การเขียนโปรแกรมสำเร็จรูปสำหรับการสร้างภาพวัตถุในระบบสามมิติ

THREE-DIMENSIONAL GRAPHICS PACKAGE

นายสุวัฒน์ ศรีธนะรัตน์

MR.SUWAT SRITHANARAT

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2539

ISBN 974-621-632-5

THREE-DIMENSIONAL GRAPHICS PACKAGE

SUWAT SRITHANARAT

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

1996

ISBN 974-621-632-5

หัวข้อวิทยานิพนธ์	การเขียนโปรแกรมสำเร็จรูปสำหรับการสร้างภาพวัตถุในระบบสามมิติ
นักศึกษา	นายสุวัฒน์ ศรีธนะรัตน์
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.ดร. กิตติ ไพฑูรย์วัตนกิจ
ระดับการศึกษา	วิศวกรรมศาสตร์มหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า
ภาควิชา	วิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.	2539

บทคัดย่อ

จุดประสงค์ของวิทยานิพนธ์นี้ คือศึกษาวิธีการสร้างส่วนการทำงานพื้นฐานที่ใช้ในการแสดงภาพวัตถุในระบบสามมิติ ที่สามารถนำไปใช้งานได้จริง เช่น การสร้างภาพจำลองวัตถุในแบบสามมิติ และ การสร้างภาพจำลองพื้นผิวด้วย Fractal Surfaces นอกจากจะใช้เป็นแนวทางการเขียนโปรแกรมเพื่อการสร้างภาพในระบบสามมิติ ซึ่งสามารถพัฒนาเพื่อการใช้งานอื่นได้ หรือใช้ประโยชน์ทางด้านการศึกษา เพราะได้แสดงแนวคิดในการกำหนดองค์ประกอบของระบบ ความสัมพันธ์ระหว่างองค์ประกอบแต่ละส่วน รวมทั้งขั้นตอนในการสร้างส่วนการทำงานต่าง ๆ ไว้ด้วย

โปรแกรมถูกเขียนด้วยภาษาซี โดยไม่ใช้ส่วนการทำงานเกี่ยวกับสร้างภาพซึ่งมีมากับตัวภาษา เนื่องจากต้องการแสดงให้เห็นวิธีการเก็บข้อมูล และ กระบวนการแสดงภาพที่มีลักษณะเฉพาะ เพื่อให้เข้าระบบการแสดงภาพที่ได้ออกแบบไว้ ส่วนการทำงานพื้นฐานต่าง ๆ ถูกเขียนขึ้นใหม่ ไม่ว่าจะเป็นส่วนการทำงานซึ่งขึ้นอยู่กับอุปกรณ์ เช่น จอภาพ VGA หรือการใช้หน่วยความจำ XMS ส่วนการทำงานอื่น ๆ ที่เป็นพื้นฐานของระบบ เช่น การเก็บข้อมูลภาพการสร้างองค์ประกอบของภาพ การแสดงภาพ การกำหนดมุมมองของภาพ และ การประมวลผลเพื่อแสดงภาพ ถูกเขียนขึ้นมาเพื่อให้เข้ากันได้อย่างสมบูรณ์ทั้งระบบ

Thesis Title	Three-Dimensional Graphics Package
Student	Suwat Srithanarat
Thesis Advisor	Assoc.Prof. Dr.Kitti Paitoonwatanakij
Level of Study	Master of Engineering in Electrical Engineering
Department	Computer Engineering , King Mongkut's Institute of Technology Ladkrabang
Year	1996

Abstract

The purpose of this thesis is to construct a graphics programe that can be used for the creation of 3-dimensional graphics images and many other academic uses. Algorithms and procedures to be shown here can be used as a basis for the creation of a complete graphics software or developed for other applications such as the generation of the fractal surfaces and the simulation of geographical maps.

The program has been written using C language. All of the graphics primitives for the creation and the display of 3-dimensional images are explicitly written without the employment of the C Graphics Library. Display procedures, the machine-dependent procedures, can be used with any VGA display. Since all of the graphics primitives had been written to suit the designed system, they can also be applied for other purposes as a complete set of graphics primitives.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดี เพราะได้รับความสนับสนุนจากท่านรองศาสตราจารย์ ดร.กิตติ ไพฑูรย์วัฒนกิจ ที่ได้กรุณาให้คำแนะนำในการวิจัย การค้นหาเอกสารอ้างอิง ตลอดจนสนับสนุนอุปกรณ์ที่ใช้ในการวิจัยอย่างเต็มที่ ผู้วิจัยรู้สึกทราบซึ้งและกราบขอบพระคุณอย่างสูง

สุวัฒน์ ศรีธนะรัตน์

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญภาพ

หน้า

สารบัญภาพ (ต่อ)

หน้า

สารบัญภาพ (ต่อ)

หน้า

บทที่ 1
บทนำ

ภาพที่ปรากฏบนจอคอมพิวเตอร์ประกอบด้วยจุดเล็ก ๆ ซึ่งเรียกว่า จุดภาพ (pixel) เรียงติดต่อกัน ที่มาของภาพเหล่านี้ สามารถจะแบ่งได้เป็นสองกลุ่มใหญ่ ๆ กลุ่มแรกเป็นภาพที่มีลักษณะเหมือนกับภาพถ่ายหรือภาพวาด ดังตัวอย่างในภาพที่ 1 ข้อมูลภาพจะเป็นข้อมูลแบบเรียงจุด (Bitmap) เหมือนกับการเรียงจุดบนจอภาพ ข้อมูลเหล่านี้จะได้มาจากอุปกรณ์รับข้อมูลภาพต่าง ๆ เช่น กล้องถ่ายภาพหรือ เครื่องสแกนภาพ (Scanner) ซึ่งจะเปลี่ยนค่าสีที่ตำแหน่งจุดต่าง ๆ บนภาพให้เป็นข้อมูลทางคอมพิวเตอร์ ภาพที่ได้จึงเป็นภาพแบบสองมิติที่มีข้อมูลอยู่ในระนาบเดียวกันทั้งหมด

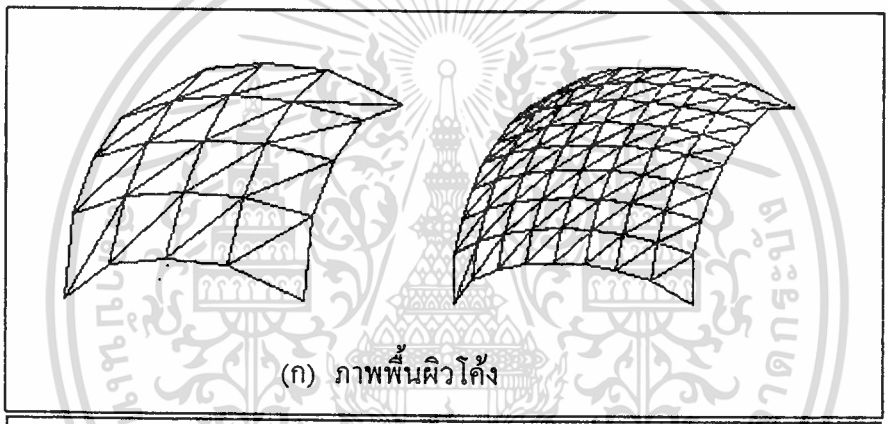
ภาพที่ 1
ภาพที่ได้จากอุปกรณ์ถ่ายภาพ

ภาพในกลุ่มที่สอง เป็นภาพซึ่งองค์ประกอบของภาพถูกสร้างขึ้น ด้วยวิธีการทางคณิตศาสตร์จากข้อมูลเบื้องต้น ดังตัวอย่างในภาพที่ 2 เป็นการสร้างภาพของพื้นผิววัตถุโดยมีข้อมูลเบื้องต้นเป็นตำแหน่งมุมทั้งสี่ของรูปสี่เหลี่ยมซึ่งจะถูกแบ่งออกเป็นสามเหลี่ยมสองรูป ภาพของพื้นผิวเกิดจากการแบ่งพื้นที่สามเหลี่ยมออกเป็นรูปสามเหลี่ยมเล็ก ๆ จำนวนมาก ตำแหน่งมุมของสามเหลี่ยมย่อยทั้งหมด จะได้จากการคำนวณค่าเบี่ยงเบนของจุดกึ่งกลางของด้านแต่ละด้าน

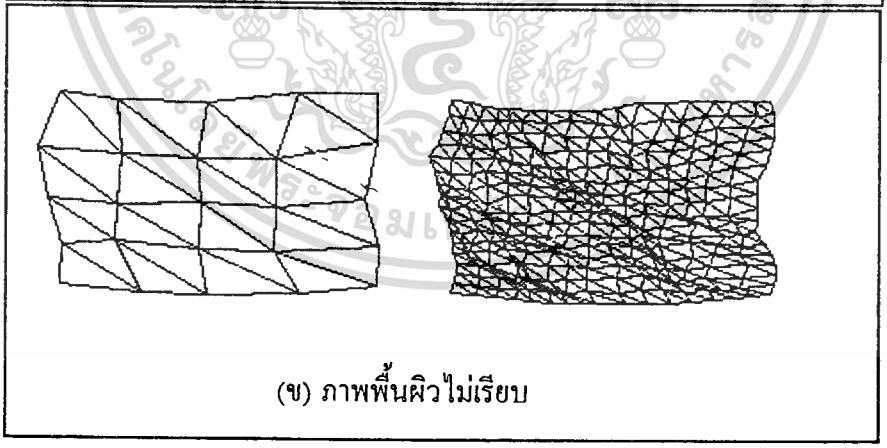
ของสามเหลี่ยมเดิม ถ้าใช้สมการของเส้นโค้งในการคำนวณค่าเบี่ยงเบน ก็จะได้ภาพพื้นผิวโค้ง ดังภาพที่ 2 (ก) แต่ถ้าใช้ค่าเบี่ยงเบนเป็นค่าสุ่ม จะได้ภาพพื้นผิวไม่เรียบที่เรียกว่า Fractal Surfaces [1] ดังภาพที่ 2 (ข) เมื่อได้ตำแหน่งมุมของรูปสามเหลี่ยมทั้งหมดแล้ว ก็จะใช้ส่วนภารทำงานภายในระบบภาพของโปรแกรม สร้างภาพของเส้นตรงขึ้นมาจากข้อมูลตำแหน่งจุดปลายทั้งสองของเส้นตรงแต่ละเส้น

ภาพที่ 2

ภาพพื้นผิวที่สร้างจากข้อมูลของรูปสี่เหลี่ยมรูปเดียว



(ก) ภาพพื้นผิวโค้ง



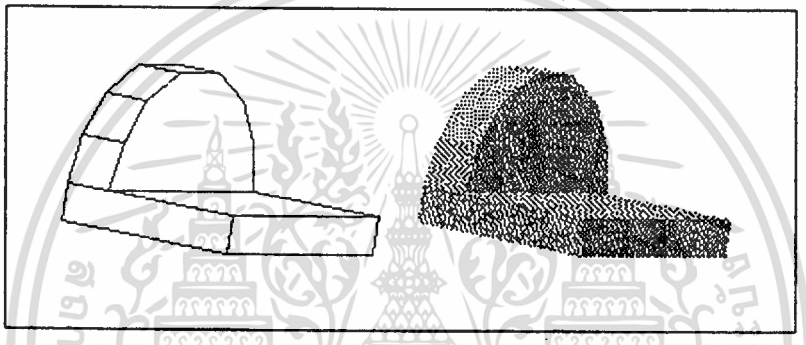
(ข) ภาพพื้นผิวไม่เรียบ

จะเห็นได้ว่าข้อมูลของภาพในลักษณะนี้ เป็นข้อมูลซึ่งสามารถกำหนดตำแหน่งและขนาดที่แน่นอนขององค์ประกอบภาพได้ ภาพแบบนี้จึงมีประโยชน์มากสำหรับงานซึ่งไม่ได้สนใจเฉพาะความสวยงามของภาพเท่านั้น แต่สามารถจะนำไปใช้กับงานที่คำนึงถึงตำแหน่งและขนาดของวัตถุในภาพด้วย ตัวอย่างเช่น การออกแบบทางสถาปัตยกรรม หรือ การสร้างแบบจำลองทางวิศวกรรม ดังตัวอย่างในภาพที่ 3 เป็นการแสดงภาพของชิ้นงาน ซึ่งออกแบบโดย

การให้ผู้ใช้เป็นผู้กำหนดรูปร่างและตำแหน่งโครงสร้างของชิ้นงาน จากนั้นก็ให้โปรแกรมทำหน้าที่คำนวณหาองค์ประกอบของภาพที่เป็นเส้นตรงทั้งหมด โดยที่ข้อมูลของเส้นตรงก็คือตำแหน่งจุดปลายในระบบสามมิติ ซึ่งนอกจากจะใช้ในการแสดงภาพแล้ว ยังสามารถนำไปใช้ในการสร้างชิ้นงานจริงได้

ภาพที่ 3

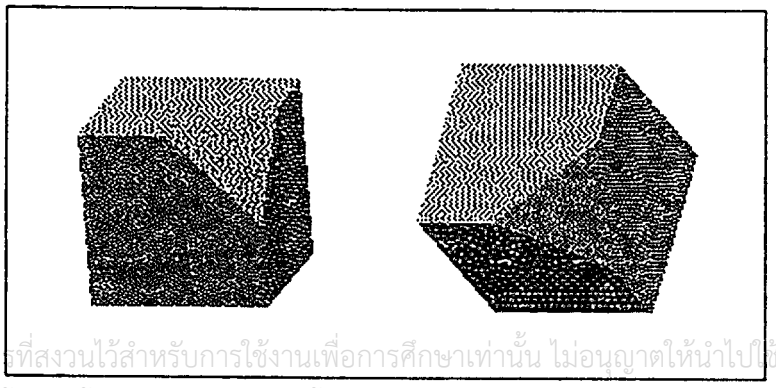
ภาพแสดงชิ้นงานในแบบสามมิติ



การที่มีข้อมูลวัตถุภาพเป็นตำแหน่งจุด ทำให้สามารถใช้กระบวนการทางคณิตศาสตร์เปลี่ยนแปลงลักษณะของวัตถุ ไม่ว่าจะเป็นการเปลี่ยนขนาด การย้ายตำแหน่ง หรือ การหมุน ดังตัวอย่างในภาพที่ 4 เป็นภาพวัตถุรูปสี่เหลี่ยมถูกบาศก์ตัดมุม ซึ่งสร้างจากข้อมูลตำแหน่งมุมของวัตถุนั้น ภาพของวัตถุทั้งสองชิ้นสร้างขึ้นจากข้อมูลชุดเดียวกัน แต่ถูกกำหนดให้หมุนไปด้วยมุมที่ต่างกัน

ภาพที่ 4

ภาพของวัตถุซึ่งจัดวางในมุมที่ต่างกัน



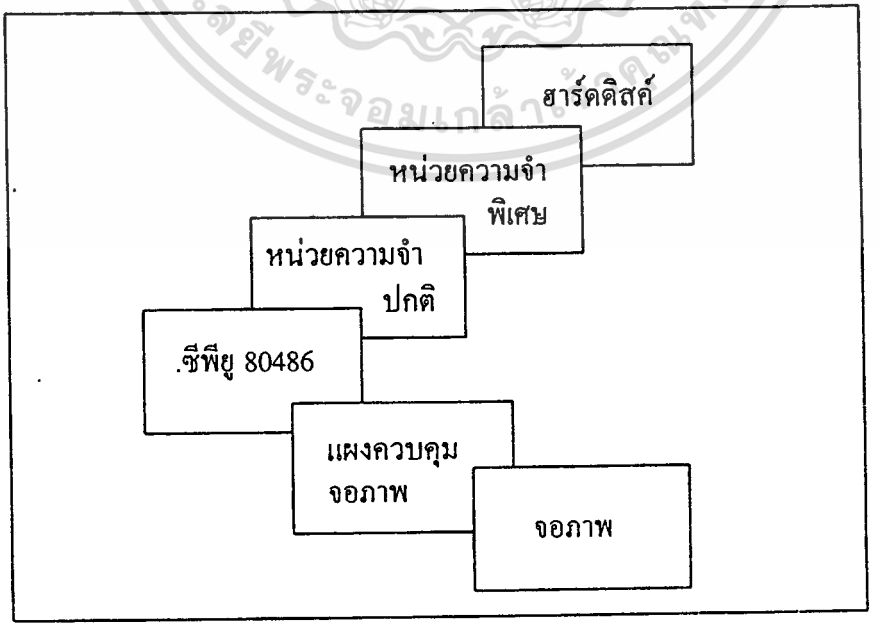
สำหรับในประเทศไทย งานที่เกี่ยวข้องกับการสร้างภาพด้วยคอมพิวเตอร์จะใช้โปรแกรมที่มาจากต่างประเทศเป็นส่วนใหญ่ ดังนั้น วิทยานิพนธ์นี้จึงมีวัตถุประสงค์ในการสร้างแนวทางสำหรับการพัฒนาโปรแกรมเกี่ยวกับการสร้างภาพ เพื่อประโยชน์ทางการศึกษาและการนำไปประยุกต์ใช้กับงานต่าง ๆ โดยเลือกเอาเรื่องการสร้างภาพจำลองวัตถุในระบบสามมิติ ซึ่งจะครอบคลุมไปถึงบางส่วนของการสร้างแสดงภาพในระบบสองมิติด้วย เริ่มจากการกำหนดโครงสร้างของระบบสร้างภาพทั้งหมด พร้อมกับเขียนส่วนการทำงานที่เกี่ยวกับสร้างภาพขึ้นใหม่จากคำสั่งพื้นฐาน โดยการนำทฤษฎีการสร้างภาพสามมิติมาพัฒนาเป็นส่วนการทำงานที่เข้ากันได้อย่างสมบูรณ์กับระบบที่ต้องการ และสามารถนำไปปรับเข้ากับระบบที่ต้องการได้ง่าย

บทที่ 2
โครงสร้างของระบบ

• ระบบคอมพิวเตอร์สำหรับการสร้างภาพ

โครงสร้างของระบบคอมพิวเตอร์สำหรับการสร้างภาพ ประกอบด้วยส่วนที่เป็นตัวเครื่องคอมพิวเตอร์ และส่วนของโปรแกรม ในภาพที่ 5 แสดงโครงสร้างทั่วไปของเครื่องคอมพิวเตอร์ที่ใช้ในการสร้างภาพ โดยในที่นี้จะใช้ไมโครคอมพิวเตอร์ในระบบของไอบีเอ็มซึ่งประกอบด้วยซีพียู 80486 โดยมี Math CoProcessor ซึ่งจะช่วยให้เครื่องมีความเร็วในการคำนวณตัวเลขมากพอที่จะใช้คำนวณตำแหน่งจุดภาพ สำหรับภาพที่มีองค์ประกอบภาพจำนวนมาก และแสดงภาพนั้นออกมาได้โดยไม่ใช้เวลานานจนเกินไป

