwindow(10);
    SetBackground(0);
    drawborder;
  end;

Procedure Arm1(angle:Real);
var tem  : real;
    tem1 : array[1..,3] of real;
  begin
    SelectWindow(3);
    invertwindow;
    selectwindow(10);
    setbackground(0);
    drawborder;
    if angle < 0 then
    drawtextw(100,500,1,' DOWN')       {'Down'}
    else drawtextw(100,500,1,' UP');   {'Up'}
    invertwindow;
    tem := status3.anglez;
    if tem = 0 then begin end
    else
    for o := 3 to 7 do
    rotation(o,'z',-tem);                    {step 1}
    tem1[1]:= v[3,25,1]; tem1[2]:= v[3,25,2]; tem1[3]:= v[3,25,3];
    translation(3,7,-tem1[1],-tem1[2],-tem1[3]);    {step 2}
    for o := 3 to 7 do
    rotation(o,'x',angle);

    status3.anglex:=status3.anglex+angle;
    status4.anglex:=status4.anglex+angle;
    status5.anglex:=status5.anglex+angle;

    translation(3,7,tem1[1],tem1[2],tem1[3]);       { INV step 2}
    if tem <> 0 then
    for o := 3 to 7 do
    rotation(o,'z',tem);                        { INV step 1}
    Selectwindow(3);
    invertwindow;
    Selectwindow(10);
    SetBackground(0);
    drawborder;
  end;

Procedure Arm2(angle:Real);
var tem  : real;
    tem1 : array[1..3] of real;
 begin
    SelectWindow(4);
    invertwindow;
```

```
    selectwindow(10);
    setbackground(0);
    drawborder;
    if angle < 0 then
    drawtextw(100,500,1,' DOWN')       ('Down')
    else drawtextw(100,500,1,' UP');   {'Up'}
    invertwindow;
    tem1[1]:= v[2,25,1]; tem1[2]:= v[2,25,2]; tem1[3]:= v[2,25,3];
    tem := status1.anglez;
    if tem = 0 then begin end
    else
    for o := 2 to 7 do
    rotation(o,'z',-tem);
    translation(2,7,0,-tem1[2],-tem1[3]);
    for o := 2 to 7 do
    rotation(o,'x',angle);

    status2.anglex:=status2.anglex+angle;
    status3.anglex:=status3.anglex+angle;
    status4.anglex:=status4.anglex+angle;
    status5.anglex:=status5.anglex+angle;

    translation(2,7,0,tem1[2],tem1[3]);
    if tem <> 0 then
    begin
      for o := 2 to 7 do
      rotation(o,'z',tem);
    end;
    Selectwindow(4);
    invertwindow;
    Selectwindow(10);
    SetBackground(0);
    drawborder;
  end;

Procedure arm3(angle:real);
  begin
    SelectWindow(5);
    invertwindow;
    selectwindow(7);
    invertwindow;
    if angle < 0 then
    begin
    Selectwindow(8);
    invertwindow;
    end
    else
    begin
    Selectwindow(9);
    invertwindow;
    end;
    for o:= 1 to 7 do
    rotation(o,'z',angle);
    status1.anglez := status1.anglez+angle;
    status2.anglez := status2.anglez+angle;
    status3.anglez := status3.anglez+angle;
    status4.anglez := status4.anglez+angle;
    status5.anglez := status5.anglez+angle;
    status6.anglez := status6.anglez+angle;
    status7.anglez := status7.anglez+angle;
    Selectwindow(5);
    invertwindow;
```

```
    Selectwindow(7);
    invertwindow;
    if angle < 0 then
    begin
    Selectwindow(8);
    invertwindow;
    end
    else
    begin
    Selectwindow(9);
    invertwindow;
    end;
  end;

Procedure initObj(object:Integer);
  begin
    vertex_array(object);
    normal_array(object);
    visibility_test(object);
    visible_edge_array(object);
  end;

Procedure SetStatus;
  begin
    with status1 do
    begin
      anglex := 0;
      angley := 0;
      anglez := 0;
    end;
    with status2 do
    begin
      anglex := 0;
      angley := 0;
      anglez := 0;
    end;
    with status3 do
    begin
      anglex := 0;
      angley := 0;
      anglez := 0;
    end;
    with status4 do
    begin
      anglex := 0;
      angley := 0;
      anglez := 0;
    end;
    with status5 do
    begin
      anglex := 0;
      angley := 0;
      anglez := 0;
    end;
    with status6 do
    begin
      anglex := 0;
      angley := 0;
      anglez := 0;
      Iz     := 0;
    end;
    with status7 do
```

```
      begin
        anglex := 0;
        angley := 0;
        anglez := 0;
        Tz     := 0;
      end;
    end;