ภาพที่ 5
ระบบคอมพิวเตอร์พื้นฐาน

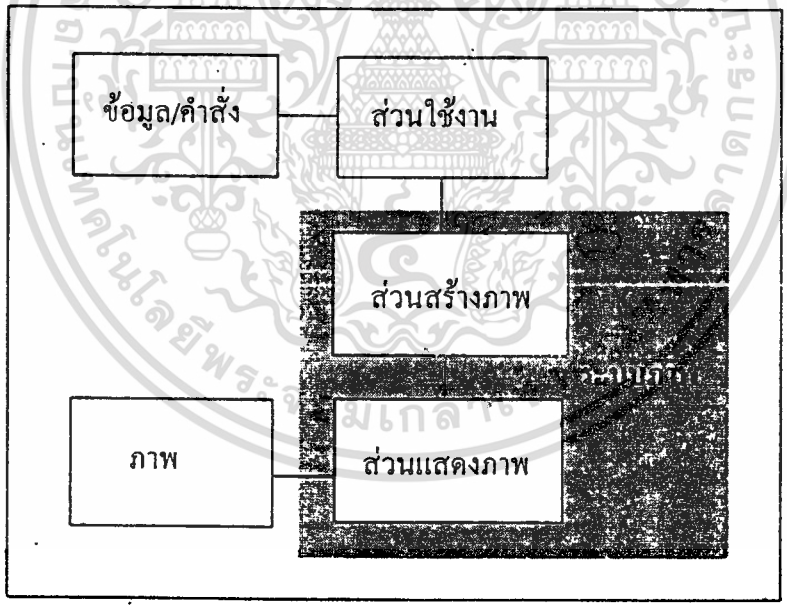


ส่วนประกอบที่สำคัญอย่างหนึ่งของตัวเครื่อง ซึ่งเกี่ยวข้องกับการเขียนโปรแกรม ได้แก่ อุปกรณ์ที่ใช้แสดงภาพ ในปัจจุบันจะใช้จอภาพแบบจุดสว่าง (Raster Screen) ซึ่งแสดงภาพด้วยจุดสีเล็ก ๆ จำนวนมาก และจอภาพที่ใช้กันมากก็คือ จอภาพแบบ VGA (Video Graphics Array) เนื่องจากเป็นอุปกรณ์แสดงผลที่มีราคาไม่สูงมากและสามารถแสดงภาพที่ให้รายละเอียดภาพได้ถึง 1024x768 จุด แสดงสีได้พร้อมกันทั้งหมด 256 สี

ส่วนของหน่วยความจำซึ่งเป็นส่วนสำหรับเก็บโปรแกรมและข้อมูลที่ใช้ในระบบนั้น ถูกจำกัดโดยวิธีการอ้างตำแหน่งให้มีขนาดเพียง 640K ในกรณีที่ข้อมูลทั้งหมดมีจำนวนมากเกินกว่าที่จะเก็บในส่วนความจำปกติได้ ก็จะเลือกเก็บข้อมูลบางส่วนที่ใช้ในระหว่างการทำงานไว้ในหน่วยความจำพิเศษ เช่น หน่วยความจำส่วนขยาย (Extended Memory)

ภาพที่ 6

โครงสร้างหลักของโปรแกรม



ในส่วนของโปรแกรม จะใช้ภาษาซีสำหรับการเขียนส่วนการทำงานทั้งหมด โดยเลือกใช้ระบบภาษาซีของ Borland C และ Turbo C ซึ่งเป็นภาษาที่มีความคล่องตัวในการใช้งานมาก ลักษณะที่เป็นโครงสร้างของภาษาซีทำให้เห็นขั้นตอนการทำงานได้ชัดเจน การใช้คำสั่งก็มีความยืดหยุ่นมาก สามารถเลือกคำสั่งที่เหมาะกับการทำงานได้ โดยเฉพาะในการเขียนส่วนควบคุมอุปกรณ์ การเขียนโปรแกรมจะใช้วิธีแยกส่วนการทำงานออกเป็นส่วน ๆ เพื่อความสะดวกในการนำแต่ละส่วนมาใช้งานหรือแก้ไขเพิ่มเติม

ส่วนใช้งานเป็นส่วนที่มีองค์ประกอบเปลี่ยนไปตามสภาพการใช้งาน ถ้าเป็นงานที่มีการติดต่อรับข้อมูลจากผู้ใช้ในแบบ Interactive [2] จะต้องเน้นในเรื่องของการติดต่อกับผู้ใช้ (User Interface) แต่ถ้าเป็นการสร้างภาพจากแฟ้มภาพ ก็จะให้ความสำคัญกับรูปแบบของการเก็บข้อมูลมากกว่า ดังนั้น ในการกำหนดโครงสร้างหลักของโปรแกรมสร้างภาพวัตถุในระบบสามมิติ ส่วนของโปรแกรมใช้งาน (Application Program) จึงถูกแยกออกจากส่วนของระบบภาพ (Graphics System) ดังรูปที่ 6 เพื่อให้ได้ระบบภาพที่ไม่ขึ้นอยู่กับลักษณะการใช้งาน หน้าที่ของส่วนใช้งานก็คือ เปลี่ยนข้อมูลภาพจากผู้ใช้โปรแกรมหรือจากแฟ้มภาพ ให้เป็นคำสั่งเรียกใช้ส่วนการทำงานพื้นฐานในระบบภาพ ตัวอย่างเช่นคำสั่ง

```
Draw_Line_3D (point1, point2, color);
```

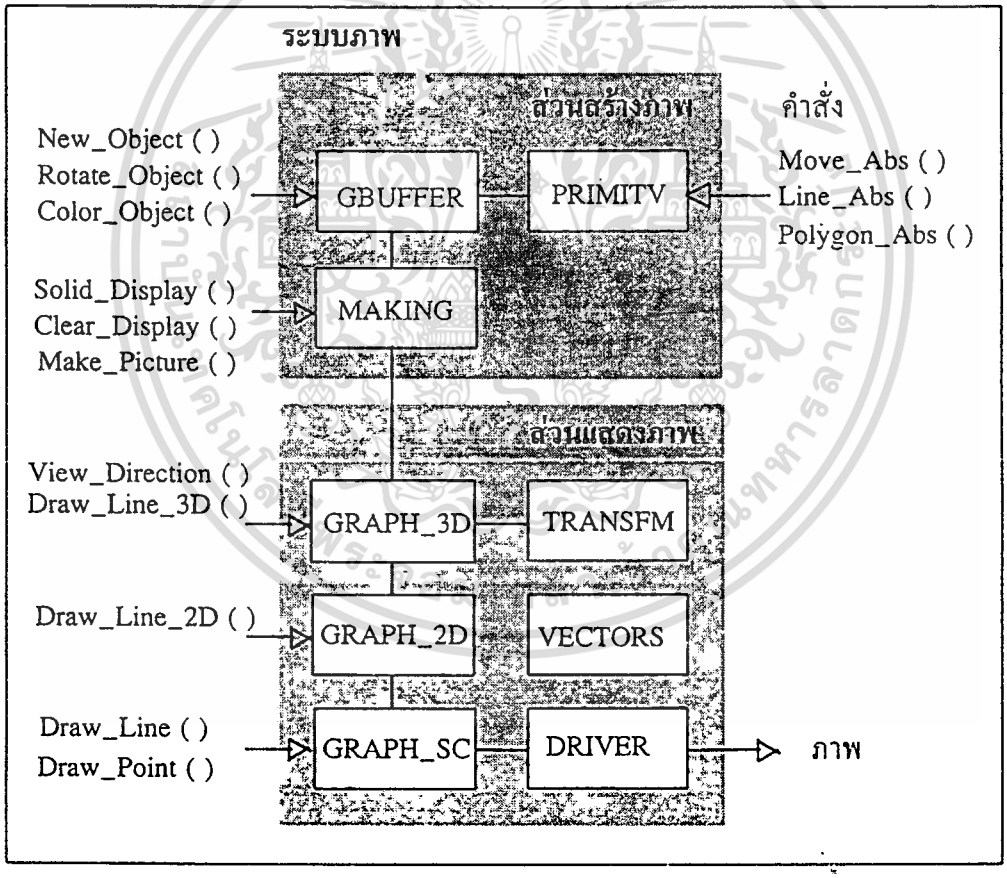
เป็นคำสั่งให้ลากเส้นตรงให้แสดงเส้นตรงสี color จากตำแหน่งจุด point1 ไปยังตำแหน่งจุด point2 โดยส่วนการทำงานในระบบภาพจะเปลี่ยนค่าตำแหน่งจุดทั้งสอง ซึ่งเป็นตำแหน่งในระบบสามมิติ ให้เป็นตำแหน่งบนจอภาพ แล้วนำตำแหน่งนั้นไปแสดงภาพของเส้นตรงบนจอภาพ ในที่นี้จะได้ แสดงตัวอย่างการกำหนดโครงสร้างของระบบภาพที่ต้องการ และกระบวนการในการสร้างส่วนการทำงานซึ่งเป็นองค์ประกอบพื้นฐานของระบบภาพ ก่อนที่จะกล่าวถึงตัวอย่างของการนำส่วนการทำงานเหล่านี้ไปใช้งาน

โครงสร้างของระบบภาพ

ในเบื้องต้นนี้ จะพูดถึงส่วนของระบบภาพ ซึ่งเป็นส่วนพื้นฐานที่นำไปประยุกต์ใช้กับงานต่าง ๆ จากภาพที่ 7 จะเห็นได้ว่า ระบบภาพสามารถแยกองค์ประกอบหลักได้เป็นสองส่วน คือ ส่วนแสดงภาพ และ ส่วนสร้างภาพ

ภาพที่ 7

โครงสร้างของระบบภาพ



ส่วนแสดงภาพประกอบด้วย ส่วนการทำงานพื้นฐานสำหรับการแสดงภาพ โดยมีส่วนสำหรับการแสดงจุดภาพและเส้นตรงเป็นส่วนการทำงานหลัก นอกจากนั้นก็จะเป็นส่วนที่กำหนดสภาพแวดล้อมของการแสดงภาพ ในขณะที่ ส่วนสร้างภาพ จะใช้ในกรณีที่องค์ประกอบแต่ละส่วนมีผลกระทบถึงกัน ตัวอย่างเช่น วัตถุในภาพอยู่ในตำแหน่งที่มีการบังกัน

เป็นต้น โดยคำสั่งสร้างภาพทั้งหมดจะถูกเก็บรวบรวมไว้ แล้วนำไปผ่านกระบวนการประมวลผลข้อมูลจนได้เป็นคำสั่งแสดงภาพ

กระบวนการที่ระบบภาพรับข้อมูลและคำสั่งกำหนดรายละเอียดของภาพจากภายนอก จะต่างกันไปตามลักษณะการใช้งาน กระบวนการเหล่านี้จะเป็นตัวกำหนดโครงสร้างของส่วนใช้งาน ซึ่งจะพูดถึงในส่วนของการนำระบบภาพไปใช้ สำหรับส่วนการสร้างภาพจะแบ่งตามขั้นตอนการทำงานออกเป็น 3 ส่วน แต่ละส่วนประกอบด้วยส่วนการทำงานซึ่งมีหน้าที่ต่างกัน ดังนี้

PRIMITV ประกอบด้วยส่วนการทำงานพื้นฐาน (Primitives) สำหรับการสร้างภาพ ซึ่งทำหน้าที่เปลี่ยนคำสั่งและข้อมูลภาพทั้งหมดให้เป็นองค์ประกอบพื้นฐานของระบบภาพ ตัวอย่างเช่น การสร้างภาพของรูปหลายเหลี่ยม (polygon) จะใช้คำสั่ง

Polygon_Abs (n, points)

โดยที่ points เป็นข้อมูลตำแหน่งมุมของรูปหลายเหลี่ยม ซึ่งจะถูกเปลี่ยนให้เป็นองค์ประกอบพื้นฐาน ได้แก่ การสร้างเส้นตรงที่เป็นด้านแต่ละด้าน ดังนี้

Line_Abs (point)

กำหนดให้ลากเส้นจากตำแหน่งปัจจุบันไปยังจุดต่อไป คือ point เมื่อลากเส้นต่อเนื่องจนครบทุกจุด ก็จะได้รูปหลายเหลี่ยมที่ต้องการ องค์ประกอบพื้นฐานทั้งหมดจะถูกส่งไปให้ส่วนการทำงานใน GBUFFER เก็บรวบรวมไว้

GBUFFER เป็นส่วนที่ทำหน้าที่เก็บข้อมูลภาพที่ได้จากส่วน PRIMITV โดยข้อมูลทั้งหมดจะถูกเก็บเป็นกลุ่มของวัตถุภาพแต่ละชิ้นในเนื้อที่ซึ่งจัดเตรียมไว้โดยคำสั่ง

New_Object ( )

แต่ละกลุ่มก็จะมีคุณสมบัติประจำกลุ่ม ตัวอย่างเช่น ค่าการหมุนก็จะถูกกำหนดด้วยคำสั่ง

Rotate_Object (rx, ry, rz)

เมื่อเก็บข้อมูลพื้นฐานขององค์ประกอบภาพได้ทั้งหมดแล้ว ก็จะส่งข้อมูลเหล่านี้เข้าสู่กระบวนการสร้างภาพ

MAKING ก่อนที่จะนำข้อมูลภาพทั้งหมดมาผ่านกระบวนการแปลข้อมูล ต้องกำหนดรูปแบบของการแสดงภาพ เช่น

```
Solid_Display ( )      // ให้แสดงภาพแบบทึบแสง
Clear_Display ( )      // ให้ลบภาพเดิมก่อน
```

เมื่อได้ข้อมูลพร้อมกับรูปแบบการแสดงภาพแล้ว ก็จะสั่งให้นำข้อมูลมาสร้างเป็นภาพโดยใช้คำสั่ง

```
Make_Picture ( )
```

ข้อมูลภาพจะถูกเปลี่ยนให้เป็นคำสั่งแสดงภาพ เพื่อให้ส่วนแสดงภาพทำการส่งภาพที่ได้ไปยังจอภาพ

ส่วนแสดงภาพ เป็นส่วนการทำงานที่เกี่ยวข้องกับการแสดงองค์ประกอบพื้นฐานของภาพ เช่น การแสดงจุดภาพ หรือ การลากเส้นตรง ส่วนการทำงานสำหรับการแสดงภาพสามารถจะแบ่งตามลักษณะของข้อมูลได้เป็น 3 ระดับ โครงสร้างแต่ละระดับประกอบด้วยส่วนที่สามารถนำไปใช้งานได้ในสภาพแวดล้อมที่ต่างกัน ดังนี้

GRAPH_3D เป็นส่วนการทำงานสำหรับการแสดงภาพ ซึ่งมีตำแหน่งต่าง ๆ เป็นเวิคเตอร์ซึ่งอยู่ในระบบ 3 มิติ (3-Dimensional Graphics) ตัวอย่างเช่น

```
Draw_Line_3D (point1, point2, color);
```

ข้อมูลตำแหน่งที่ใช้ เป็นข้อมูลที่มีหน่วยเป็นกลาง (normalized coordinate) ซึ่งจะถูกเปลี่ยนเป็นข้อมูลในระบบ 2 มิติ เพื่อใช้ส่วนการทำงานใน GRAPH_2D อีกทีหนึ่ง

GRAPH_2D ส่วนการทำงานสำหรับการแสดงภาพแบบ 2 มิติ ซึ่งใช้ข้อมูลตำแหน่งภาพที่ประกอบด้วยแกน 2 แกน ตัวอย่างเช่น

```
Draw_Line_2D (x1, y1, x2, y2, color);
```

ข้อมูลตำแหน่งที่มีหน่วยเป็นกลางจะถูกเปลี่ยนเป็นตำแหน่งบนจอภาพ เพื่อที่จะใช้ส่วนการทำงานใน GRAPH_SC ในการแสดงภาพ

GRAPH_SC ประกอบด้วยส่วนการทำงานสำหรับแสดงภาพออกที่จอภาพ ข้อมูลตำแหน่งที่ใช้ก็จะเป็นตำแหน่งจริงบนจอภาพ (Screen Coodinate) ตัวอย่างเช่น

Draw_Line (x1, y1, x2, y2, color);

ส่วนการทำงานนี้จะทำหน้าที่ลากเส้นตรงจากตำแหน่ง (x1,y1) บนจอภาพ ไปยังตำแหน่ง (x2,y2)

นอกจากส่วนที่เกี่ยวข้องกับการแสดงภาพโดยตรงแล้ว การใช้งานที่เกี่ยวข้องกับข้อมูลในระบบสามมิติ ต้องอาศัยส่วนการทำงานต่อไปนี้

TRANSFM เป็นส่วนการทำงานสำหรับการเปลี่ยนพิกัดข้อมูลบอกตำแหน่งในระบบสามมิติ อันเกิดจากการหมุนวัตถุภาพ หรือ ทำให้วัตถุภาพมีขนาดหรือตำแหน่งที่ผิดไปจากเดิม

VECTORS เป็นส่วนการทำงานที่เกี่ยวข้องกับการใช้งานเวิ์คเตอร์ ซึ่งเป็นข้อมูลตำแหน่งในระบบสามมิติ เช่น การกำหนดค่าให้เวิ์คเตอร์ หรือการคำนวณต่าง ๆ

สำหรับขั้นตอนสุดท้ายก่อนที่จะได้ภาพที่ต้องการจากระบบภาพ ก็คือการใช้คำสั่งแสดงผลออกที่จอภาพ ในการแสดงจุดภาพบนจอภาพ จะมีส่วนการทำงาน DRIVER ทำหน้าที่เป็นจอภาพเสมือน (Virtual Screen) เพื่อให้ส่วนแสดงภาพไม่มีการติดต่อกับจอภาพโดยตรง ทำให้ระบบภาพเป็นอิสระจากอุปกรณ์ที่ใช้ในการแสดงภาพ (Device Independence) และไม่ได้รับผลกระทบใด ๆ จากการเปลี่ยนแปลงชนิดของจอภาพที่ใช้เป็นอุปกรณ์แสดงภาพ

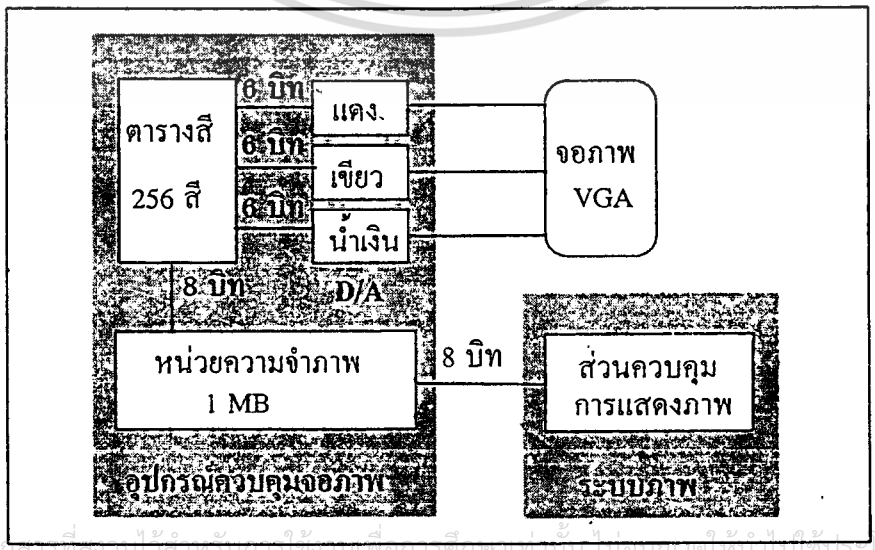
บทที่ 3
ส่วนการทำงานที่ขึ้นอยู่กับอุปกรณ์

ส่วนควบคุมการแสดงภาพ

ปัญหาอย่างหนึ่งของการเขียนโปรแกรมแสดงภาพ ก็คือ การทำให้ส่วนการทำงานของโปรแกรมเป็นอิสระจากอุปกรณ์ที่ใช้ในระบบ (Device Independence) หรือมีผลกระทบที่เกิดจากความเปลี่ยนแปลงของระบบน้อยที่สุด โดยเฉพาะอย่างยิ่ง จอภาพ ซึ่งเป็นอุปกรณ์ที่สำคัญสำหรับระบบแสดงภาพ เป็นอุปกรณ์ที่มีความหลากหลายและยังไม่เป็นมาตรฐานเดียวกันทั้งหมด จอภาพที่จะใช้ในระบบนี้เป็นจอภาพแบบจุดสว่าง (Raster Display) เหมือนกับจอโทรทัศน์ ภาพบนจอประกอบด้วยจุดภาพเล็ก ๆ (Picture Element) ที่เรียกว่า Pixel จำนวนมาก ภาพที่ 8 เป็นแผนผังโครงสร้างของแผงอุปกรณ์ควบคุมการแสดงภาพ ซึ่งทำหน้าที่เปลี่ยนข้อมูลดิจิตอลจากคอมพิวเตอร์ให้เป็นสัญญาณอนาลอกสำหรับจอภาพ

ภาพที่ 8

โครงสร้างของแผงอุปกรณ์ควบคุมจอภาพ VGA



ข้อมูลจุดภาพจากโปรแกรมถูกส่งไปเก็บไว้ที่หน่วยความจำภาพ (Display Buffer) บนแผง อุปกรณ์ควบคุมจอภาพ ซึ่งทำหน้าที่สร้างสัญญาณภาพส่งไปยังจอภาพตลอดเวลา และเนื่องจากการจัดเก็บข้อมูลเรียงตามลำดับของจุดบนจอภาพ จอภาพในแบบนี้จึงถูกเรียกว่า จอ VGA (Video Graphics Array) จุดบนจอภาพจะเกิดจากการผสมกันของ แม่สี 3 สี ได้แก่ สีแดง สีเขียว และ สีน้ำเงิน ข้อมูลสีแต่ละสีถูกจำกัดโดยอุปกรณ์แปลงสัญญาณ (D/A) ให้มีขนาด 6 บิท เมื่อรวม ทั้งสามสีก็ต้องใช้ข้อมูลถึง 18 บิท ในขณะที่ข้อมูลจุดภาพจากโปรแกรมสร้างภาพเป็นข้อมูลขนาด 8 บิท ไม่สามารถนำมาเป็นข้อมูลสีโดยตรงได้ แต่จะใช้เป็นค่าชี้ตำแหน่งในตารางสี (Palette) โดยที่ข้อมูลในตารางสีซึ่งเป็นข้อมูลขนาด 18 บิท จะถูกนำมาเปลี่ยนเป็นสัญญาณสีเพื่อส่งไปให้จอภาพอีกต่อหนึ่ง เนื่องจากข้อมูลชี้ตำแหน่งมีขนาด 8 บิท ตารางนี้จึงเก็บข้อมูลสีได้ 256 ตำแหน่ง นั่นก็หมายความว่า โปรแกรมภาพจะแสดงสีพร้อมกันได้เพียง 256 สี แต่ยอมให้ผู้ใช้เปลี่ยนค่าสีในตารางข้อมูลสีเพื่อสร้างสีที่เหมาะกับงานได้

ในการแสดงภาพบนจอ VGA นั้น จะมีแบบการแสดงภาพที่มีความละเอียดต่างกันหลายแบบ และแผงอุปกรณ์ควบคุมการแสดงภาพของแต่ละบริษัทก็จะมีความแตกต่างกัน ตารางที่ 1 แสดงความแตกต่างที่เห็นได้ชัดเจน นั่นคือ มีการใช้หมายเลขของการแสดงภาพ (Display Mode) ต่างกัน สำหรับการแสดงภาพขนาดเดียวกัน

ตารางที่ 1

หมายเลขของแบบแสดงภาพ 256 สีบนแผงวงจรต่าง ๆ

	ความละเอียดของภาพ (Resolution)		
แผงวงจร	640x480	800x600	1024x768
TRIDENT	5Dh	5Eh	62h
PARADISE	5Fh	5Ch	60h
TSENG	2Eh	30h	38h
ATI	62h	63h	64h

สำหรับภาษาซีที่ใช้ในการเขียนโปรแกรม มีส่วนการทำงานเกี่ยวกับการแสดงผลซึ่งจะใช้งานได้ดีกับแบบการแสดงภาพที่เป็นมาตรฐาน แต่ถ้าต้องการความเร็วในการทำงาน ก็สามารถ

เขียนส่วนควบคุมการแสดงภาพให้นำข้อมูลสีไปใส่ในหน่วยความจำภาพโดยตรง ซึ่งจะต้องมีการคำนวณตำแหน่ง ดังนี้

ตำแหน่งจริง = ความกว้างของจอภาพ * ตำแหน่งแนวตั้ง + ตำแหน่งแนวนอน

ในการอ้างอิงตำแหน่งข้อมูลของหน่วยความจำภาพ เลขตำแหน่งถูกจำกัดไว้เพียง 64K คือตั้งแต่ A000:0000 ถึง A000:FFFF ในขณะที่ต้องใช้หน่วยความจำในการแสดงภาพที่มีความละเอียดสูงมากกว่า 64K จึงต้องแบ่งหน่วยความจำภาพออกเป็นส่วนย่อย (page) ที่มีขนาด 64K ในการอ้างอิงตำแหน่งเพื่อใส่ข้อมูลจุดภาพก็ต้องระบุด้วยว่าอยู่ในส่วนย่อยส่วนไหน การหาเลขลำดับส่วนย่อยของหน่วยความจำภาพ และตำแหน่งภายในส่วนความจำย่อยทำได้ดังนี้

ลำดับที่ส่วนย่อย = ตำแหน่งจริง / 64K
ตำแหน่งภายในส่วนย่อย = ตำแหน่งจริง - (ลำดับที่ส่วนย่อย * 64K)

แผงอุปกรณ์ควบคุมภาพของแต่ละบริษัทมีความแตกต่างในวิธีการเลือกส่วนความจำภาพ [3] ทำให้เป็นปัญหาอย่างมากในการเขียนส่วนแสดงภาพ ถ้าจะตัดปัญหาเรื่องความแตกต่างของแผงอุปกรณ์ควบคุมภาพ ก็สามารถจะใช้ Interrupt 10h ติดต่อกับส่วนควบคุมการแสดงภาพใน BIOS (Basic I/O System) ดังนี้

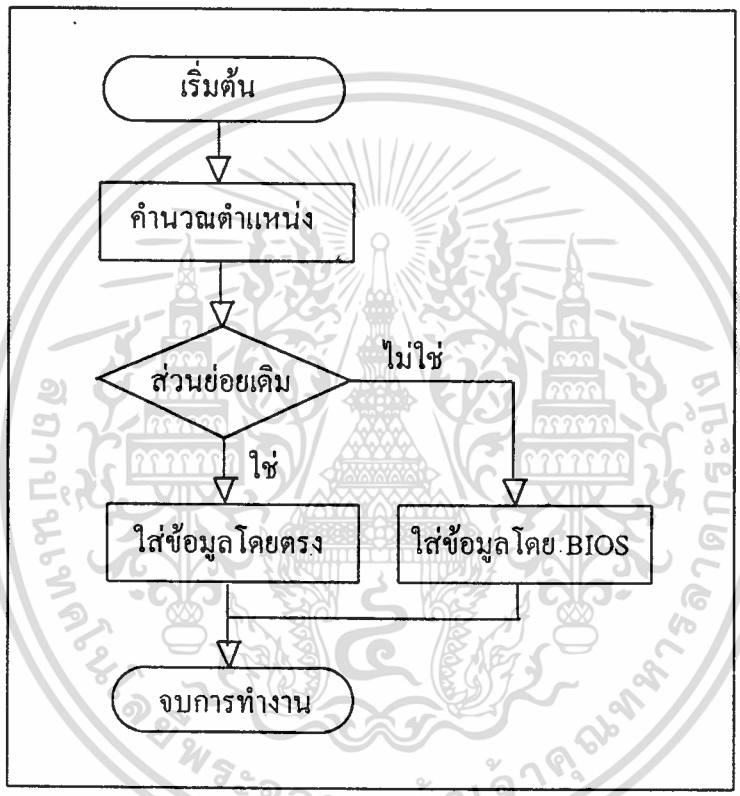
```
MOV   DX, ตำแหน่งแกน X
MOV   CX, ตำแหน่งแกน Y
MOV   AL, ค่าสี
MOV   AH,0CH              ;คำสั่งบันทึกข้อมูล
INT   10H                 ;เรียก BIOS
```

ส่วนการทำงานภายใน BIOS จะทำหน้าที่คำนวณตำแหน่ง เลือกส่วนความจำ รวมไปถึงการส่งข้อมูลสีไปที่ตำแหน่งของหน่วยความจำภาพที่คำนวณได้ ถึงแม้ว่าวิธีนี้จะสะดวกสำหรับการเขียนโปรแกรม แต่เมื่อนำส่วนการทำงานนี้ไปทดลองใช้ดูแล้วปรากฏว่า แสดงผลได้ช้ากว่าการใส่ข้อมูลจุดภาพโดยตรงมาก ดังนั้น เพื่อที่จะแก้ปัญหาเรื่องความแตกต่างของการเลือกส่วน

ความจำภาพ และในขณะเดียวกันก็ลดเวลาที่ใช้ในการแสดงจุดภาพด้วย จึงได้นำวิธีการทั้งสองมาใช้ร่วมกัน โดยมีขั้นตอนดังแสดงในภาพที่ 9

ภาพที่ 9

ขั้นตอนการแสดงจุดภาพ



หลักการสำคัญของวิธีการนี้อยู่ที่ว่า เมื่อต้องการอ้างอิงตำแหน่งจุดภาพซึ่งอยู่ในหน่วยความจำภาพส่วนเดิม ก็ไม่จำเป็นต้องระบุส่วนความจำที่ต้องการ ดังนั้น เมื่อมีการตรวจสอบแล้วว่า ตำแหน่งของจุดอยู่ในหน่วยความจำภาพส่วนเดิม ก็สามารถจะใส่ข้อมูลสีลงในหน่วยความจำภาพส่วนนั้นได้เลย โดยไม่ต้องเลือกส่วนความจำ แต่ถ้ามีการเปลี่ยนส่วนความจำใหม่ก็จะใช้ BIOS ในการแสดงจุดภาพ

ในตารางที่ 2 แสดงเวลาที่ใช้ในการแสดงเส้นตรงที่เกิดจากการสุ่มตำแหน่ง 5000 เส้น โดยใช้ส่วนการแสดงจุดภาพทั้งสามแบบในการลากเส้นตรง ซึ่งจะเห็นได้ว่า ส่วนการทำงานที่ใช้ BIOS จะช้ากว่าการใส่ข้อมูลโดยตรงมาก ในขณะที่ วิธีตรวจสอบส่วนความจำจะช่วยให้แสดงจุดภาพได้เร็วขึ้น ทั้งนี้ก็เนื่องจากว่า จุดภาพที่อยู่ติดกันส่วนมากจะอยู่ภายในส่วนความจำเดียวกัน ทำให้การแสดงจุดภาพต่อเนื่องกันเป็นการใส่ข้อมูลโดยตรงเป็นส่วนใหญ่

ตารางที่ 2

เปรียบเทียบวิธีที่ใช้ในการแสดงจุดภาพบนเครื่อง 486DX2-66 และแผงควบคุมภาพของ TRIDENT

ลากเส้นตรง 5000 เส้น	เวลา (วินาที)
ใช้ BIOS	68
ใส่ข้อมูลโดยตรง	7
ตรวจสอบส่วนความจำ	4

การที่ยังไม่มีการกำหนดมาตรฐานให้กับแผงอุปกรณ์ควบคุม ทำให้ส่วนควบคุมการแสดงภาพเป็นส่วนที่มีการเปลี่ยนแปลงไปตามสภาพของการใช้งานได้มากที่สุด โดยทั่วไปโปรแกรมที่เกี่ยวกับการแสดงภาพจะแยกส่วนควบคุมการแสดงภาพ โดยเฉพาะส่วนที่เกี่ยวข้องกับอุปกรณ์ออกไปไว้ในแฟ้มเป็น Loadable Driver ที่สามารถดึงเข้ามารวมกับโปรแกรมหลักในขณะใช้งานได้ เพื่อให้โปรแกรมหลักเป็นอิสระจากอุปกรณ์ควบคุมการแสดงภาพ การเขียนส่วนควบคุมการแสดงภาพโดยใช้ภาษาแอสเซมบลี (ภาคผนวก ก) จะมีความสะดวกในการติดต่อกับแผงอุปกรณ์ควบคุมการแสดงภาพ ได้ส่วนการทำงานที่กระชับ และทำงานได้รวดเร็วกว่าส่วนการทำงานที่เขียนด้วยภาษาอื่น

ปัญหาของการเขียนส่วนควบคุมที่แยกออกมาจากตัวโปรแกรม ก็คือ จะต้องมีวิธีการดึงเอาส่วนควบคุมนี้เข้าไปรวมกับโปรแกรมก่อนการใช้งานทุกครั้ง ดังนั้น โครงสร้างหลักของส่วนควบคุมซึ่งประกอบด้วยส่วนของการทำงานต่าง ๆ (Procedures) จะต้องมีส่วนที่จะเชื่อมโยงกับโปรแกรมหลัก เพื่อให้โปรแกรมซึ่งจะนำส่วนควบคุมนี้ไปใช้งาน สามารถหาตำแหน่งของส่วนการทำงานที่ต้องการได้ โดยส่วนเชื่อมโยงนี้จะนำตำแหน่งของส่วนการทำงานทั้งหมดมารวมไว้ในส่วนแรกสุดของส่วนควบคุม (Header) ดังนี้

```
ORG         0008H                ;ตำแหน่ง offset ของ driver
HEADER      PROC
        DW  ON_GRAPH             ;ตำแหน่ง offset ของส่วนการ
        DW  OFF_GRAPH            ;ทำงานที่เกี่ยวกับการแสดงภาพ
```

```
        DW      CLEAR_SCREEN
        DW      PUT_POINT
        DW      GET_POINT
        DW      PUT_LINE
        DW      GET_LINE
        DW      FILL_LINE
        DW      SET_PALETTE
        DW      GET_PALETTE
        DW      SCREEN_WIDTH
        DW      SCREEN_DEPTH
HEADER  ENDP
```

ข้อมูลตัวแรกของส่วนเชื่อมโยงซึ่งเริ่มต้นที่ตำแหน่ง 0008 จะเป็นค่าชี้ตำแหน่งออฟเซ็ทของส่วนการทำงาน ON_GRAPH ในขณะที่ค่าชี้ตำแหน่งของส่วนการทำงาน OFF_GRAPH ก็จะอยู่ที่ตำแหน่งถัดไปคือ 000Ah ค่าเหล่านี้จะถูกนำไปใส่ให้กับตัวแปรชี้ตำแหน่งส่วนการทำงาน (function pointer) ต่อไปนี้

```
int   far (*On_Graph)    (int type. int mode);
void far (*Off_Graph)    (void);
void far (*Clear_Screen)(int color);
void far (*Put_Point)    (int x, int y, int color, int mode);
int   far (*Get_Point)    (int x, int y);
void far (*Put_Line)     (char *p, int x, int y, int n);
void far (*Get_Line)     (char *p, int x, int y, int n);
void far (*Fill_Line)     (int x, int y, int n, int color);
void far (*Set_Palette)   (char *p, int s, int n);
void far (*Get_Palette)   (char *p, int s, int n);
int   far (*Screen_Width)(void);
int   far (*Screen_Depth)(void);
```

ค่าตำแหน่งดังกล่าวเป็นเพียงค่าออฟเซ็ท แต่การที่จะเรียกใช้งานได้จะต้องมีค่าเซ็กเมนต์ของส่วนการทำงานเหล่านั้นด้วย ก่อนจะพูดการได้มาของค่าเซ็กเมนต์นี้จะต้องทราบถึงขั้นตอนการนำส่วนควบคุมมาใช้งาน ซึ่งแบ่งได้เป็น 3 ขั้นตอน ดังนี้

1. จองเนื้อที่หน่วยความจำสำหรับส่วนควบคุม
2. นำส่วนควบคุมมาใส่ในเนื้อที่ซึ่งจองไว้
3. หาตำแหน่งของส่วนการทำงานแต่ละอัน

ในส่วนของการจองเนื้อที่หน่วยความจำ เนื่องจาก คำสั่งสำหรับจองเนื้อที่ (malloc) มีความแตกต่างกันระหว่าง Turbo C ซึ่งกำหนดให้ตำแหน่ง offset ของเนื้อที่มีค่าเท่ากับ 8 ในขณะที่ Borland C จะใช้ค่า offset เท่ากับ 4 ดังนั้น จึงต้องบังคับให้ใช้ค่า offset เท่ากับ 8 เป็นตำแหน่งเริ่มต้นของส่วนควบคุม แล้วกำหนดตำแหน่งเริ่มต้นของส่วนควบคุมเป็น 8 เพื่อบังคับให้ค่าตำแหน่งที่อ้างอิงภายในส่วนควบคุมทั้งหมดตรงกับตำแหน่งจริง ดังนี้

```
screen = (unsigned *) malloc (n+8);      // จองเนื้อที่
segment = (long) screen >> 16;           // ค่าเซ็กเมนต์ของเนื้อที่
screen = (unsigned *) MK_FP(sgm,8);      // บังคับให้ค่าออฟเซ็ทเท่ากับ 8
```

เมื่อดึงเอาส่วนควบคุมมาใส่ในเนื้อที่นี้แล้ว ตัวแปร screen จะชี้ไปที่ส่วนของค่าตำแหน่งออฟเซ็ท ในส่วนควบคุมที่กล่าวข้างต้น จากนั้นก็จะเป็นนำค่าตำแหน่งออฟเซ็ทของส่วนการทำงานแต่ละอันมารวมกับค่าเซ็กเมนต์ มากำหนดให้กับตัวแปรชี้ตำแหน่งฟังก์ชัน เริ่มจาก

```
On_Graph = (int far(*)( )) MK_FP (segment, screen[0])
```

หลังจากนั้นก็สามารถจะการเรียกใช้ส่วนการทำงานเหล่านี้ได้เหมือนฟังก์ชันทั่วไป ตัวอย่างเช่น On_Graph (3,0x62) เป็นคำสั่งให้เข้าสู่การแสดงผลด้วยภาพที่มีความละเอียด 1024x768 จุด หมายเลขแบบการแสดงภาพสำหรับแผงควบคุมของ TRIDENT ก็คือ 62h ส่วนค่า 3 เป็นเลขกำหนดความละเอียดของภาพ ซึ่งมีค่าเป็น 1, 2 หรือ 3 สำหรับการแสดงภาพขนาด 640 x480 จุด 800x600 จุด และ 1024x768 จุด ตามลำดับ

แต่ในกรณีที่ใช้ส่วนการแสดงภาพของ BIOS มาช่วยในการแสดงจุดภาพ และส่วนการทำงานใน BIOS ของแผงอุปกรณ์จอภาพส่วนใหญ่ จะรองรับการแสดงภาพที่มีความละเอียดของภาพทั้งสามระดับดังกล่าว ทำให้สามารถใช้ส่วนควบคุมร่วมกันได้เป็นส่วนใหญ่ ดังนั้นถ้า

ไม่มีปัญหาเรื่องความเร็วแล้ว ก็สามารถจะเขียนส่วนควบคุมรวมอยู่ในโปรแกรมหลักโดยใช้ภาษาซีได้ จากการทดลองใช้ส่วนควบคุม DRIVERS.C ซึ่งเขียนโดยใช้ภาษาซี ให้แสดงเส้นตรง 5000 เส้น จะใช้เวลาประมาณ 7 วินาที ซึ่งจะช้ากว่าส่วนการทำงานที่เขียนโดยภาษาแอสเซมบลี แต่ก็ยังเร็วกว่าการใช้ BIOS มาก ในตารางที่ 3 เป็นรายชื่อของส่วนการทำงานพื้นฐานที่จำเป็นต้องใช้ในระบบแสดงภาพ