  Procedure CheckLink;
  var i,j,k,l,m : integer;
      F,B       : integer; {F = Front, B = Back}
      stack     : array[1..7,Check] of integer;

    begin
      for i := 1 to object do
        begin
        Stack[i,inf]:=1;
        for j := 1 to 20 do
          begin
            Link[i,INF,j] := 0;
          end;
        end;

      for i := 1 to object-1 do
        for j := i+1 to object do
          begin
            overlap := false;
            minimax_test(i,j,overlap);
            if overlap then
               begin
                 stop := False; F:=0; B:=0;
                 determine_object_priority(i,j,stop);
                 F:= referance; B:= obj;
                 if stop <> true then
                 begin
                 Link[B,inF,stack[B,inF]] := F;
                 stack[B,inf] := stack[B,inf]+1;
                 kk[B,F] := kkk;
                 end
                 else
                 begin
                 Link[B,inf,stack[B,inf]] := B;
                 kk[B,F] := kkk;
                 end;
               end;
        end;
    end;

Procedure Define_Viewing_Parameters;
begin
  D:=500;Vd:=550;theta:=0.5;Phi:=90.5;
  Tx:=320; Ty:=180; SCF:=2.4; S:=1;
  Theta := Theta * Pi/180;
  Phi   := Phi * Pi/180;
  SN1   := SIN(Theta);
  SN2   := SIN(Phi);
  CN1   := COS(Theta);
  CN2   := COS(Phi);
  K1    := 7;                {** K1<K2 **}
  K2    := 10;
  W     := K2/(K2-K1);
```

```
end;

Procedure Multivisible_endpoint(Obj,referance:integer);
label exit;
var j,k,ii,ll : integer;
    a,c,d     : integer;
    ax,ay,cx,cy,dx,dy : real;
    v1,v2,u1,u2,test  : real;
begin
  ii:=obj; ll:=referance;
  for j := 1 to nve[ii] do  {nve is number of edge to see}
  begin
    a:=e[ii,j,1];
    for k := 1 to nve[ll] do
    begin
      if e[ll,k,3]<>1 then begin end
      else
      begin
        c:=e[ll,k,1]; d:=e[ll,k,2];
        ax:=sc[ii,a,1]; ay:=sc[ii,a,2];
        cx:=sc[ll,c,1]; cy:=sc[ll,c,2];
        dx:=sc[ll,d,1]; dy:=sc[ll,d,2];
        v1:=dx-cx; v2:=dy-cy;
        u1:=ax-cx; u2:=ay-cy;
        test:=u2*v1-u1*v2;
        if test <= 0 then
        begin
          endpoint[j]:=1;
          goto exit;
        end;
      end;
    end;{next k}
    endpoint[j]:=-1;
exit:
  end;{next j}
end;

Procedure Unplot_of_visible_edge_segment(vi,tcut,ax,ay,bx,by:integer);
begin
  if(tcut=0)and(vi=1 )then begin end
  else
    if(tcut=0)and(vi=-1)then begin
                               drawline(ax,ay,bx,by);
                               drawline(bx,by,ax,ay);
                             end
   else
     if(tcut=1)and(vi=1)then begin
                               drawline(trunc(ix[1]),trunc(iy[1]),bx,by);
                               drawline(bx,by,trunc(ix[1]),trunc(iy[1]));
                             end
      else
        if(tcut=1)and(vi=-1)then begin
                                   drawline(ax,ay,trunc(ix[1]),trunc(iy[1]));
                                   drawline(trunc(ix[1]),trunc(iy[1]),ax,ay);
                                 end
          else
            if (tcut=2) then
            begin
              if((ax<bx)and(trunc(ix[1])<trunc(ix[2]))and(ax>bx)and(trunc(ix[2])<trunc(ix[1])))then
              begin
                drawline(trunc(ix[1]),trunc(iy[1]),trunc(ix[2]),trunc(iy[2]));
                drawline(trunc(ix[2]),trunc(iy[2]),trunc(ix[1]),trunc(iy[1]));
```

```
                end
                else
                begin
                  drawline(trunc(ix[2]),trunc(iy[2]),trunc(ix[1]),trunc(iy[1]));
                  drawline(trunc(ix[1]),trunc(iy[1]),trunc(ix[2]),trunc(iy[2]));
                end;
            end {tcut=2}
            else
            if (tcut>2) then
            begin
              if(ax<bx)and(trunc(ix[1])<trunc(ix[tcut]))and(ax>bx)and(trunc(ix[tcut])<trunc(ix[1]))then
                begin
                  drawline(trunc(ix[1]),trunc(iy[1]),trunc(ix[tcut]),trunc(iy[tcut]));
                  drawline(trunc(ix[tcut]),trunc(iy[tcut]),trunc(ix[1]),trunc(iy[1]));
                end
                else
                begin
                  drawline(trunc(ix[tcut]),trunc(iy[tcut]),trunc(ix[1]),trunc(iy[1]));
                  drawline(trunc(ix[1]),trunc(iy[1]),trunc(ix[tcut]),trunc(iy[tcut]));
                end;
            end;{tcut>2}
end;{of Plot}