ตารางที่ 3

ส่วนควบคุมการทำงานของจอภาพ

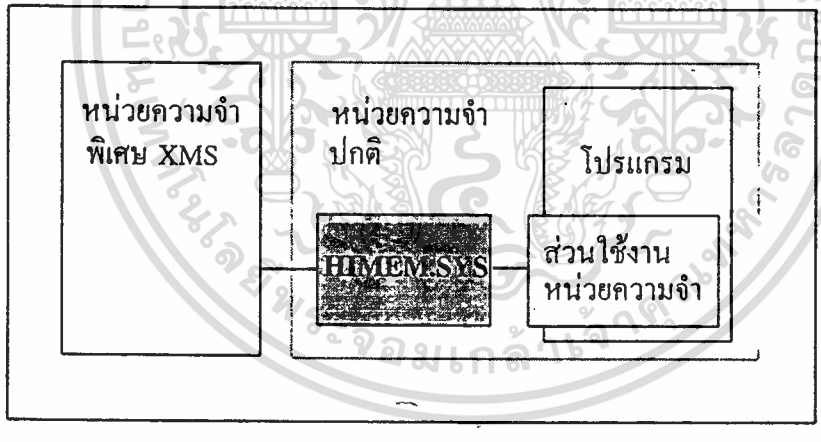
On_Graph (type,mode)	เข้าสู่ระบบภาพ
Off_Graph ( )	ออกจากระบบภาพ
Clear_Screen (color)	ลบจอภาพด้วยสี color
Put_Point (x,y,color)	แสดงจุดสี color ที่ตำแหน่ง x,y
Get_Point (x,y)	อ่านข้อมูลสีที่ตำแหน่ง x,y
Put_Line (p,x,y,n)	นำข้อมูลจาก p จำนวน n ไบท์ไปใส่ที่ตำแหน่ง (x,y)
Get_Line (p,x,y,n)	อ่านข้อมูลจากตำแหน่ง (x,y) จำนวน n ไบท์มาให้ p
Fill_Line (x,y,n,color)	ตีเส้นในแนวนอนยาว n จุด เริ่มจากตำแหน่ง (x,y)
Set_Palette (p,s,n)	นำข้อมูลสี n สีจาก p ไปใส่ในตารางสี เริ่มจากสี s
Get_Palette (p,s,n)	อ่านข้อมูลสี n สีจากตารางสีมาไว้ที่ p เริ่มจากสี s
Screen_Width ( )	จำนวนจุดภาพในแนวนอน
Screen_Depth ( )	จำนวนจุดภาพในแนวตั้ง
	(p คือ ค่าชี้ตำแหน่งข้อมูล char*)

ผู้เขียนโปรแกรมจะต้องเลือกระหว่างความเร็วในการทำงาน กับความสะดวกในการใช้งาน และการดัดแปลงส่วนควบคุมให้เข้ากับระบบที่เปลี่ยนไป โดยพิจารณาจากลักษณะของงานและความต้องการของผู้ใช้งาน

ส่วนการใช้หน่วยความจำพิเศษ

บางครั้งโปรแกรมที่เกี่ยวกับการสร้างภาพ ก็มีความจำเป็นต้องใช้หน่วยความจำในระหว่าง การทำงานจำนวนมาก ตัวอย่างเช่น โปรแกรมแสดงภาพที่มีการนำภาพมาไว้ในหน่วยความจำ ก่อนที่ส่งออกไปที่จอภาพ ต้องใช้หน่วยความจำถึง 300K สำหรับภาพที่มีขนาด 640x480 จุด 256 สี เมื่อรวมกับเนื้อที่ซึ่งส่วนการทำงานอื่น ๆ ต้องใช้แล้ว อาจจะมากกว่าที่ระบบจะจัดให้ได้จากหน่วยความจำปกติ (Conventional Memory) ซึ่งมีอยู่เพียง 640K เท่านั้น หรือในกรณีของโปรแกรมสร้างภาพ ซึ่งต้องใช้หน่วยความจำสำหรับการเก็บข้อมูลจำนวนมาก จำเป็นต้องนำหน่วยความจำพิเศษอื่น ๆ มาใช้เพิ่มเติมจากหน่วยความจำปกติ

ภาพที่ 10
การใช้หน่วยความจำ XMS



หน่วยความจำที่จะพูดถึงก็คือ หน่วยความจำ XMS (Extended Memory Specification) ซึ่งเป็นหน่วยความจำที่อยู่ในตำแหน่งสูงกว่า 1 Mb แต่ไม่สามารถอ้างถึงโดยใช้เลขตำแหน่งเหมือนหน่วยความจำปกติ และการใช้เนื้อที่ของหน่วยความจำ XMS ไม่มีส่วนการทำงานพื้นฐานของระบบรองรับอยู่ แต่จะมีส่วนควบคุมการใช้อุปกรณ์ (Device Driver) ชื่อ HIMEM.SYS ทำหน้าที่ควบคุมการติดต่อระหว่างหน่วยความจำส่วนนี้กับโปรแกรมทั่วไป ดังภาพที่ 10 ในที่นี้จะแสดงให้เห็นขั้นตอนต่าง ๆ ในการเรียกใช้ส่วนควบคุมนี้ เพื่อที่จะสามารถนำไปใช้ในงานต่าง ๆ ได้ อย่างถูกต้อง เนื่องจากภาษาซีไม่มีฟังก์ชันสำหรับเรียกใช้ส่วนควบคุมหน่วยความจำ XMS โดยตรง จึงต้องเขียนส่วนการทำงานขึ้นมาเพื่อการนี้โดยเฉพาะ

ส่วนควบคุมนี้จะถูกดึงเข้ามาไว้ในระบบโดยการกำหนดไว้ในแฟ้ม CONFIG.SYS การติดต่อกับส่วนควบคุม HIMEM.SYS เริ่มจากการใช้ Interrupt 2Fh หาตำแหน่งของส่วนควบคุม ดังนี้

```
reg.h.ah = 0x43;                          //เลขติดต่อกับ HIMEM.SYS
reg.h.al = 0x10;                          //คำสั่ง
int86x (0x2F, &reg, &reg, &sreg);         //เรียก interrupt 2Fh
Xms_Call = MK_FP (sreg.es, reg.x.bx);     //ตำแหน่งของส่วนควบคุม
```

ตัว Interrupt 2Fh ซึ่งเป็น Multiplex Interrupt [4] หรือส่วนการทำงานที่ถูกใช้ในการเชื่อมโยงกับโปรแกรมหรือส่วนควบคุมต่าง ๆ มากกว่าหนึ่งโปรแกรม แต่ละโปรแกรมก็จะมีหมายเลขสำหรับการติดต่อโดยเฉพาะ โดยส่วนควบคุมหน่วยความจำ XMS ใช้หมายเลข 43h

ตำแหน่งของส่วนควบคุมถูกนำมากำหนดเป็นค่าของตัวแปร Xms_Call ซึ่งเป็นตัวแปรชี้ตำแหน่งฟังก์ชัน เพื่อใช้ติดต่อกับส่วนควบคุม ตัวอย่างเช่น

```
_AH = 9;                                  //คำสั่ง
_DX = size;                               //ขนาดหน่วยความจำที่ต้องการเป็นกิโลไบท์
Xms_Call ( );                             //เรียกส่วนควบคุม
Xhandle = _DX;                            //หมายเลขประจำเนื้อที่
```

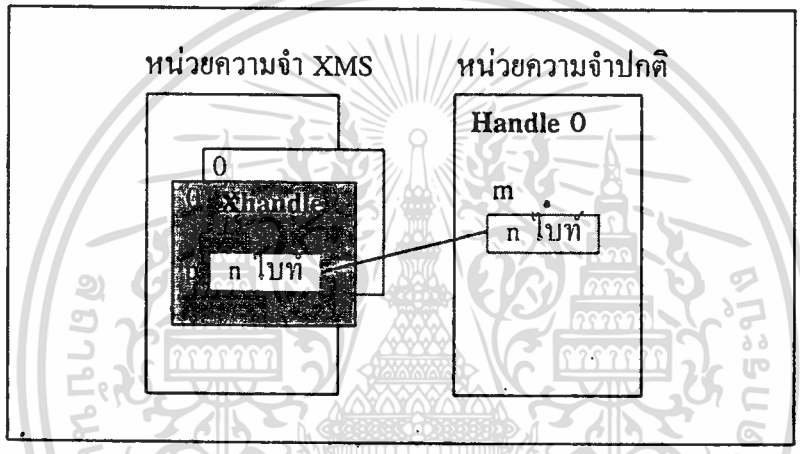
ส่วนควบคุมจะจัดเนื้อที่ให้พร้อมกับหมายเลข (Handle) ที่จะใช้ในการติดต่อเพื่ออ่านหรือบันทึกข้อมูล ในการอ่านและบันทึกข้อมูลในหน่วยความจำ XMS ต้องมีข้อมูลควบคุมการส่งข้อมูลซึ่งกำหนดที่มาที่ไปและจำนวนข้อมูลที่ต้องการ โดยมีลักษณะเป็นข้อมูลโครงสร้างดังนี้

```
struct  Xmove {                           // โครงสร้างกำกับการย้ายข้อมูล
        unsigned long   Length;           // จำนวนไบท์
        unsigned int    Shandle;          // หมายเลขหน่วยความจำต้นทาง
        unsigned long   Saddress;         // ตำแหน่งต้นทาง
        unsigned int    Dhandle;          // หมายเลขหน่วยความจำปลายทาง
        unsigned long   Daddress;         // ตำแหน่งปลายทาง
} Xm;
```

หมายเลขของหน่วยความจำ XMS คือ ค่าตัวเลขกำกับเนื้อที่ซึ่งได้มาจากการจองเนื้อที่ในแต่ละครั้งนั่นเอง แต่ถ้าเป็นการส่งข้อมูลไปยังหน่วยความจำปกติ ดังภาพที่ 11 จะใช้หมายเลขของหน่วยความจำปกติเท่ากับ 0

ภาพที่ 11

การส่งข้อมูลระหว่างหน่วยความจำ XMS กับหน่วยความจำปกติ



สำหรับตำแหน่งของข้อมูลในหน่วยความจำ XMS จะนับเป็นตำแหน่งไบท์จากจุดเริ่มต้นของเนื้อที่ซึ่งจองไว้ โดยให้จุดเริ่มต้นของเนื้อที่แต่ละส่วนเป็นตำแหน่ง 0 เมื่อกำหนดค่าควบคุมการส่งข้อมูลระหว่างหน่วยความจำครบถ้วนแล้ว ก็จะส่งค่าควบคุมเหล่านั้นให้กับส่วนควบคุมเพื่อทำการเคลื่อนย้ายข้อมูล ดังนี้

```
_SI = FP_OFF (&Xm);                    // ตำแหน่งของข้อมูลควบคุม
_DS = FP_SEG (&Xm);
_AH = 0x0B;                            // คำสั่งเคลื่อนย้ายข้อมูล
Xms_Call ();
```

ข้อควรระวังในการใช้หน่วยความจำ XMS ก็คือ ถ้าไม่บอกยกเลิกการใช้เนื้อที่หน่วยความจำส่วนที่จองไว้ เนื้อที่ส่วนนั้นก็จะไม่ถูกปลดปล่อยหลังจากจบการทำงานของโปรแกรม เหมือนกับหน่วยความจำปกติ ทำให้ไม่สามารถนำกลับมาใช้ได้อีกจนกว่าจะเปิดเครื่องใหม่ ใน ตารางที่ 4 เป็นรายชื่อของส่วนการทำงานที่เกี่ยวข้องกับการใช้หน่วยความจำ XMS จะเห็นว่า

การกำหนดตำแหน่งข้อมูลจะใช้เลขลำดับของชุดข้อมูล แทนการกำหนดเป็นตำแหน่งไบท์ ทั้งนี้เพราะว่า ส่วนใหญ่แล้วจะใช้ในการเก็บข้อมูลที่มีลักษณะเป็นชุด

ตารางที่ 4

ส่วนการทำงานสำหรับการใช้หน่วยความจำ XMS

Xms_Alloc (n,s)	จองเนื้อที่จำนวน n ชุด ๆ ละ s ไบท์
Xms_Free (handle)	ยกเลิกการใช้เนื้อที่
Xms_Write (handle, n, s, m)	บันทึกข้อมูลชุดที่ n ขนาด s ไบท์
Xms_Read (handle, n, s, m)	อ่านข้อมูลชุดที่ n ขนาด s ไบท์

หมายเหตุ handle หมายเลขหน่วยความจำที่ได้จากฟังก์ชัน Xms_Alloc ( )
m ตำแหน่งข้อมูลในหน่วยความจำปกติ

ตัวอย่างของการนำหน่วยความจำ XMS มาใช้งานก็คือ ใช้เป็นที่เก็บสำรองภาพก่อนที่จะส่งภาพนั้นออกไปแสดงที่จอภาพ ทำให้สามารถแสดงภาพเดิมซ้ำกันได้หลายครั้ง หรือเลือกแสดงส่วนต่าง ๆ ของภาพได้อย่างรวดเร็ว เป็นการตัดปัญหาเรื่องการใช้หน่วยความจำปกติ โดยเฉพาะในกรณีที่ภาพมีขนาดใหญ่ จากการทดสอบใช้หน่วยความจำ XMS เก็บข้อมูลภาพขนาด 200x150 จุด แล้วเรียกภาพนั้นมาแสดงบนจอภาพโดยอ่านข้อมูลมาทีละเส้นภาพ (ภาคผนวก ข) โดยกำหนดให้แสดงภาพนี้ จำนวน 500 ครั้ง ซึ่งจะเท่ากับการอ่านชุดข้อมูลขนาด 200 ไบท์ ทั้งหมด 75,000 ครั้ง เมื่อทดสอบบนเครื่อง 486DX2-66 ใช้เวลาทั้งสิ้น 21 วินาที ในขณะที่การเก็บภาพในหน่วยความจำปกติจะใช้เวลา 19 วินาที ซึ่งก็แตกต่างกันไม่มากนัก

การที่หน่วยความจำ XMS เป็นหน่วยความจำที่มีเนื้อที่มาก มีการรับส่งข้อมูลที่ค่อนข้างเร็ว จึงสามารถนำไปใช้ในงานต่าง ๆ ได้มาก แต่โดยทั่วไปแล้ว จะมีการพูดถึงรายละเอียดในเรื่องนี้น้อยมาก จึงได้นำมาอธิบายในที่นี้ เพื่อที่จะสามารถนำไปดัดแปลงใช้ในงานที่ต้องการได้อย่างถูกต้อง

บทที่ 4
ส่วนแสดงภาพ

การแสดงภาพบนจอภาพ

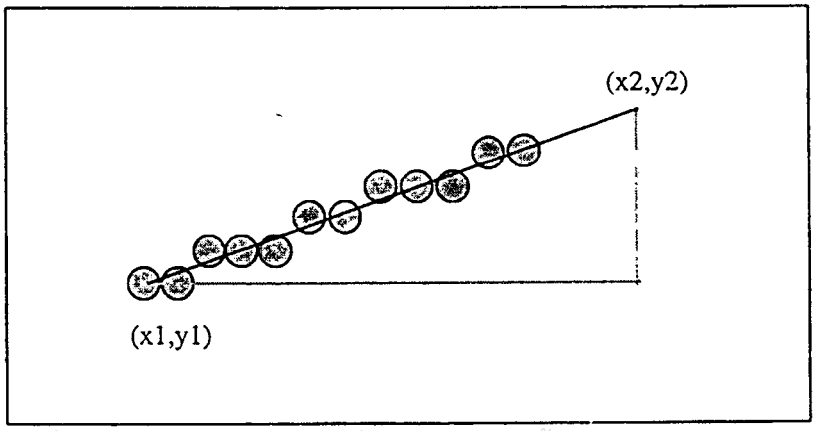
จากการกำหนดให้โครงสร้างของระบบแสดงภาพในที่นี้ มีองค์ประกอบทั้งหมดของภาพถูกสร้างขึ้นมาจากเส้นตรง ส่วนการทำงานพื้นฐานที่สำคัญของระบบแสดงภาพ ก็คือ ส่วนการลากเส้นตรง ดังนี้

Draw_Line (x1, y1, x2, y2, color)

ค่า (x1,y1) และ (x2,y2) เป็นค่าบอกตำแหน่งจุดปลายทั้งสองของเส้นตรง และไม่ว่าข้อมูลภาพจะเป็นระบบใดก็ตาม ท้ายที่สุดแล้ว ข้อมูลตำแหน่งจะถูกเปลี่ยนมาเป็นตำแหน่งจริงของจุดบนจอภาพ (screen coordinate) เพื่อที่จะแสดงภาพนั้นบนจอภาพ โดยเส้นตรงที่ปรากฏบนจอภาพจะเกิดจากจุดที่เรียงต่อกัน

ภาพที่ 12

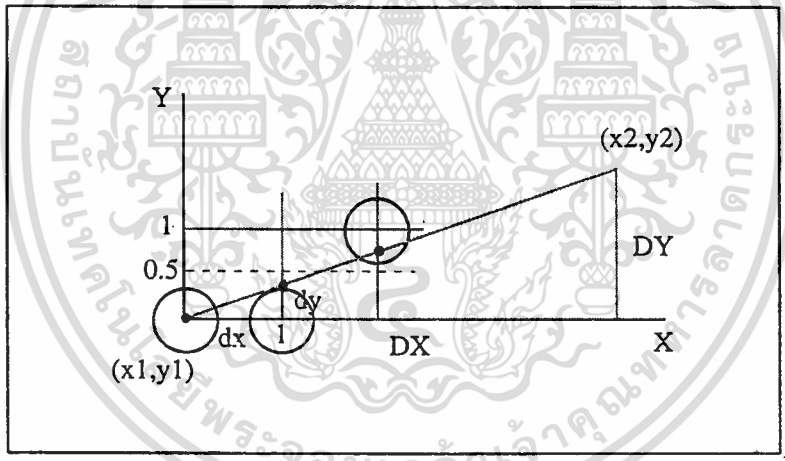
การเรียงจุดเพื่อสร้างเส้นตรงระหว่างจุดสองจุด



ในภาพที่ 12 จะเห็นว่า มีการขยับตำแหน่งในแกนนอนซึ่งเป็นแกนที่ระยะระหว่างจุดปลายยาวกว่าไปเรื่อย ๆ ทีละหนึ่งจุด ส่วนตำแหน่งทางแกนตั้งจะขยับขึ้นตามความลาดชันของเส้นตรง หลักการที่ใช้ในการตัดสินใจว่า จะขยับตำแหน่งจุดขึ้นไปหรือจะอยู่ในแนวเดิม . เพื่อให้ใกล้เคียงกับเส้นตรงที่ต้องการมากที่สุดนั้น ดัดแปลงมาจากวิธีการเทียบจุดตรงกลาง (Midpoint Algorithm) [5] ซึ่งจะนำระยะที่ควรจะเพิ่มขึ้นทางแกนที่สั้นกว่ามาเทียบกับระยะระหว่างจุด ถ้ายังไม่ถึงครึ่งหนึ่งของระยะระหว่างจุดก็ให้อยู่ในตำแหน่งเดิม แต่ถ้าเลยจุดกึ่งกลางแล้ว ก็จะขยับตำแหน่งในแกนนั้นไปหนึ่งจุด

ภาพที่ 13

การเลือกตำแหน่งจุดโดยเทียบกับจุดกึ่งกลาง



ในภาพที่ 13 DX ซึ่งเป็นระยะระหว่างจุดทางแกน X มีค่ามากกว่า DY ซึ่งเป็นระยะทางแกน Y ตำแหน่งทางแกน X จึงถูกเพิ่มขึ้นตลอด ถ้าให้ระยะที่เพิ่มขึ้นทางแกน X มีค่าเท่ากับ dx ระยะที่ควรจะเพิ่มขึ้นในแนวแกน Y ก็จะหาได้จากสมการที่ (1) ซึ่งเป็นสมการเส้นตรงที่มี DY/DX เป็นค่าความลาดชันของเส้นตรง ระยะที่ได้จะเป็นระยะบนเส้นตรงจริง แต่เนื่องจากการใช้หน่วยของระยะทางเป็นตำแหน่งจุด ค่าของ dx ก็จะเท่ากับ 1 ตำแหน่งจุด ดังนั้นค่า dy ที่มีหน่วยเป็นตำแหน่งจุดจะเป็นดังสมการที่ (2)

$$dy = dx * DY / DX \quad (1)$$

$$dy = DY / DX \quad (2)$$

เงื่อนไขสำหรับการขยับตำแหน่งในแนวแกน Y นั้น ดูจากค่า Acc (dy) ซึ่งเป็นฟังก์ชันสำหรับสะสมค่าระยะ dy ว่าเกินตำแหน่งกึ่งกลางของระยะระหว่างจุดหรือยัง เมื่อแทนค่า dy เงื่อนไขจะเป็น

Acc (DY/DX) > 1/2

เมื่อนำมาจัดรูปแบบใหม่ ก็จะได้เงื่อนไขในการขยับตำแหน่งดังนี้

1/2 + Acc (DY/DX) > 1

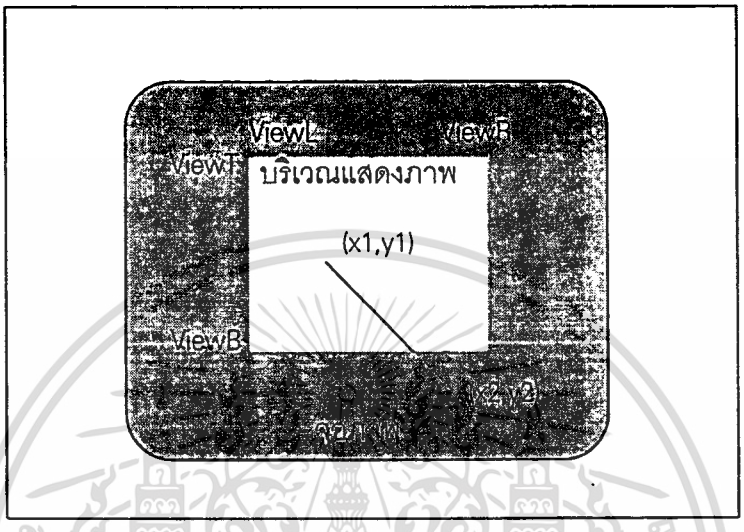
DX/2 + Acc (DY) > DX

จากหลักการดังกล่าวจะเขียนเป็นขั้นตอนการลากเส้นตรงดังนี้

```
acc = dx/2;                     // ค่าตรวจสอบเริ่มต้น
while (x1!=x2) {                // ถึงจุดที่สองหรือยัง
        Draw_Point (x1, y1, color);   // แสดงจุดภาพ
        acc += dy;              // สะสมค่าตรวจสอบ
        if (acc>dx) {           // เกินระยะครึ่งจุด
                y1 += ys;       // เปลี่ยนตำแหน่ง y
                acc -= dx;      // ปรับค่าตรวจสอบ
        }
        x1 += xs;               // เปลี่ยนค่า x
}
Draw_Point (x2, y2, color);     // แสดงจุดสุดท้าย
```

xs มีค่าเป็น 1 หรือ -1 ขึ้นอยู่กับทิศทางของเส้นตรง ในตัวอย่างข้างต้น กำหนดให้เปลี่ยนตำแหน่งในแกน X เป็นหลัก แต่ในความเป็นจริงแล้ว เส้นตรงแต่ละเส้นจะมีความลาดชันต่างกัน ดังนั้นเพื่อให้เกิดความต่อเนื่องของเส้นตรง จะต้องมีการตรวจระยะระหว่างจุดทางแกน X และแกน Y แล้วใช้แกนที่มีระยะยาวกว่าเป็นหลัก

ภาพที่ 14
การกำหนดพื้นที่สำหรับแสดงภาพ



ในการแสดงจุดภาพจะใช้ส่วนการทำงาน Draw_Point ( ) แทนการใช้ Put_Point ( ) โดยตรง เพื่อให้มีการตรวจสอบตำแหน่งที่ต้องการแสดงจุดภาพ ถ้าตำแหน่งนั้นอยู่ในขอบเขตที่กำหนดไว้ ก็จะเรียกส่วนการทำงาน PutPoint ( ) ให้แสดงจุดภาพนั้น การกำหนดขอบเขตการแสดงภาพจะใช้ส่วนการทำงาน Set_View () กำหนดให้ตัวแปร ViewL และ VIewR เก็บค่าตำแหน่งซ้ายสุดและขวาสุดของพื้นที่แสดงภาพ และ ViewT และ VIewB เก็บตำแหน่งบนสุดและล่างสุดดังภาพที่ 14 โดยตำแหน่งบนจอภาพจะเริ่มต้นนับจากตำแหน่ง (0,0) ที่มุมบนซ้ายของจอภาพ ตำแหน่งทางขวาสุดจะเท่ากับ Screen_Width ( ) - 1 และทางด้านล่างสุดจะเท่ากับ Screen_Depth ( ) - 1

ในตารางที่ 5 เป็นรายชื่อของส่วนการทำงานพื้นฐานที่จำเป็นต้องมีในส่วน GRAPH_SC สำหรับการแสดงภาพในระบบพิกัดของจอภาพ ส่วนการทำงานทั้งหมดนอกเหนือจากส่วนที่ใช้ติดต่อกับจอภาพจะเขียนโดยใช้ภาษาซี เพื่อให้ได้โปรแกรมที่สามารถจะนำไปปรับใช้กับระบบอื่นได้สะดวก

ตารางที่ 5

ส่วนการแสดงภาพในพิกัดจอภาพ

Draw_Point (x,y,color)	แสดงจุดสี color ที่ตำแหน่ง (x,y)
Draw_Line (x1,y1,x2,y2,color)	ลากเส้นตรงจาก (x1,y1) ไปยัง (x2,y2)
Set_View (left,top,right,bottom)	กำหนดพื้นที่การแสดงภาพ
Clear_View (color)	ลบพื้นที่แสดงภาพด้วยสี color
Init_Screen ( )	กำหนดให้แสดงภาพเต็มจอภาพ

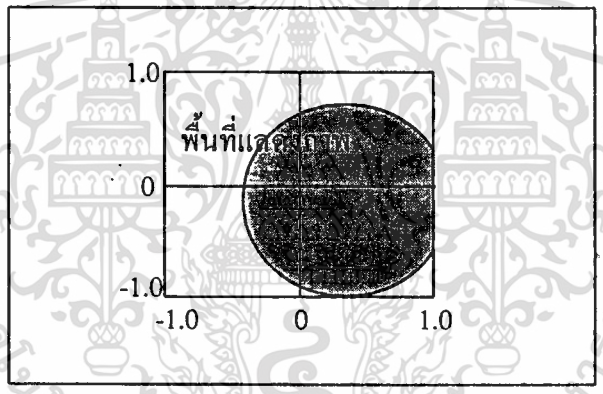
เนื่องจากในที่นี้ ได้กำหนดให้ใช้เส้นตรงเป็นองค์ประกอบพื้นฐานของระบบ จึงไม่ได้พูดถึงส่วนการทำงานสำหรับสร้างภาพในลักษณะอื่น เช่น การแสดงภาพวงกลม หรือ การแสดงตัวอักษร แต่ถ้าจะเพิ่มเติมส่วนการทำงานที่ใช้พิกัดจอภาพ ซึ่งมีการเรียกใช้ส่วนการทำงานในระดับเดียวกันหรือต่ำกว่าก็สามารถทำได้เลย แต่ถ้าเป็นการแก้ไขส่วนการทำงานเดิมในส่วนของโครงสร้างวิธีการเรียกใช้งาน หรือผลที่ได้จากการทำงาน ก็จะมีผลกระทบต่อส่วนการทำงานในระดับสูงขึ้นไป

การแสดงภาพในระบบสองมิติ

การแสดงภาพในแบบสองมิติ (2D-Graphics) จะเป็นการแสดงภาพในพื้นที่ของจอภาพ ซึ่งได้กำหนดไว้ในส่วนที่แล้ว แต่จะเปลี่ยนมาใช้ระบบพิกัดที่เป็นกลาง (Normalized Coordinate) เพื่อตัดปัญหาเรื่องหน่วยของตำแหน่งของจุดภาพ โดยในที่นี้ จะกำหนดให้แกนทั้งสองใช้พิกัดกลางที่มีระยะตั้งแต่ -1 ถึง 1 ดังภาพที่ 15

ภาพที่ 15

ขอบเขตการแสดงภาพในระบบพิกัดกลาง



ในการเรียกใช้ส่วนการทำงานก็จะใช้พิกัดกลางทั้งหมด แล้วปล่อยให้เป็นหน้าที่ของส่วนการทำงานนั้น ทำการเปลี่ยนค่าบอกตำแหน่งในพิกัดกลางให้เป็นตำแหน่งบนจอภาพ เพื่อเรียกใช้ส่วนการทำงานในพิกัดของจอภาพอีกต่อหนึ่ง ตำแหน่ง (x, y) ในพิกัดกลาง จะถูกเปลี่ยนให้เป็นตำแหน่ง (xs, ys) ในพิกัดจอภาพ ตามสมการที่ (3) และ (4)

$$xs = CenterX + x*MapX \quad (3)$$

$$ys = CenterY - y*MapY \quad (4)$$

โดยที่ (CenterX, CenterY) ก็คือจุดศูนย์กลางของพื้นที่แสดงภาพ หรือตำแหน่ง (0,0) ในพิกัดกลาง จะมีค่าเท่ากับ

$$CenterX = (ViewL + ViewR) / 2$$

$$CenterY = (ViewT + ViewB) / 2$$

ส่วนค่า MapX และ MapY เป็นอัตราส่วนระหว่างขนาดในพิกัดของจอภาพกับขนาดในพิกัดกลาง ถ้ากำหนดให้ค่าตำแหน่ง -1 และ 1 บนแกน X ในระบบพิกัดกลาง ตรงกับตำแหน่ง ViewL และ ViewR บนจอภาพ ในขณะที่ ตำแหน่ง -1 และ 1 บนแกน Y ตรงกับตำแหน่ง ViewB และ ViewT ตามลำดับ อัตราส่วนระหว่างระยะในพิกัดจอภาพและพิกัดกลางจะเท่ากับ

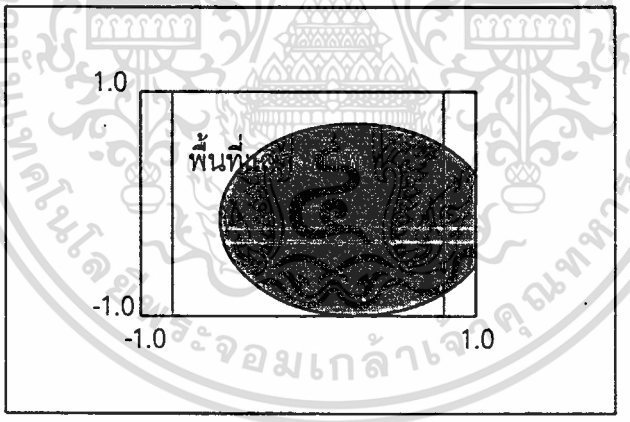
MapX = (ViewR - ViewL) / 2.0

MapY = (ViewB - ViewT) / 2.0

แต่ถ้ามีการกำหนดความกว้างและความยาวของพื้นที่แสดงภาพไม่เท่ากัน ก็จะทำให้ภาพมีสัดส่วนผิดไปดังตัวอย่างในภาพที่ 16

ภาพที่ 16

การกำหนดขอบเขตที่ไม่ได้สัดส่วน



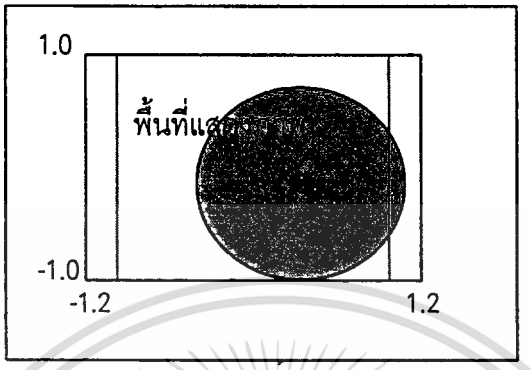
ถ้าต้องการรักษาสัดส่วนของภาพที่แสดงออกมาที่จอภาพ ให้ตรงกับความเป็นจริง โดยแสดงรูปวงกลมออกมาให้เห็นเป็นวงกลม และรูปสี่เหลี่ยมจตุรัสมีด้านทุกด้านเท่ากันจริง ๆ ก็จะต้องให้อัตราส่วนพิกัดของทั้งสองแกนมีค่าเท่ากับอัตราส่วนของระยะระหว่างจุดในแนวแกนทั้งสอง ด้วยการปรับขยายค่าขอบเขตในพิกัดกลางออกไป ดังภาพที่ 17 โดยกำหนดให้ขอบเขตในด้านที่สั้นกว่าอยู่ที่พิกัด -1 ถึง 1 เหมือนเดิม แล้วปล่อยให้ค่าขอบเขตของแกนที่ยาวกว่าขยายออกไป กรณีที่ขอบเขตในแนวแกน X สั้นกว่าขอบเขตในแนวแกน Y ค่าอัตราส่วนพิกัดจะเท่ากับ

MapY = (ViewB - ViewT) / 2.0

MapX = MapY

ภาพที่ 17

การปรับค่าขอบเขตเพื่อรักษาสัดส่วนของภาพ



ที่จริงแล้ว ค่าอัตราส่วนพิกัด MapX และ MapY ก็คือ จำนวนจุดต่อความยาวในพิกัดกลาง เมื่อกำหนดให้ใช้ค่าอัตราส่วนเท่ากันแล้ว ถ้าความยาวในพิกัดกลางของทั้งสองแกนมีค่าเท่ากันอย่างในกรณีของสี่เหลี่ยมจตุรัส จะได้ความยาวเป็นจำนวนจุดเท่ากัน ดังนั้น ถ้าระยะระหว่างจุดในแนวแกนทั้งสองเท่ากัน ระยะบนจอภาพก็จะเท่ากันด้วย แต่ถ้าระยะระหว่างจุดในแนวแกนทั้งสองไม่เท่ากัน ก็ต้องปรับค่าอัตราส่วนเพื่อให้ได้ระยะในแต่ละแกนเท่ากัน ดังนี้

MapX = MapY * ratio

โดย ratio เป็นค่าอัตราส่วนของระยะระหว่างจุดในแนวแกนทั้งสอง ดังสมการที่ (5)

ratio = dy / dx (5)

โดยที่ dx และ dy เป็นระยะระหว่างจุดในแนวแกน X และ Y ตามลำดับ ซึ่งมีค่าเท่ากับขนาดจอภาพหารด้วยจำนวนจุด ดังนี้

dx = ความกว้างจอภาพ / จำนวนจุดแนวนอน

dy = ความสูงจอภาพ / จำนวนจุดแนวตั้ง

เมื่อนำไปแทนค่าในสมการที่ (5) จะได้ค่าอัตราส่วนของระยะระหว่างจุดเท่ากับ

ratio = aspect * จำนวนจุดแนวนอน / จำนวนจุดแนวตั้ง (6)

โดยที่ aspect ก็คือ อัตราส่วนทางกายภาพ ซึ่งเป็นอัตราส่วนระหว่างความสูงกับความกว้างของจอภาพ จอภาพโดยทั่วไปจะมีค่าอัตราส่วนทางกายภาพเท่ากับ 3/4 ส่วนจำนวนจุดในแนวแกนทั้งสองนั้นจะได้จากฟังก์ชัน Screen_Width( ) และ Screen_Depth( ) เมื่อนำไปแทนค่าในสมการที่ (6) จะได้ค่า ratio เท่ากับ

$$ratio = 0.75 * Screen\_Width( ) / Screen\_Depth( ) \quad (7)$$

จากสมการที่ (7) ถ้าเลือกใช้แบบการแสดงภาพขนาด 640x480 จุด ค่าอัตราส่วนนี้ก็จะเท่ากับ

$$ratio = 0.75 * 640x\ 480 = 1$$

ในกรณีของภาพขนาด 800x600 จุด หรือ 1024x768 จุด ก็จะได้ค่าเท่ากับ 1 เช่นเดียวกัน นั่นก็หมายความว่า ระยะระหว่างจุดจะเท่ากันไม่ว่าจะเป็นแนวตั้งหรือแนวนอน

ตารางที่ 6

ส่วนการแสดงภาพในระบบพิกัดกลาง 2 มิติ

Draw_Point_2D (x,y,color)	แสดงจุดสี color ที่ตำแหน่ง (x,y)
Draw_Line_2D (x1,y1,x2,y2,color)	ลากเส้นตรงสี color จากตำแหน่ง (x1,y1) ไปยังตำแหน่ง (x2,y2)
Set_Window (L,T,R,B)	กำหนดพื้นที่แสดงภาพจากตำแหน่ง (L,T) ถึงตำแหน่ง (R,B)
Init_Window ( )	กำหนดพื้นที่แสดงภาพเต็มจอภาพ

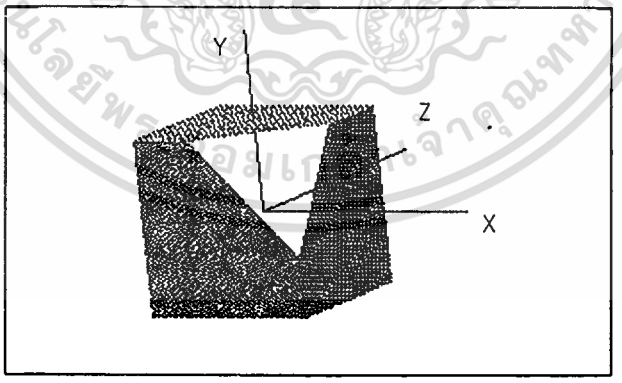
ในตารางที่ 6 เป็นรายชื่อของส่วนการทำงานพื้นฐานใน GRAPH_2D สำหรับการใช้งานในระบบสองมิติ นอกส่วนการทำงานสำหรับลากเส้นตรงซึ่งเป็นส่วนสำคัญแล้ว จะมีส่วนของการกำหนดพื้นที่แสดงภาพ Set_Window ( ) ซึ่งเรียกใช้ส่วนการทำงาน Set_View ( ) แต่จะมีการคำนวณค่า MapX และ MapY สำหรับการเปลี่ยนพิกัดกลางให้เป็นพิกัดจอภาพด้วย