Procedure UnPlot_of_line_segment(obj,referance:integer);
label Exit1,Exit2,Exit3;
var ncut    : boolean;
    i,l,j,k : integer;
    vi,tcut : integer;
    a,b,c,d : integer;
    ax,ay,bx,by,cx,cy,dx,dy,xi,yi : real;
begin
  SetcolorBlack;
  i:=obj; l:=referance;
  for j := 1 to nve[obj] do
  begin
    vi:=endpoint[j]; ncut:=false; tcut:=0;
    a:=e[obj,j,1]; b:=e[obj,j,2];
    for k := 1 to nve[referance] do
    begin
      if kk[i,referance]=1 then goto Exit1;
      if e[referance,k,3]<>1 then goto Exit3 else goto Exit2;
      Exit1:
      if e[referance,k,3]<>2 then goto Exit3 else goto Exit2;
      Exit2:
      c:=e[referance,k,1]; d:=e[referance,k,2];
      intersection_test(i,l,a,b,c,d,ax,ay,bx,by,cx,cy,dx,dy,xi,yi,ncut);
      if ncut=false then goto Exit3;
      tcut:=tcut+1;
      ix[tcut]:=scf*xi+tx; iy[tcut]:=ty-yi;
     Exit3:
   end;{next k}
   ax:=scf*ax+tx; ay:=ty-ay;
   bx:=scf*bx+tx; by:=ty-by;
   Unplot_of_visible_edge_segment(vi,tcut,trunc(ax),trunc(ay),
   trunc(bx),trunc(by));
  end;{next j}
  SetcolorWhite;
end;

Procedure mul(var status,command :integer); external 'Joy.com';

Procedure JoyStic;
```

```
begin
  Mul(JoyStatus,JoyCommand);
  if JoyStatus=0 then Joystatus:=13;
  case JoyStatus of
  13 :       begin
             if JoyCommand=11 then arm3(30) else
             if JoyCommand=12 then arm3(-30) else
             begin end;
             end;
  14 :       begin
             if JoyCommand=9 then  arm2(30) else
             if JoyCommand=10 then arm2(-30) else
             begin end;
             end;
  15 :       begin
             if JoyCommand=07 then arm1(30) else
             if JoyCommand=08 then arm1(-30) else
             begin end;
             end;
  16 :       begin
             if JoyCommand=05 then H2(30) else
             if JoyCommand=06 then H2(-30) else
             begin end;
             end;
  17 :       begin
             if JoyCommand=03 then Clipper(30) else
             if JoyCommand=04 then Clipper(-30) else
             begin end;
             end;
  18 :       begin
             if JoyCommand=01 then H1(2.5) else
             if JoyCommand=02 then H1(-2.5) else
             begin end;
             end;
  End;{Case}
end;{joystic}

Procedure KeyBord;
begin
  k:=10; j:=40;
  for i := 1 to 8 do
  begin
    DefineWindow(i+2,71,j,77,j+k);
    SelectWindow(i+2);
    SelectWorld(1);
    SetBackground(0);
    drawTextW(100,500,1,String1Status[i]);
    drawborder;
    j:=j+(2*k);
  end;
  read(kbd,ch);
  k:=10; j:=40;
  for i := 1 to 8 do
  begin
    DefineWindow(i+2,71,j,77,j+k);
    SelectWindow(i+2);
    SelectWorld(1);
    SetBackground(0);
    drawTextW(100,500,1,StringStatus[i]);
    drawborder;
    j:=j+(2*k);
  end;
```

```
    ch := Upcase(ch);
    case ch of
    'Z':arm2(-30);
    'A':arm2(30);
    'X':arm1(-30);
    'S':arm1(30);
    'V':arm3(-30);
    'B':arm3(30);
    'D':H2(30);
    'C':H2(-30);
    'N':clipper(-30);
    'M':clipper(30);
    'F':H1(2.5);
    'G':H1(-2.5);
    end;{Case}
  end;

  Procedure InitGraphic_System;
  begin

    InitGraphic;
    SetColorWhite;

    drawText(125,10,3,'Robot Arm Simulate');
    drawborder;

    DefineWindow(1,1,20,68,197);
    DefineHeader(1,'Simulate Here');
    SetHeaderOn;
    DefineWorld(1,0,1000,1000,0);
    DefineWorld(2,0,199,639,0);
    SelectWindow(1);
    SelectWorld(2);
    SetBackground(0);
    drawborder;