การแสดงภาพในระบบสามมิติ

ขั้นตอนที่สำคัญที่สุดสำหรับระบบภาพแบบสามมิติ ก็คือ การคำนวณตำแหน่งข้อมูลในระบบพิกัดสามมิติ และเมื่อพูดถึงระบบสามมิติก็จะหมายถึง ระบบที่มีทั้งความกว้าง ความยาว และความลึก ซึ่งในที่นี้ได้กำหนดให้ใช้ระบบพื้นฐานซึ่งประกอบด้วยแกนสามแกนตั้งฉากกัน (Cartesian Coordinate) สำหรับระบบที่ใช้งานทางคณิตศาสตร์ จะกำหนดทิศทางของแกนโดยใช้ระบบมือขวา นั่นคือ ถ้าใช้มือขวากำรอบแกน Z โดยให้นิ้วทั้งสี่ชี้จากแกน X ไปยังแกน Y นิ้วหัวแม่มือก็จะชี้ไปตามแนวแกน Z เมื่อนำไปเทียบกับจอภาพ โดยให้แกน X ชี้จากซ้ายไปขวา และแกน Y ชี้จากล่างขึ้นบน แกน Z ก็จะชี้ออกจากจอภาพ แต่สำหรับผู้ใช้งานทั่วไปซึ่งไม่คุ้นเคยกับระบบทางคณิตศาสตร์ มักจะมีความรู้สึกว่า ระยะแกน Z ที่เดินหน้าเข้าไปในจอภาพน่าจะมีค่าเป็นบวก ดังนั้น เพื่อความสะดวกของผู้ใช้งาน จึงกำหนดให้ใช้ทิศทางของแกนในระบบมือซ้าย ซึ่งมีทิศทางของแกน Z ชี้เข้าไปในจอภาพ ดังภาพที่ 18

ภาพที่ 18

ระบบพิกัดสามมิติที่ใช้ในโปรแกรม



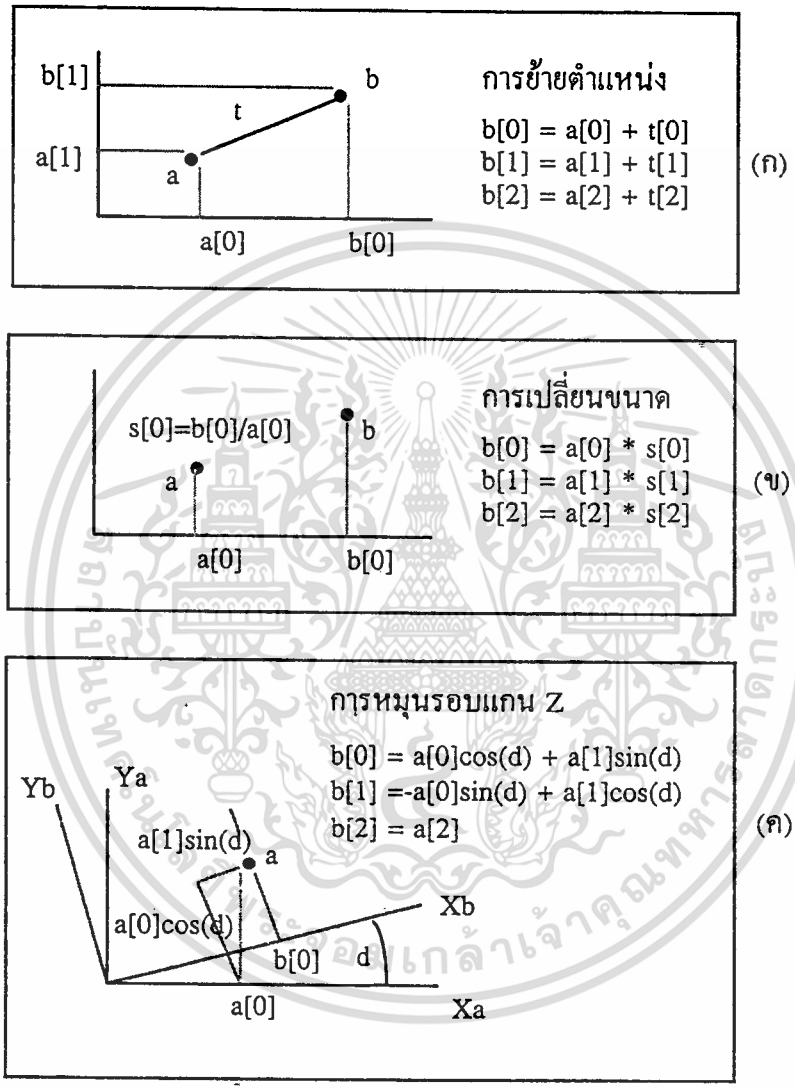
ส่วนการทำงานในระบบสามมิติ จะใช้ข้อมูลบอกตำแหน่งเป็นเว็คเตอร์ ตัวอย่างเช่น ส่วนการทำงานสำหรับลากเส้นตรงจะเป็นดังนี้

```
Draw_Line_3D (Vector p1, Vector p2, int color)
```

โดยที่ p1 และ p2 เป็นข้อมูลที่ถูกกำหนดให้อยู่ในรูปของเว็คเตอร์ ซึ่งประกอบด้วยค่าตำแหน่ง

ภาพที่ 25

การเปลี่ยนพิกัดของข้อมูลบอกตำแหน่ง



ในทำนองเดียวกัน การเปลี่ยนขนาดพิกัด ในภาพที่ 25 (ข) เมื่อกำหนดให้ s เป็นค่าเวคเตอร์กำหนดอัตราส่วนการเปลี่ยนขนาด แมตริกซ์สำหรับการเปลี่ยนขนาดจะเป็นดังนี้

$$Ms = \begin{bmatrix} s[0] & 0 & 0 & 0 \\ 0 & s[1] & 0 & 0 \\ 0 & 0 & s[2] & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

สำหรับการหมุนรอบแกน ดูจะยุ่งยากกว่าเล็กน้อย จากตัวอย่างในภาพที่ 25 (ค) เป็นการหมุนรอบแกน Z แกนเดียว ซึ่งจะได้ค่าแมตริกซ์สำหรับการหมุน ดังนี้

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

ค่าแมตริกซ์สำหรับการหมุนรอบแกน X จะเท่ากับ

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

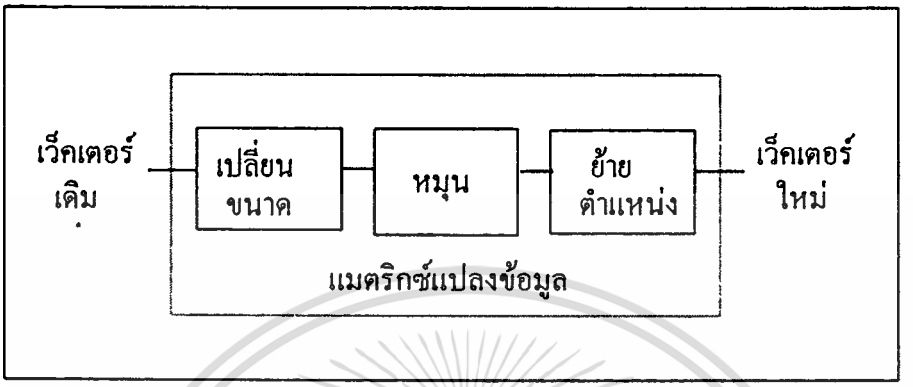
ค่าแมตริกซ์สำหรับการหมุนรอบแกน Y จะเท่ากับ

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

การเปลี่ยนพิกัดทั้งสามแบบได้แก่ การเปลี่ยนขนาด การหมุน และการย้ายตำแหน่งเป็นกระบวนการต่อเนื่องที่สามารถนำมารวมอยู่ในแมตริกซ์เดียวกันได้ โดยการนำแมตริกซ์ที่เกิดจากการเปลี่ยนแต่ละขั้นตอนมาคูณกันตามลำดับ เริ่มจากการเปลี่ยนขนาด แล้วหมุนรอบแกนทั้งสาม จากนั้นจึงจะเคลื่อนย้ายไปที่ตำแหน่งใหม่ ดังภาพที่ 26

ภาพที่ 26

การแปลงข้อมูลตำแหน่ง



ในกรณีที่ต้องการจะได้ค่าเวิ็คเตอร์เดิมกลับมา ก็สามารถนำมาผ่านกระบวนการถอยหลังกลับได้ เริ่มจากการเคลื่อนย้ายตำแหน่งกลับในทิศทางตรงกันข้าม ดังนี้

$$Mt = \begin{bmatrix} 1 & 0 & 0 & -t[0] \\ 0 & 1 & 0 & -t[1] \\ 0 & 0 & 1 & -t[2] \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

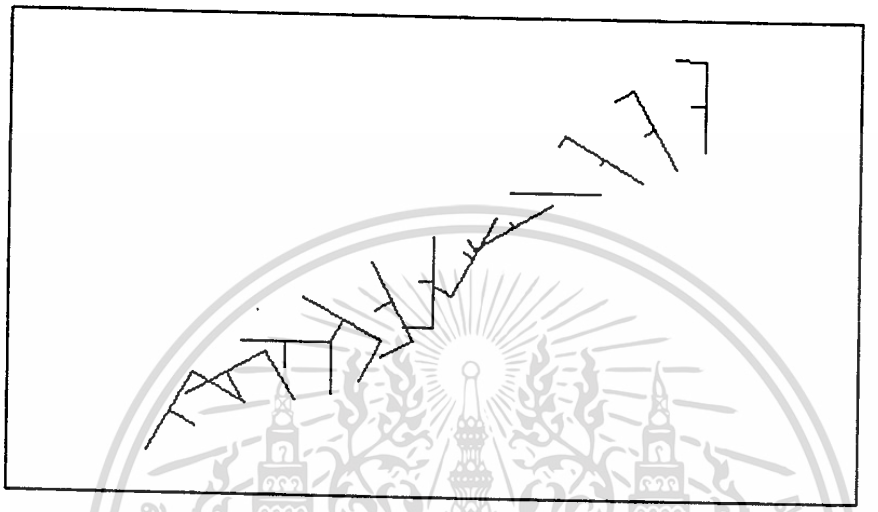
จากนั้นก็หมุนด้วยมุมย้อนกลับด้วยการให้ค่ามุมเป็นลบ แล้วปิดท้ายด้วยการเปลี่ยนขนาด ดังนี้

$$Ms = \begin{bmatrix} 1/s[0] & 0 & 0 & 0 \\ 0 & 1/s[1] & 0 & 0 \\ 0 & 0 & 1/s[2] & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ตารางที่ 9 เป็นรายชื่อของส่วนการทำงานที่เกี่ยวกับการแปลงข้อมูลบอกตำแหน่งทั้งหมด และในภาพที่ 27 เป็นตัวอย่างของการใช้ส่วนการแสดงภาพในระบบสามมิติ ร่วมกับการแปลงตำแหน่งข้อมูล โดยการหมุนตัวอักษร 'F' รอบแกน X และ แกน Z พร้อมกับการเคลื่อนย้ายตำแหน่งในแนวแกน X และแกน Y

ภาพที่ 27

ภาพที่ได้จากการหมุนและเคลื่อนย้ายวัตถุภาพ



ตารางที่ 9

ส่วนการทำงานสำหรับการเปลี่ยนพิกัด

Translate_3D (t, mat)	สร้างแมตริกซ์ย้ายตำแหน่งด้วยเว็คเตอร์ t
Scale_3D (s, mat)	สร้างแมตริกซ์เปลี่ยนขนาดด้วยเว็คเตอร์ s
Rotate_3D (axis, theta, mat)	สร้างแมตริกซ์หมุนรอบแกน axis ด้วยมุม theta
Set_Tmatx (s, t, r, mat)	สร้างแมตริกซ์รวมสำหรับการเปลี่ยนพิกัด
Inv_Tmatx (s, t, r, mat)	สร้างแมตริกซ์รวมสำหรับการเปลี่ยนย้อนกลับ
Transform (a, b, mat)	เปลี่ยนพิกัดจากเว็คเตอร์ a เป็นเว็คเตอร์ b โดยใช้แมตริกซ์ mat

หมายเหตุ mat เป็นแมตริกซ์ขนาด 4x4

ส่วนการทำงานต่อไปนี้ แสดงการสร้างแมตริกซ์สำหรับเปลี่ยนพิกัด ที่รวมเอาการเปลี่ยนตำแหน่ง และการหมุนรอบแกนไว้ด้วยกัน แล้วนำแมตริกซ์นั้นไปใช้ในการเปลี่ยนข้อมูลตำแหน่ง ต่าง ๆ ดังนี้

```
#include "graph_a.h"

Vector  p[5] = { -0.1, -0.15,  0.0,          // ข้อมูลตำแหน่งจุด
                 -0.1,  0.15,  0.0,
                  0.1,  0.15,  0.0,
                 -0.1,  0.0,   0.0,
                  0.0,  0.0,   0.0 };
Tmatx  tx;                                   // สร้างแมตริกซ์เปลี่ยนพิกัด

void Draw_Object ( )                         // แสดงภาพตัวอักษร F
{
        Vector m[5];
        int i;
        for (i=0; i<5; i++)
                Transform (p[i], tx, m[i]);  // เปลี่ยนพิกัด
        Draw_Line_3D (m[0], m[1], 0);// ลากเส้น
        Draw_Line_3D (m[1], m[2], 0);
        Draw_Line_3D (m[3], m[4], 0);
}

void main()
{
        Vector  rot, trn, sca;
        int i;
```

```
if (On_Graph(1,0x5D)) return;          // เข้าระบบภาพ
Init_Window ();                        // เตรียมพื้นที่
Init_3D ();                            // เตรียมการแสดงภาพสามมิติ
Do_Vector (1, 1, 1, sca);              // เว็คเตอร์ขนาด
Do_Vector (0, 0, 0, rot);              // เว็คเตอร์หมุน
Do_Vector (-0.8, -0.4, 0, trn);        // เว็คเตอร์เปลี่ยนตำแหน่ง
for (i=0; i<12; i++) {
        rot[1] += 10;                  // หมุนรอบแกน Y
        rot[2] += 30;                  // หมุนรอบแกน Z
        trn[0] += 0.15;                // เลื่อนตำแหน่งบนแกน X
        trn[1] += 0.1;                 // เลื่อนตำแหน่งบนแกน Y
        Set_Tmatx (sca,trn,rot,tx);    // สร้างแมตริกซ์เปลี่ยนพิกัด
        Draw_Object ();                // แสดงภาพ
}
getch();
Off_Graph ();                          // ออกจากระบบภาพ
}
```

ในตัวอย่างนี้เป็นการใช้คำสั่งพื้นฐานต่าง ๆ ไม่ว่าจะเป็นการลากเส้น หรือการเปลี่ยนพิกัดของตำแหน่งภาพ ให้แสดงภาพที่ต้องการโดยตรง แต่ในบทต่อไป จะแสดงให้เห็นวิธีการเก็บข้อมูลภาพไว้ ก่อนที่จะนำข้อมูลเหล่านั้นมาเปลี่ยนให้เป็นคำสั่งสร้างภาพในภายหลัง

บทที่ 5
ส่วนสร้างภาพ

คำสั่งสร้างองค์ประกอบภาพ

ในส่วนของการสร้างองค์ประกอบภาพ ข้อมูลกำหนดลักษณะภาพจากส่วนใช้งานจะถูกเปลี่ยนให้เป็นข้อมูลคำสั่งพื้นฐานซึ่งประกอบด้วย รหัสการทำงาน (Operation Code) และข้อมูลระบุตำแหน่ง (Position Vector) แล้วเก็บรวบรวมไว้ก่อนจะนำไปใช้ในกระบวนการแสดงภาพ และคำสั่งพื้นฐานคำสั่งแรกก็คือ

```
Move_Abs ( Vector point )
```

ส่วนการทำงานนี้ใช้สำหรับการกำหนดตำแหน่งปัจจุบัน ซึ่งจะเปลี่ยนคำสั่งให้เป็นข้อมูลพื้นฐาน ซึ่งมีองค์ประกอบดังนี้

MOVE point

ข้อมูลส่วนที่เป็นรหัสการทำงานคือ MOVE เป็นคำสั่งให้เคลื่อนย้ายตำแหน่งบนภาพไปที่ point ซึ่งเป็นตำแหน่งในระบบสามมิติ

องค์ประกอบพื้นฐานที่สำคัญของภาพในระบบนี้ได้แก่ เส้นตรง และข้อมูลที่จะใช้ในการสร้างเส้นตรงก็คือ ตำแหน่งปลายทั้งสองของเส้นตรง ในขณะที่ หน่วยข้อมูลพื้นฐานจะมีข้อมูลตำแหน่งเพียงตำแหน่งเดียว ดังนั้น คำสั่งลากเส้นจึงแยกการทำงานเป็นสองส่วน ได้แก่ การกำหนดตำแหน่งที่หนึ่ง แล้วลากเส้นไปยังตำแหน่งที่สอง ดังนี้

```
Move_Abs (point1)
Line_Abs (point2)
```

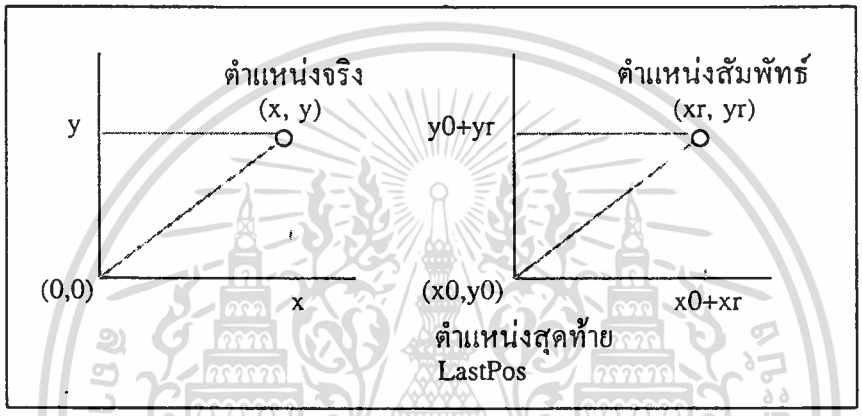
ในส่วนของคำสั่งลากเส้น ข้อมูลของคำสั่งก็จะประกอบด้วยรหัสการทำงานและข้อมูลตำแหน่งเหมือนกับคำสั่งกำหนดตำแหน่ง ดังนี้

LINE point2

คำสั่งนี้กำหนดให้ลากเส้นจากตำแหน่งสุดท้ายคือ point1 มายังตำแหน่ง point2

ภาพที่ 28

ภาพที่ได้จากการหมุนและเคลื่อนย้ายวัตถุภาพ



การระบุตำแหน่งในคำสั่งต่าง ๆ สามารถกำหนดโดยใช้ตำแหน่งจริง (absolute position) และตำแหน่งสัมพัทธ์ (relative position) ดังภาพที่ 28 ในคำสั่งเคลื่อนย้ายตำแหน่งจึงอาจจะใช้ข้อมูลเป็นตำแหน่งสัมพัทธ์ได้ดังนี้

Move_Rel (Vector rpoint)

ในทำนองเดียวกัน คำสั่งลากเส้นสามารถระบุเป็นตำแหน่งสัมพัทธ์ได้ดังนี้

Line_Rel (rpoint);

แต่ในการเก็บข้อมูลภาพ ได้กำหนดให้เก็บข้อมูลตำแหน่งเป็นตำแหน่งจริงทั้งหมด ดังนั้น ข้อมูลตำแหน่งสัมพัทธ์ จึงถูกเปลี่ยนเป็นตำแหน่งจริงก่อนที่จะนำไปเก็บ ดังนี้

MOVE (LastPos+rpoint)

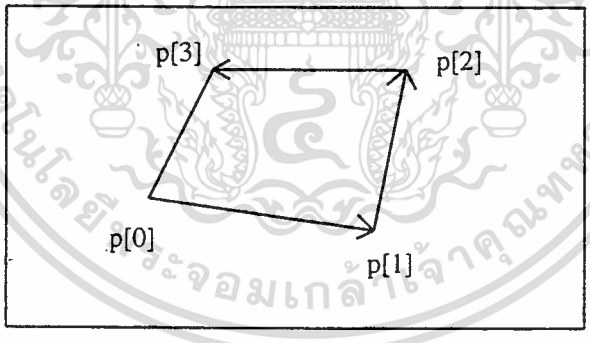
และเช่นเดียวกัน การกำหนดตำแหน่ง ข้อมูลตำแหน่งสัมพัทธ์ในคำสั่งลากเส้นจะถูก เปลี่ยนเป็นตำแหน่งจริง ดังนี้