    DefineWindow(2,70,20,78,197);
    DefineHeader(2,'Status');
    SetHeaderOn;
    SelectWindow(2);
    SelectWorld(1);
    SetBackground(85);
    drawborder;

    k:=10; j:=40;
    for i := 1 to 8 do
    begin
      DefineWindow(i+2,71,j,77,j+k);
      SelectWindow(i+2);
      SelectWorld(1);
      SetBackground(0);
      drawTextW(100,500,1,StringStatus[i]);
      drawborder;
      j:=j+(2*k);
    end;
  end;

  Procedure SetString;
  begin
    StringStatus[1]:=' Arm 1 ';
    StringStatus[2]:=' Arm 2 ';
    StringStatus[3]:=' Arm 3 ';
```

```
  StringStatus[4]:=' Cliper ';
  StringStatus[5]:=' Rotate ';
  StringStatus[6]:=' Left   ';
  StringStatus[7]:=' Right  ';
  StringStatus[8]:=' UP     ';
  String1Status[1]:=' V B';
  String1Status[2]:=' A Z';
  String1Status[3]:=' S X';
  String1Status[4]:=' D C';
  String1Status[5]:=' M N';
  String1Status[6]:=' F G';
  String1Status[7]:='    ';
  String1Status[8]:='    ';
end;

begin { Main }

     repeat
     ClrScr;
     Write('You want use Joystic or Keybord (J/K) : ');read(kbd,StatusKBD);
     Statuskbd:=Upcase(Statuskbd);
     Until Statuskbd in ['J','K'];

     Setmatrix;
     Setstatus;
     SetString;
     initGraphic_System;
     JoyStatus := 13;
     JoyCommand := 0;
     Read_data_file;
     v[4,1,3]:=1.25; v[4,8,3]:=1.25;
     v[4,2,3]:=1.25; v[4,9,3]:=1.25;
     for o:=1 to 7 do
     scaling(o,10,10,10);
     translation(2,2,2,0,0);
     translation(4,7,2,0,0);
     translation(5,7,0,2,0);
     translation(6,7,0,2,0);
     arm2(90);
     arm1(-90);
     Define_Viewing_Parameters;
     repeat
       SelectScreen(2);
       Selectwindow(10);
       drawtextw(100,500,1,' Wait');
       invertwindow;
       Swapscreen;
       for o := 1 to 7 do
       initobj(o);
       CheckLink;
       SelectScreen(2);
       SelectWindow(1);
       SelectWorld(2);
       SetBackground(0);
       drawborder;
       for i := 1 to 7 do
         begin
           if (Link[i,inf,1]<>0) then
             begin
             visible_endpoint(i,Link[i,inf,1]);
               plot_of_line_segment(i,Link[i,inf,1]);
               if (Link[i,inf,2]<>0)and(Link[i,inf,2]<>-1) then
```

```
          begin
            Multivisible_endpoint(i,Link[i,inf,2]);
            Unplot_of_line_segment(i,Link[i,inf,2]);
          end;
          if (Link[i,inf,3]<>0)and(Link[i,inf,3]<>-1) then
          begin
            Multivisible_endpoint(i,Link[i,inf,3]);
            Unplot_of_line_segment(i,Link[i,inf,3]);
          end;
          if (Link[i,inf,4]<>0)and(Link[i,inf,4]<>-1) then
          begin
            Multivisible_endpoint(i,Link[i,inf,4]);
            Unplot_of_line_segment(i,Link[i,inf,4]);
          end;
          if (Link[i,inf,5]<>0)and(Link[i,inf,5]<>-1) then
          begin
            Multivisible_endpoint(i,Link[i,inf,5]);
            Unplot_of_line_segment(i,Link[i,inf,5]);
          end;
         if (Link[i,inf,6]<>0)and(Link[i,inf,6]<>-1) then
          begin
            Multivisible_endpoint(i,Link[i,inf,6]);
            Unplot_of_line_segment(i,Link[i,inf,6]);
          end;
        end;
      end;
    for o := 1 to 7 do
    if Link[o,inf,1]=0 then display(o);
    SelectWorld(1);
    Selectwindow(10);
    setbackground(0);
    drawborder;
    copyScreen;
    SelectScreen(1);
    if StatusKbd = 'J' then JoyStic;
    if StatusKbd = 'K' then Keybord;
    Until ch = ' ';