LINE (LastPos+rpoint)

ด้วยเหตุนี้ ทุกครั้งที่มีการเปลี่ยนตำแหน่ง ไม่ว่าจะเป็นการเคลื่อนย้ายตำแหน่งธรรมดา หรือการลากเส้นตรง จะต้องมีการเก็บค่าตำแหน่งสุดท้าย คือ LastPos ไว้

ในระบบภาพที่มีเส้นตรงเป็นองค์ประกอบพื้นฐานของภาพ คำสั่งสร้างภาพใน ระดับสูงขึ้นมาที่ขาดไม่ได้ ก็คือ คำสั่งสำหรับสร้างรูปหลายเหลี่ยม (Polygon) ซึ่งใช้ข้อมูลเป็นตำแหน่งมุมทั้งหมดของรูปหลายเหลี่ยมนั้น โดยกำหนดตำแหน่งมุมให้เรียงลำดับกันในแบบทวนเข็มนาฬิกาสำหรับรูปเหลี่ยมที่หันหน้าออกมาทางผู้มองภาพ ดังแสดงในภาพที่ 29 สำหรับรูปเหลี่ยมที่มีข้อมูลตำแหน่งมุมเรียงลำดับตามเข็มนาฬิกา ก็จะเป็นพื้นผิวที่หันหลังให้กับผู้มองภาพ ข้อกำหนดในเรื่องลำดับข้อมูลนี้จะมีประโยชน์มากในขั้นตอนการประมวลผลภาพ เพราะทำให้ทราบว่าพื้นผิวส่วนไหนของวัตถุภาพหันมาทางผู้มองภาพ

ภาพที่ 29
การเก็บข้อมูลรูปเหลี่ยม



คำสั่งสร้างรูปหลายเหลี่ยมก็จะมีที่ใช้ข้อมูลตำแหน่งจริงและตำแหน่งสัมพัทธ์เช่นเดียวกัน ถ้าเป็นตำแหน่งสัมพัทธ์ก็จะถูกเปลี่ยนให้เป็นตำแหน่งจริงทั้งหมด แล้วเก็บอยู่ในรูปของข้อมูลพื้นฐานดังนี้

MOVE p[0]
POLY p[1]
POLY p[2]
POLY p[3]
LINE p[0]

รหัสคำสั่ง POLY เป็นคำสั่งให้ลากเส้นเหมือนกับ LINE แต่ที่ใช้รหัสต่างกันเพื่อบอกให้รู้ว่าเส้นตรงนี้เป็นส่วนหนึ่งของรูปหลายเหลี่ยม เนื่องจากขั้นตอนในการแสดงรูปเหลี่ยมจะต่างการแสดงเส้นตรง การปิดท้ายด้วยรหัสคำสั่ง LINE เพื่อบอกให้รู้ว่าเป็นจุดสุดท้าย

ตารางที่ 10

ส่วนการทำงานสำหรับการสร้างภาพ

Move_Abs (p)	กำหนดตำแหน่งจริง p
Move_Rel (pr)	กำหนดตำแหน่งสัมพัทธ์ pr
Line_Abs (p)	ลากเส้นไปที่ตำแหน่งจริง p
Line_Rel (pr)	ลากเส้นไปที่ตำแหน่งสัมพัทธ์ pr
Polygon_Abs (n, p)	สร้างรูป n เหลี่ยมจากข้อมูลตำแหน่งจริง p
Polygon_Rel (n, pr)	สร้างรูป n เหลี่ยมจากข้อมูลตำแหน่งสัมพัทธ์ pr

ในตารางที่ 10 เป็นรายชื่อของการทำงานพื้นฐานในส่วนของการสร้างภาพ ถ้าต้องการเพิ่มส่วนการทำงานพื้นฐานเข้าไปในระบบก็สามารถทำได้ คำสั่งพื้นฐานเหล่านี้อาจจะมีข้อมูลตำแหน่งมากกว่าหนึ่งตำแหน่ง เหมือนกับคำสั่งสร้างรูปหลายเหลี่ยม ซึ่งจะถูกเปลี่ยนมาอยู่ในรูปของหน่วยข้อมูลพื้นฐาน ตัวอย่างเช่น ข้อมูลสำหรับการสร้างภาพวงกลม ประกอบด้วยตำแหน่งจุดศูนย์กลาง และจุดสองจุดบนเส้นรอบวง เป็นต้น

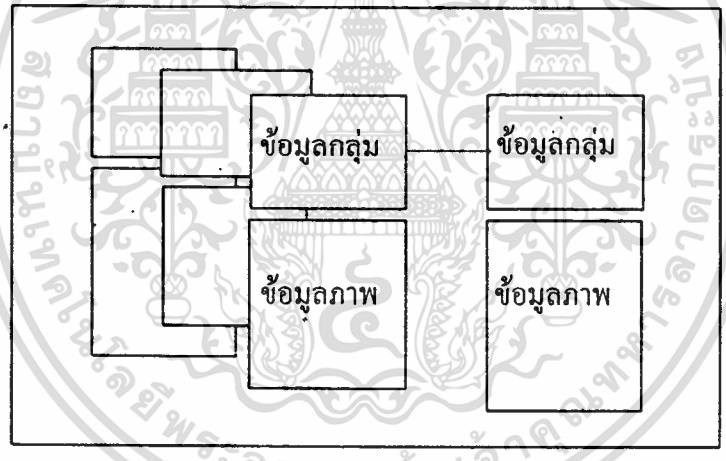
ข้อมูลรหัสการทำงานเป็นข้อมูลขนาดหนึ่งไบท์ ซึ่งเก็บรหัสคำสั่งได้ทั้งหมด 256 คำสั่ง ค่าตั้งแต่ 32 ซึ่งตรงกับรหัสตัวอักษร (ASCII Code) ก็จะถูกใช้ในกรณีที่ต้องการเก็บคำสั่งสำหรับสร้างตัวอักษร ส่วนการกำหนดค่าให้รหัสการทำงานอื่น ๆ จะใช้ค่ารหัสที่น้อยกว่า 32 และไม่ซ้ำกับค่าของรหัสเดิมที่มีอยู่แล้ว ส่วนของข้อมูลบอกตำแหน่งจะเป็นข้อมูลเว็คเตอร์ ซึ่งเป็นข้อมูลในระบบสามมิติ ที่มีค่าบนแกนทั้งสามเป็นเลขจำนวนจริง (float) ระหว่าง -1 กับ 1 ในกรณีที่ต้องการประหยัดเนื้อที่เก็บข้อมูล ก็อาจจะใช้วิธีเปลี่ยนค่านั้นให้เป็นเลขจำนวนเต็มโดยการคูณด้วยเลขจำนวนหนึ่ง เช่น 30000 เพื่อให้ได้ค่าที่อยู่ระหว่าง -30000 ถึง 30000 ซึ่งสามารถนำไปเก็บไว้ในตัวแปรเลขจำนวนเต็ม (integer) ได้ ในการนำข้อมูลกลับมาใช้ก็จะใช้ตัวเลขเดิมมาหาร ถึงแม้ว่า วิธีนี้จะช่วยประหยัดเนื้อที่เก็บข้อมูลได้ แต่ก็ทำให้ความละเอียดถูกต้องของข้อมูลลดลงไป

การเก็บข้อมูลภาพ

ปัญหาของการเก็บข้อมูลภาพ ก็คือ การกำหนดรูปแบบและวิธีการเก็บข้อมูล ในกรณีที่ข้อมูลภาพมีจำนวนมากหรือต้องการนำข้อมูลไปใช้ในภายหลัง ก็สามารถเก็บข้อมูลไว้ในแฟ้มแสดงภาพ (Display File) ได้ แต่โดยปกติข้อมูลในระหว่างการทำงานจะถูกเก็บอยู่ในหน่วยความจำเพื่อความสะดวกในการใช้งาน แต่ถ้าต้องการออกแบบเพื่อนำไปใช้กับงานที่มีข้อมูลภาพจำนวนมาก ก็อาจจะต้องเลี่ยงไปใช้ส่วนเก็บข้อมูลอื่น ๆ เช่น การใช้แฟ้มข้อมูล เป็นต้น

ภาพที่ 30

รูปแบบการเก็บข้อมูลภาพ



ในที่นี้ กำหนดให้เก็บข้อมูลภาพในหน่วยความจำ โดยแบ่งออกเป็นกลุ่มย่อย แต่ละกลุ่มจะเก็บข้อมูลของวัตถุอันหนึ่ง (Object) โดยมีคุณสมบัติประจำกลุ่ม เช่น การเปลี่ยนขนาด (Scaling) การย้ายตำแหน่ง (Translation) และ การหมุน (Rotation) ทั้งนี้เพื่อให้วัตถุแต่ละชิ้นมีการเปลี่ยนแปลงที่เป็นอิสระต่อกัน ข้อมูลวัตถุภาพจะถูกแยกเป็นส่วนที่บอกคุณสมบัติของกลุ่มและส่วนที่เป็นข้อมูลภาพ ดังภาพที่ 30 แต่ก่อนที่จะเก็บข้อมูลได้ ต้องมีการจองเนื้อที่สำหรับเก็บข้อมูลของวัตถุภาพนั้นด้วยคำสั่ง

New_Object ( )

ข้อมูลชุดใหม่จะถูกเชื่อมโยงเข้ากับข้อมูลเก่า ทำให้สามารถนำข้อมูลที่เก็บไว้มาใช้งานได้ตามลำดับ และถ้าไม่ต้องการวัตถุภาพใดก็สามารถใช้คำสั่งยกเลิกได้ ดังนี้

```
Delete_Object ( )
```

ในกรณีที่ต้องการยกเลิกข้อมูลทั้งหมดก็ทำได้เช่นเดียวกัน โดยใช้คำสั่ง

```
New_Picture ( );
```

โครงสร้างของข้อมูลที่เป็นคุณสมบัติประจำวัตถุภาพจะประกอบด้วย ส่วนที่เป็นข้อมูลคุณ สมบัติ ส่วนที่จะเชื่อมโยงกับข้อมูลภาพ และส่วนที่เชื่อมโยงกับข้อมูลกลุ่มอื่น ๆ ดังนี้

```
struct   object {
         int        Hide;                    // คุณสมบัติประจำกลุ่ม
         int        Color;
         Vector     Scale;
         Vector     Translate;
         Vector     Rotate;
         CodePtr          Element;           // ตำแหน่งข้อมูลภาพ
         struct object    *Prev;             // วัตถุภาพก่อนหน้า
         struct obect     *Next;             // วัตถุภาพต่อไป
};
typedef struct object     *ObjPtr;
```

หน่วยข้อมูลพื้นฐานของภาพซึ่งประกอบด้วย รหัสการทำงาน และ ข้อมูลตำแหน่ง จะถูกกำหนดเป็นข้อมูลโครงสร้างดังนี้

```
typedef unsigned char    OpCode;
struct   code  {
         OpCodeOp;                    // รหัสการทำงาน
         Vector          Pt;                  // ค่าตำแหน่ง
} Code;
typedef struct code      *CodePtr;    // ตัวชี้ข้อมูล
```

เมื่อได้โครงสร้างของข้อมูลภาพแล้ว ก็ต้องมีส่วนการทำงานที่ช่วยในการกำหนดคุณสมบัติรวมของวัตถุภาพ ไม่ว่าจะเป็น การหมุนวัตถุภาพ การกำหนดค่าสี หรือกำหนดว่าจะให้แสดงวัตถุภาพชิ้นนั้นหรือไม่ และที่สำคัญจะต้องมีส่วนการทำงานสำหรับเลือกวัตถุภาพที่ต้องการ ก่อนที่จะนำข้อมูลภาพมาใช้ ในตารางที่ 11 เป็นส่วนการทำงานที่เกี่ยวข้องกับการเก็บข้อมูลภาพทั้งหมด

ตารางที่ 11

ส่วนการทำงานสำหรับการเก็บข้อมูล

New_Object ( )	จองที่สำหรับวัตถุภาพชิ้นใหม่
Deletel_Object ( )	ยกเลิกวัตถุภาพปัจจุบัน
New_Picture ( )	ยกเลิกข้อมูลภาพทั้งหมด
Hide_Object ( )	กำหนดให้ไม่แสดงวัตถุภาพ
Show_Object ( )	กำหนดให้แสดงวัตถุภาพ
Color_Object (color)	กำหนดสี color ให้วัตถุภาพ
Select_Object (n)	เลือกวัตถุภาพลำดับที่ n
Next_Object ( )	เลือกวัตถุภาพลำดับต่อไป
Prev_Object ( )	เลือกวัตถุภาพลำดับก่อนหน้า
Rotate_Object (dx,dy,dz)	หมุนวัตถุภาพด้วยมุม dx, dy, dz รอบแกน x, y และ z ตามลำดับ
Scale_Object (sx,sy,sz)	เปลี่ยนขนาดวัตถุภาพด้วยค่า sx, sy, sz สำหรับแกน x, y และ z ตามลำดับ
Trans_Object (tx,ty,tz)	กำหนดตำแหน่งวัตถุภาพด้วยค่า tx, ty, tz สำหรับแกน x, y และ z ตามลำดับ
First_Object ( )	เลือกวัตถุภาพลำดับแรก

ที่กล่าวมาทั้งหมดนี้ เป็นตัวอย่างของการกำหนดโครงสร้างการเก็บข้อมูลเพื่อรองรับข้อมูลภาพจำนวนมากนอกเหนือจากการใช้แฟ้มข้อมูลทั่วไป ในกรณีที่ข้อมูลมีจำนวนไม่มากก็อาจจะกำหนดเป็นตัวแปรชุด (array) ธรรมดาก็ได้

การประมวลผลภาพ

เมื่อได้ข้อมูลภาพทั้งหมดจากกระบวนการสร้างภาพแล้ว ขั้นตอนต่อไปก็จะเป็นคำสั่งให้นำข้อมูลที่เก็บไว้มาผ่านกระบวนการประมวลผลข้อมูล เพื่อให้ได้ภาพสุดท้ายออกที่จอภาพ เริ่มจากการกำหนดให้ลบพื้นที่แสดงภาพก่อนที่จะแสดงภาพใหม่ โดยใช้คำสั่ง

Clear_Display ( )

นอกจากนี้ก็จะมีคำสั่งกำหนดตัวแปรสำหรับคุมรูปแบบการประมวลผล เช่น

Solid_Display ( )

กำหนดให้ใช้วิธีการประมวลผลสำหรับการแสดงภาพแบบทึบแสง"เมื่อกำหนดค่าต่าง ๆ แล้ว ก็จะสั่งให้นำข้อมูลภาพไปประมวลผลเพื่อแสดงภาพ ด้วยคำสั่ง

Make_Picture ( )

ขั้นตอนในการประมวลผลเพื่อแสดงภาพ จะเริ่มจากการเลือกวัตถุภาพ โดยเริ่มจากวัตถุภาพชิ้นแรก หลังจากอ่านข้อมูลเข้ามาประมวลผลแล้ว จะมีการเลือกวัตถุภาพชิ้นต่อไปตามลำดับจนหมด ดังนี้

```
void Make_Picture ( )
{
        int     lastobj = False;
        if (ClearDisplay) {                          //ลบจอภาพ
                Clear_View (BackColor);
                ClearDisplay = False;
        }
        First_Object ( );                            //วัตถุภาพชิ้นแรก
        while (!lastobj) {
                Interpret ( );                       //ประมวลผล
```

```
            lastobj = Next_Object ( );          //วัตถุภาพชิ้นต่อไป
        }
        Display_Buffer ( );                     //แสดงภาพ
    }
```

ในกระบวนการของการเลือกวัตถุภาพไม่ว่าจะเป็น First_Object( ) หรือ Next_Object( ) ข้อมูลที่เกี่ยวกับคุณสมบัติของวัตถุภาพนั้น เช่น สี หรือคุณสมบัติอื่น ๆ ที่อาจจะเพิ่มเติมในภายหลัง จะถูกนำมากำหนดสภาพแวดล้อมของการประมวลผล รวมทั้งข้อมูลควบคุมการหมุน การเปลี่ยนขนาด และการย้ายตำแหน่ง ก็จะถูกนำมาสร้างเป็นแมตริกซ์แปลงข้อมูล และเมื่อมีการอ่านข้อมูลองค์ประกอบของวัตถุภาพ ส่วนที่เป็นข้อมูลตำแหน่งจะถูกนำมาเปลี่ยนโดยแมตริกซ์แปลงข้อมูล ก่อนที่จะส่งต่อไปให้กระบวนการประมวลผลขั้นต่อไป

ข้อมูลที่แปรสภาพแล้วจะถูกส่งไปให้ส่วนการทำงาน Intepret ( ) เพื่อเข้าสู่กระบวนการแยกแยะคำสั่งสร้างภาพ ซึ่งจะจำแนกการทำงานตามรหัสคำสั่ง ดังนี้

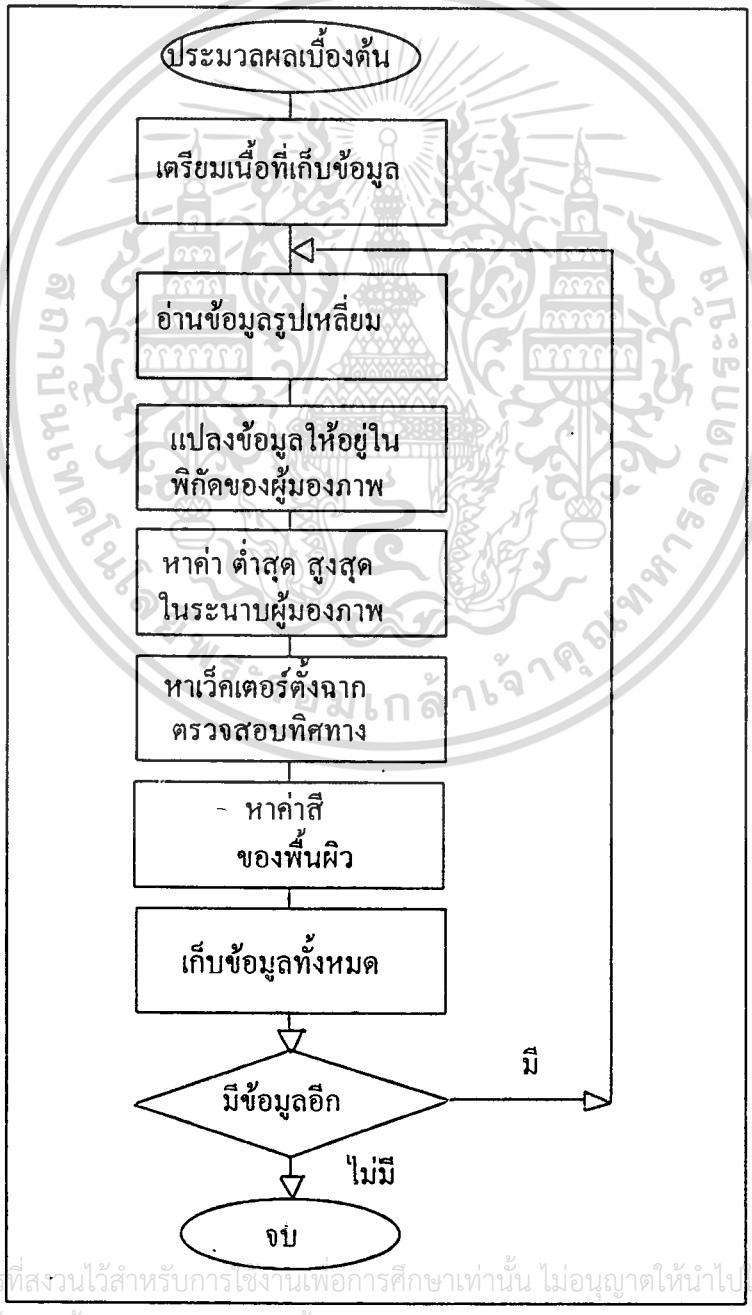
```
Get_Code (&op, position);                   // อ่านข้อมูล
switch (op) {                               // เลือกการทำงาน
case MOVE:
        Do_Move (position);                 // กำหนดตำแหน่ง
        break;
case LINE:
        Do_Line (position);                 // ลากเส้น
        break;
case POLY:
        Do_Poly (position);                 // สร้างรูปเหลี่ยม
}
```