    LeaveGraphic;

nd.{Main}
```

```
;procedure find direction of joystic
code  segment
assume cs:code
start:
            push bp
            mov bp,sp
begin     : mov dx,201h
            out dx,al
            mov cx,00bh
loop1     : in  al,dx         ;begin check loop 1
            and al,03h
            cmp al,03h
            jnz  out
            loop loop1        ;end check loop 1
            mov cx,008h
loop2     : in  al,dx         ;begin check loop 2
            and al,03h
            cmp al,01h
            jz  routine1
            cmp al,02h
            jz  routine2
            loop loop2        ;end check loop 2
            mov cx,00ah
loop3     : in  al,dx         ;begin check loop 3
            and al,03h
            cmp al,00h
            jz  loop44
            loop loop3        ;end check loop 3
loop44    : mov cx,004h
loop4     : in  al,dx         ;begin check loop 4
            and al,03h
            cmp al,01
            jz  routine5
            cmp al,02h
            jz  routine6
            loop loop4        ;end check loop 4
out       : in al,dx
            and al,30h
            cmp al,10h
            jz  routine3
            cmp al,20h
            jz  routine4
            mov cx,500h
loop5     : loop loop5
            jmp  begin
routine1 :  jmp sub1
routine2 :  jmp sub2
routine3 :  jmp sub3
routine4 :  jmp sub4
routine5 :  jmp sub5
routine6 :  jmp sub6
sub1      : les  di,[bp+8]
            mov  ax,es:[di]
            cmp  ax,12h
            jz   sub11
            cmp  ax,10h
            jz   sub12
            cmp  ax,0fh
            jz   sub13
            cmp  ax,0eh
```

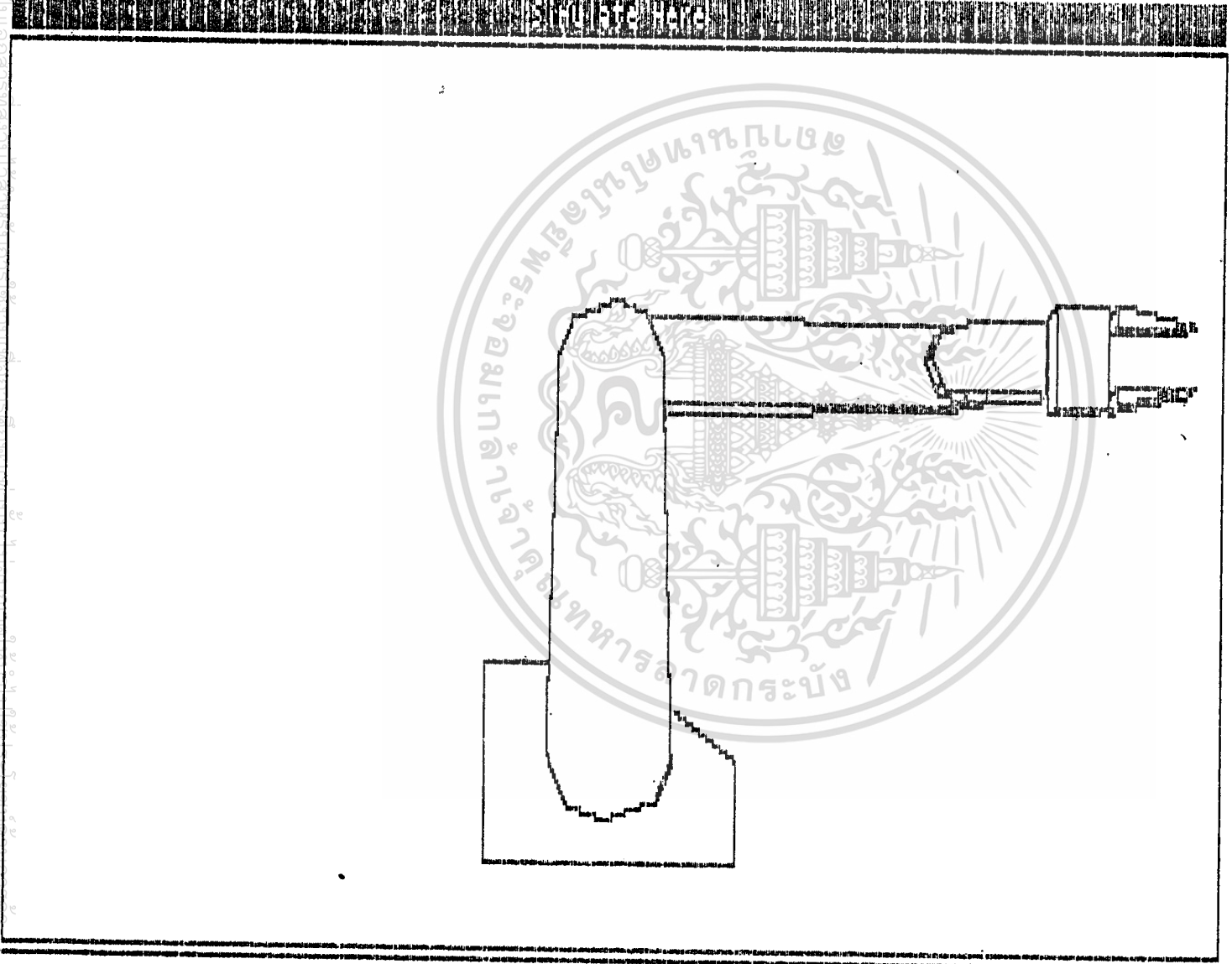
```
            les  di,[bp+4]
            mov  es:[di],ax
            pop  bp
            ret  08h
sub11   :   mov  ax,01h
            les  di,[bp+4]
            mov  es:[di],ax
            pop  bp
            ret  08h
sub12   :   mov  ax,05h
            les  di,[bp+4]
            mov  es:[di],ax
            pop  bp
            ret  08h
sub13   :   mov  ax,07h
            les  di,[bp+4]
            mov  es:[di],ax
            pop  bp
            ret  08h
sub14   :   mov  ax,09h
            les  di,[bp+4]
            mov  es:[di],ax
            pop  bp
            ret  08h
sub2    :   les  di,[bp+8]
            mov  ax,es:[di]
            cmp  ax,0dh
            jz   sub21
            cmp  ax,11h
            jz   sub22
            mov  ax,0h
            les  di,[bp+4]
            mov  es:[di],ax
            pop  bp
            ret  08h
sub21   :   mov  ax,0ch
            les  di,[bp+4]
            mov  es:[di],ax
            pop  bp
            ret  08h
sub22   :   mov  ax,04h
            les  di,[bp+4]
            mov  es:[di],ax
            pop  bp
            ret  08h
sub3    :   mov  dx,201h
            in   al,dx
            and  al,30h
            cmp  al,30h
            jnz  sub3
            les  di,[bp+8]
            mov  ax,es:[di]
            cmp  ax,12h
            jge  sub31
            inc  ax
            mov  es:[di],ax
            jmp  begin
sub31   :   mov  ax,0dh
            mov  es:[di],ax
            jmp  begin
```

```
        and  al,30h
        cmp  al,30h
        jnz   sub4
        les  di,[bp+8]
        mov  ax,es:[di]
        cmp  ax,0dh
        jle  sub41
        dec  ax
        mov  es:[di],ax
        jmp  begin
sub41   :  mov  ax,12h
        mov  es:[di],ax
        jmp  begin
sub5    :  les  di,[bp+8]
        mov  ax,es:[di]
        cmp  ax,0dh
        jz   sub51
        cmp  ax,11h
        jz   sub52
        mov  ax,0h
        mov  es:[di],ax
        pop  bp
        ret  08h
sub51   :  mov  ax,0bh
        les  di,[bp+4]
        mov  es:[di],ax
        pop  bp
        ret  08h
sub52   :  mov  ax,03h
        les  di,[bp+4]
        mov  es:[di],ax
        pop  bp
        ret  08h
sub6    :  les  di,[bp+8]
        mov  ax,es:[di]
        cmp  ax,12h
        jz   sub61
        cmp  ax,10h
        jz   sub62
        cmp  ax,0fh
        jz   sub63
        cmp  ax,0eh
        jz   sub64
        mov  ax,0h
        les  di,[bp+4]
        mov  es:[di],ax
        pop  bp
        ret  08h
sub61   :  mov  ax,02h
        les  di,[bp+4]
        mov  es:[di],ax
        pop  bp
        ret  08h
sub62   :  mov  ax,06h
        les  di,[bp+4]
        mov  es:[di],ax
        pop  bp
        ret  08h
sub63   :  mov  ax,08h
```

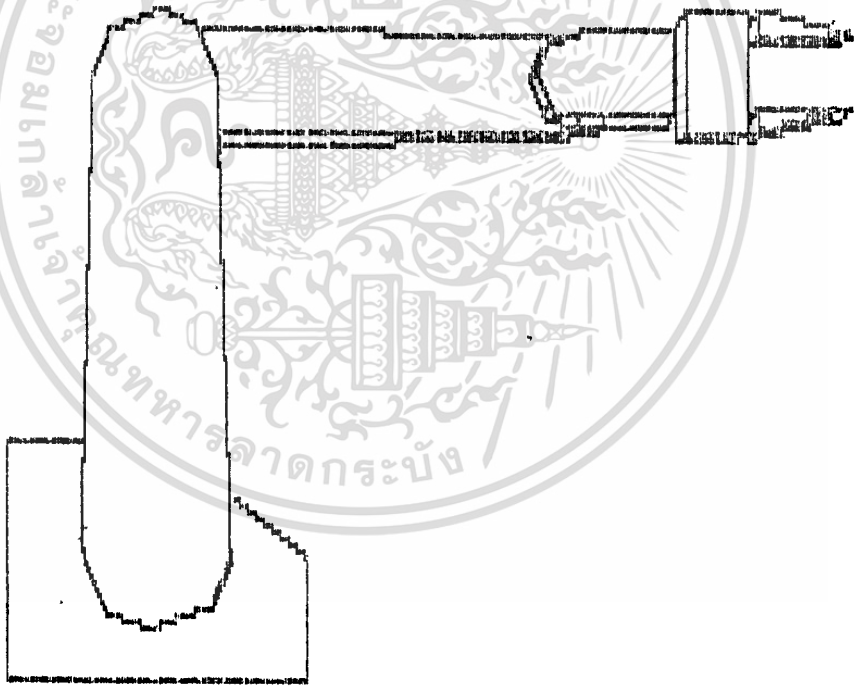
```
            ret  08h
sub64    :  mov  ax,0ah
            les  di,[bp+4]
            mov  es:[di],ax
            pop  bp
            ret  08h
code   ends
end
```

ตัวอย่างการแสดงผล

Robot Arm Simulate



Robot Arm Simulate



Arm 1

Arm 2

Arm B

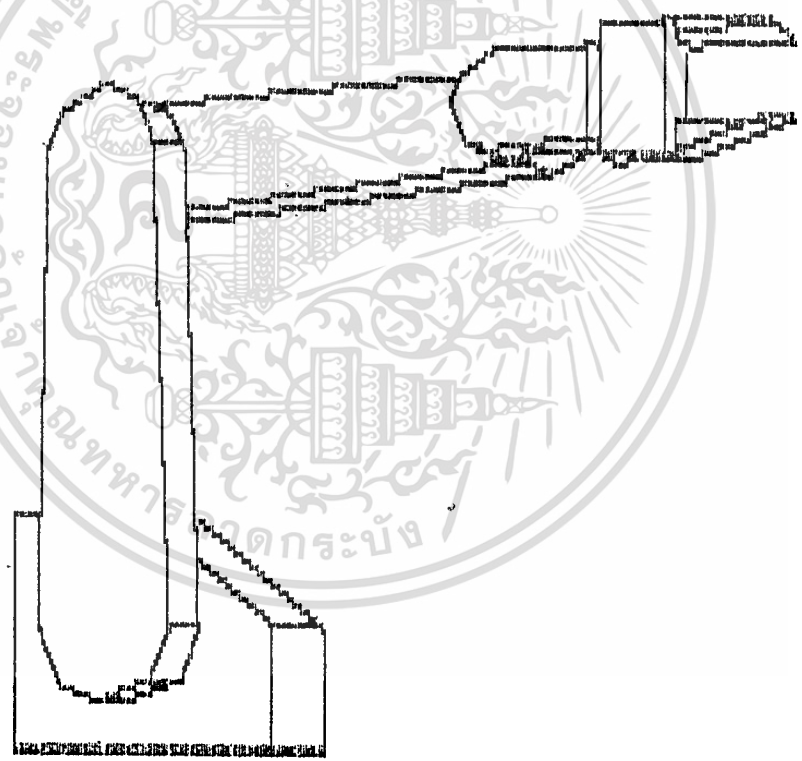
Cliper

Left

Right

UP

Robot Arm Simulate



UB

AZ

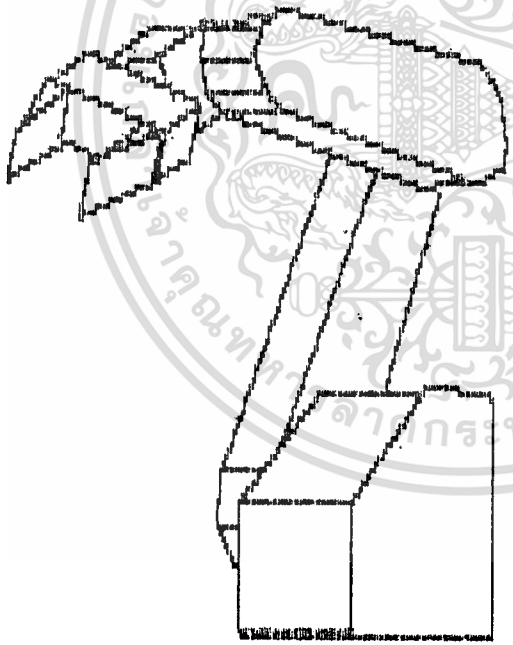
SX

DC

HN

FG

Robot Arm Simulate



V B

A Z

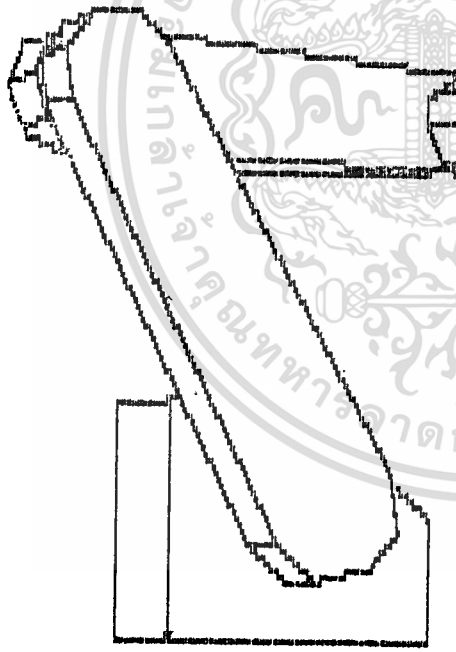
S X

D C

H N

F G

Robot Arm Simulate



V B

A Z

S X

D C

H N

F G

<u>สรุปและวิจารณ์ผลของโครงงาน</u>

ผลของโครงงานเป็นไปตามวัตถุประสงค์กล่าวคือสามารถแสดงภาพเป็นแขนหุ่นยนต์แบบ 3 มิติและสามารถควบคุมการเคลื่อนไหวของแขนหุ่นยนต์จากอุปกรณ์ภายนอกได้เช่น Joystic และ Keyboard แต่การประมวลผลของโปรแกรมค่อนข้างช้าสาเหตุเนื่องจากตัวโปรแกรมมีการคำนวณมากมายและ Micro Computer ที่ใช้มี CPU เปอร์ 8088 เพียงตัวเดียวเท่านั้น ถ้าต้องการให้มีการแสดงผลที่ต่อเนื่องมากกว่านี้ต้องใช้ Co-processor 8087 มาทำงานทางด้านคำนวณร่วมด้วย

ภาคผนวก

รายละเอียดเกี่ยวกับ Procedure ใน Graphics Toolbox

Procedure InitGraphic

Funtion : InitGraphic initializes the turbo graphix toolbox. it must be call before any other graphics procedure or funtion,but may only be called once within a program. InitGraphic selects the displayed screen as the active screen and erase it.all window and worlds are intiialized .

Procedure SelectScreen(I : integer)

Funtion : SelectScreen selects either the displayed or RAM screen for drawing . if I is 1, the displayed screen is selected .if I is 2, the RAM is select

Procedure SwapScreen

Funtion : SwapScreen exchanges the contents of the displayed screen with the contents of the RAM screen.

Procedure DefineWindow(I,XLow,YLow,XHi,YHi : integer)

Funtion : DefineWindow defines a region of the screen as a window,I. the window is define as a rectangle with the upper left corner at [XLow,YLow] and the lower right corner at [XHi,YHi].

Procedure DefineWorld(I : integer; XLow,YLow,XHi,YHi : real)

Funtion : DefineWorld defines a world coordinate system , delineated by the rectangle formed by the vertices [XLow,YLow] and [XHi,YHi]. World coordinates therefore range from [XLow,YLow] to [XHi,YHi].

Procedure SelectWindow(I : integer)

Funtion : SelectWindow selects a window I for drawing. all subsequent drawing and window commands will refer to the selected window.

Procedure SelectWorld(I : integer)

Funtion : SelectWorld selects a world coordinate system , for the drawing commands that follow. This procedure must be followed by SelectWindow to associate the world with a window.

Procedure DrawBorder

Funtion : DrawBorder draws a border around the active window in the current drawing color and line style.

Procedure DrawTextW(X,Y,Scale,Text)

Funtion : DrawTextW draws the given string , begin at world coordinates [X,Y]. The procedure uses the 4*6 pixel character set multipled both vertically and horizontally by Scale. If an ESCape (character 27 decimal) is in the string,a particular symdeb is drawn according to the next character in the string

กิตติกรรมประกาศ

ข้าพเจ้าและคณะผู้จัดทำปริญญานิพนธ์ฉบับนี้ขอขอบคุณอาจารย์กิตตินันท์ รัตนธำรงและอาจารย์ทุกท่านที่กรุณาให้คำปรึกษาตลอดจนคำแนะนำต่างๆเกี่ยวกับปัญหาและแนวทางในการแก้ไขปัญหาต่างๆจนสามารถทำให้ปริญญานิพนธ์ฉบับนี้สำเร็จสมบูรณ์

คณะผู้จัดทำ

หนังสืออ้างอิง

1. CHAN S. PARK. Interactive Microcomputer Graphics, Addison-Wesley Publising Company, 1985
2. HARRITON. STEVEN. Computer Graphics, McGRAW-Hill Book Company, 1983
3. ROBERT F. SPROULL. Principle of Interactive Computer Graphics, McGRAW-Hill Intenational Book Company, 1979