เมื่อพบรหัสการทำงาน MOVE ก็จะไปทำในส่วนการเคลื่อนย้ายตำแหน่ง โดยข้อมูลตำแหน่งจะถูกเก็บไว้เป็นตำแหน่งสุดท้าย ส่วนรหัสการทำงาน LINE จะทำในส่วนของการลากเส้นตรง โดยใช้ข้อมูลตำแหน่งสุดท้ายกับข้อมูลตำแหน่งใหม่ แต่ในกรณีของรหัสของรูปหลายเหลี่ยม POLY ก็จะทำการอ่านข้อมูลตำแหน่งมุมทั้งหมดของรูปหลายเหลี่ยมนั้นมาเก็บไว้เพื่อรอการประมวลผลเพื่อแสดงภาพ

ในการแสดงภาพของรูปหลายเหลี่ยมนั้น จะมีขั้นตอนที่ค่อนข้างซับซ้อนกว่าการลากเส้นตรงธรรมดา เพราะข้อมูลทั้งหมดในภาพมีผลกระทบถึงกัน ก่อนที่จะนำข้อมูลมาตรวจสอบว่าพื้นผิวส่วนไหนอยู่ในสายตาของผู้มองภาพ ข้อมูลวัตถุภาพจะถูกนำไปผ่านการประมวลผลเบื้องต้นก่อน ตามขั้นตอนในภาพที่ 31

ภาพที่ 31

ขั้นตอนการประมวลผลเบื้องต้น

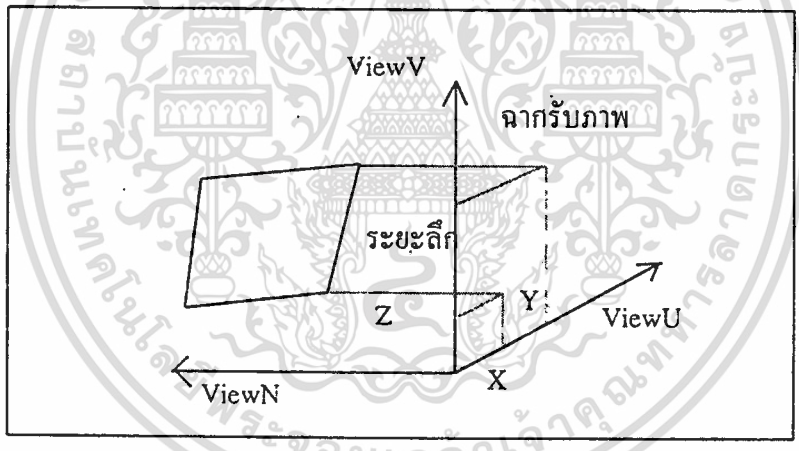


ในกระบวนการประมวลผลขั้นต้น ข้อมูลตำแหน่งทั้งหลายจะถูกคำนวณจากคุณสมบัติเฉพาะตัว เช่น การปลี่ยนขนาด และการหมุน จากนั้นก็จะฉายตำแหน่ง (map) ไปบนระบบแกนของผู้มองภาพ เพื่อให้ได้ตำแหน่งของจุดภาพในแนวแกนนอนและแกนตั้งของฉากรับภาพ ข้อมูลตำแหน่งที่ได้นี้เป็นตำแหน่งที่ผู้มองภาพมองเห็นจริง จึงสามารถจะนำไปแสดงภาพได้เลย

สิ่งที่ได้จากการประมวลผลเบื้องต้นอีกอย่างหนึ่ง ก็คือ ระยะระหว่างตำแหน่งจริงกับฉากรับภาพ ดังภาพที่ 32 ข้อมูลในแนวแกน Z ของรูปเหลี่ยมทั้งหมด จะอยู่ในทิศทางของการมองภาพ ทำให้รู้ว่าตำแหน่งไหนอยู่ใกล้ผู้มองภาพมากกว่ากันดังภาพที่ 33 ข้อมูลเหล่านี้จึงสามารถจะนำไปใช้ในการประมวลผลเพื่อหาการบังกันของรูปเหลี่ยมได้

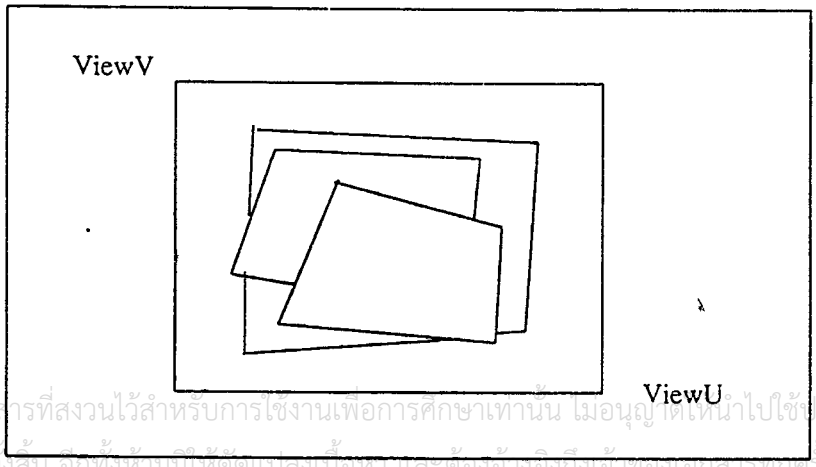
ภาพที่ 32

การเก็บข้อมูลในระบบระยะลึก



ภาพที่ 33

ข้อมูลรูปเหลี่ยมที่ผ่านการประมวลผลเบื้องต้น



การเก็บข้อมูลตำแหน่งภาพในแบบที่มีระยะใกล้ไกลจากผู้มองภาพ ซึ่งก็คือระยะในแกน Z นั่นเอง เรียกว่า การเก็บข้อมูลภาพแบบมีระยะลึก (Z-Buffering) [9] ข้อมูลเหล่านี้สามารถนำมาใช้ประโยชน์ในกระบวนการตรวจสอบต่าง ๆ ในขั้นตอนการประมวลผลเพื่อแสดงภาพ โดยข้อมูลทั้งหมดจะถูกเก็บรวมไว้ แล้วส่งต่อไปให้ส่วนการทำงาน Display_Buffer ( ) เพื่อการประมวลผลขั้นสุดท้าย

ลักษณะโครงสร้างของที่เก็บข้อมูลรูปเหลี่ยมแต่ละรูป จะประกอบด้วยส่วนที่เป็น ข้อมูลตำแหน่งมุม และส่วนของคุณบัติเพิ่มเติมที่ได้จากการประมวลผลขั้นต้น ดังนี้

```
typedef struct   {
        unsigned char Object;          // หมายเลขวัตถุภาพ
        unsigned char  Num;            // จำนวนด้าน
        unsigned char  Color;          // สี
        unsigned char Back;            // ด้านหลัง
        Vector        Normal;          // แนวตั้งฉาก
        Vector        Lower;           // ตำแหน่งต่ำสุด
        Vector        Upper;           // ตำแหน่งสูงสุด
        Vector        Node [MaxNode];  // ข้อมูลตำแหน่ง
} Polygon;
```

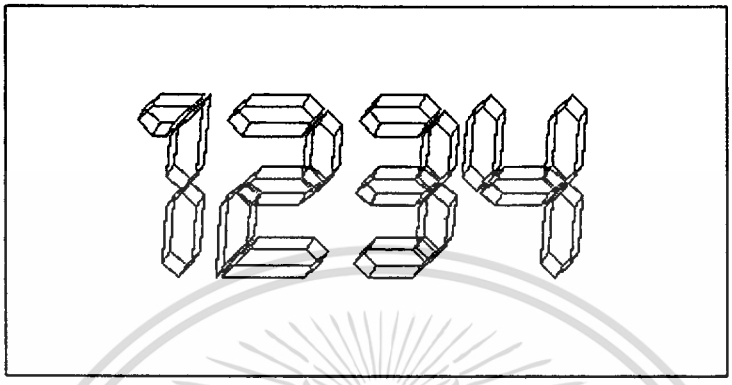
ข้อมูลในส่วนที่เป็นพื้นผิวของวัตถุภาพ จะถูกนำมาประมวลผลในแบบต่าง ๆ ทั้งนี้ก็ขึ้นอยู่กับว่า ระบบภาพมีส่วนการทำงานสำหรับประมวลผลภาพในรูปแบบใดบ้าง การประมวลผลภาพอาจจะทำได้เป็นสองระดับ ตามลักษณะของวัตถุภาพที่ต้องการ ได้แก่

- วัตถุภาพแบบโครงลวด (Wire-Frame Object)
- วัตถุภาพแบบพื้นผิวทึบ (Solid Object)

ระบบพื้นฐานที่สุดของการแสดงภาพก็คือ การแสดงภาพวัตถุแบบโครงลวด โดยไม่มีการตรวจการมองเห็นของพื้นผิววัตถุ เส้นตรงทุกเส้นที่เป็นองค์ประกอบของวัตถุภาพจะถูกแสดงออกมาทั้งหมด ดังภาพที่ 34 ในกรณีนี้ คำสั่งสร้างรูปหลายเหลี่ยมก็จะเหมือนคำสั่งลากเส้นธรรมดา ข้อมูลภาพทั้งหมดจะถูกนำมาแปลเป็นคำสั่งแสดงภาพ แล้วส่งต่อไปให้ส่วนแสดงภาพโดยไม่มีการประมวลผลใด ๆ นอกจากการแปลงข้อมูลตำแหน่งในการประมวลผลเบื้องต้นเท่านั้น

ภาพที่ 34

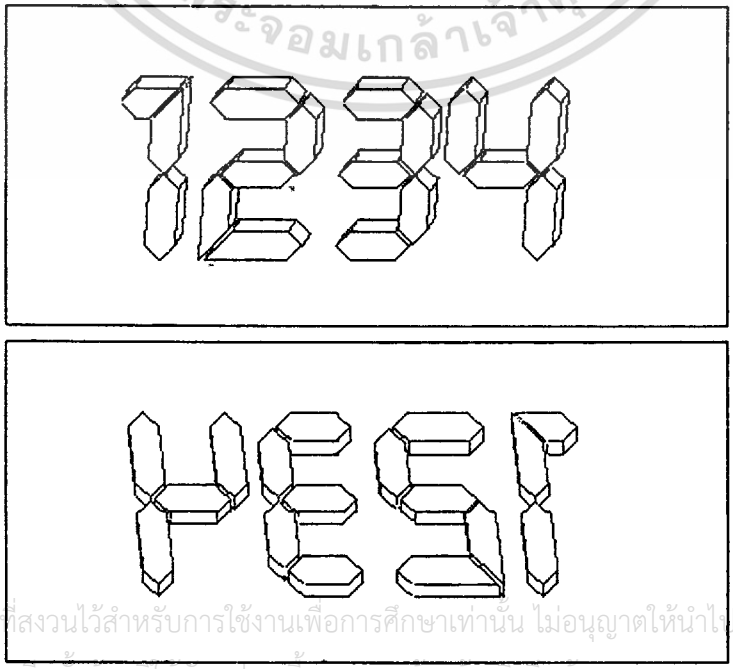
ภาพโครงร่างที่แสดงเส้นทั้งหมดของวัตถุภาพ



ตัวอย่างของการประมวลผลภาพแบบง่าย ๆ ก็คือ การตรวจสอบว่าพื้นผิวนั้นหันหน้าไปทางไหน ถ้าพื้นผิวนั้นหันหน้าไปในทิศตรงกันข้ามกับผู้มองภาพ (Back Face) ก็หมายความว่าผู้มองภาพจะมองไม่เห็นพื้นผิวนั้น ทำให้เห็นรูปทรงของวัตถุภาพซึ่งมีพื้นผิวปิดหมดทุกด้านได้ชัดเจนขึ้น ดังตัวอย่างในภาพที่ 35 พื้นผิวที่อยู่ด้านหลังภาพตัวอักษรแต่ละตัวจะไม่ถูกแสดงออกมา เมื่อพื้นผิวที่อยู่ด้านหน้าถูกพลิกกลับไปด้านหลัง ก็จะมองไม่เห็นพื้นผิวนั้น

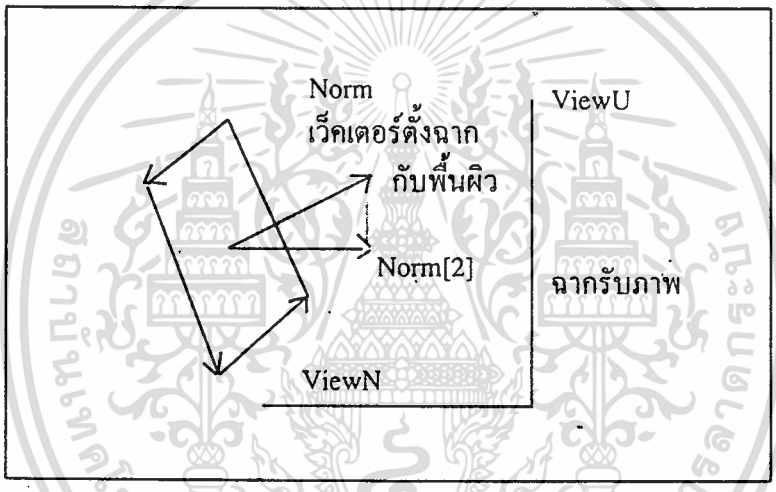
ภาพที่ 35

ภาพโครงร่างที่ไม่แสดงพื้นผิวด้านหลังของวัตถุภาพ



ในการหาทิศทางของเว็คเตอร์ตั้งฉากหรือลำดับการหมุนของตำแหน่งมุม จะใช้วิธีนำเว็คเตอร์ของด้านที่ติดกันมาทำ cross product กัน ได้เป็นเว็คเตอร์ที่ตั้งฉากกับด้านทั้งสอง ทั้งนี้มีเงื่อนไขว่าด้านทั้งสองจะต้องทำมุมน้อยกว่า 180 องศา ไม่เช่นนั้นเว็คเตอร์ที่ได้จะเป็นเว็คเตอร์ในทิศทางตรงกันข้าม ซึ่งอาจจะแก้ไขโดยใช้ค่าที่ได้จากการจับคู่ด้านอื่น ๆ มาเฉลี่ยกันก็ได้ เพราะว่ามุมส่วนใหญ่ของรูปเหลี่ยมทั่วไปจะมีค่าน้อยกว่า 180 องศา

ภาพที่ 36
การตรวจสอบทิศทางของพื้นผิว

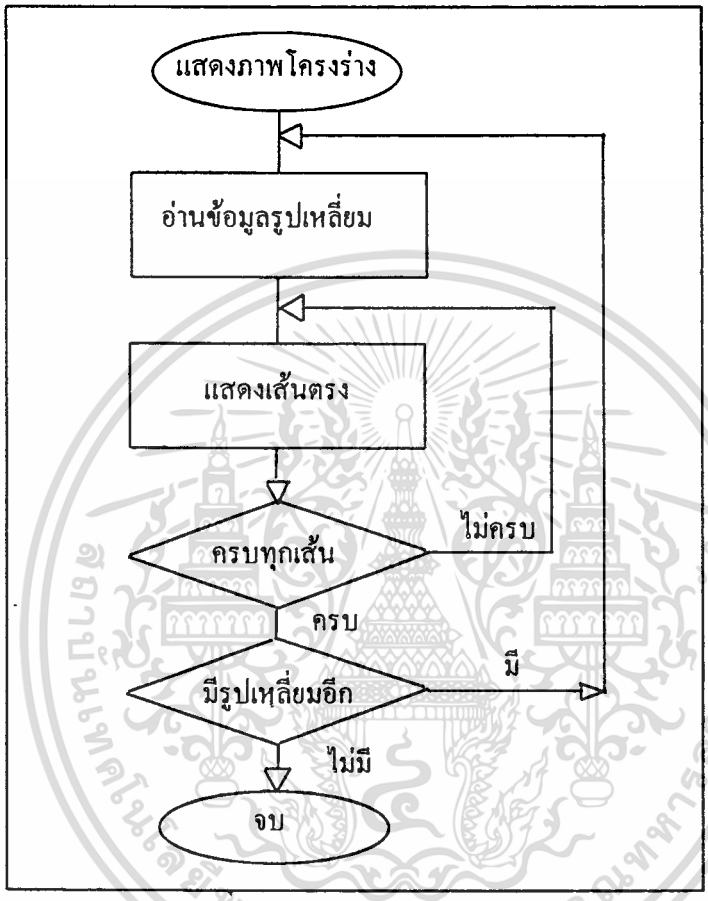


จากการตรวจสอบทิศทางของเว็คเตอร์ที่ตั้งฉากกับพื้นผิว ก็จะทราบว่าพื้นผิวนั้นหันหน้าไปทางไหน ทิศทางของเว็คเตอร์ตั้งฉากนี้เป็นผลที่ได้จากการกำหนดตำแหน่งมุมของพื้นผิวที่เรียงลำดับแบบทวนเข็มนาฬิกาดังภาพที่ 36 เป็นพื้นผิวที่หันหน้าออก และเมื่อพื้นผิวนี้พลิกกลับไปด้านหลัง ตำแหน่งมุมก็จะเรียงลำดับในทิศทางตรงกันข้าม

ถึงแม้ว่าการไม่แสดงพื้นผิวด้านหลังของวัตถุภาพ จะทำให้เห็นภาพได้ชัดเจนขึ้น แต่ถ้ามีวัตถุภาพมากกว่าหนึ่งชิ้นซ้อนกันอยู่ ต้องมีการตรวจสอบการบังกันของพื้นผิววัตถุแต่ละชิ้น การเก็บข้อมูลตำแหน่งที่มีระยะลึกดังกล่าว ก็เป็นวิธีการหนึ่งที่นำมาใช้ในกระบวนการตรวจสอบการบังกันของพื้นผิว ในกรณีของการแสดงภาพแบบโครงลวดจะใช้วิธีลากเส้นทีละเส้นตามขั้นตอนในภาพที่ 37 การลากเส้นตรงจะใช้วิธีที่ดัดแปลงมาจากขั้นตอนการลากเส้นในพิกัดจอภาพ แต่เพิ่มส่วนที่จะตรวจสอบว่าแต่ละจุดบนเส้นนั้นถูกบังด้วยพื้นผิวของวัตถุภาพต่าง ๆ หรือไม่ โดยใช้เว็คเตอร์แนวแกน Z ซึ่งผ่านจุดที่จะตรวจสอบ ถ้าเว็คเตอร์นั้นไปตัดกับรูปเหลี่ยมที่อยู่ใกล้ผู้มองภาพกว่า แสดงว่าจุดตรวจสอบนั้นถูกบังอยู่ ผลที่ได้ก็จะเป็นดังตัวอย่างในภาพที่ 5.11

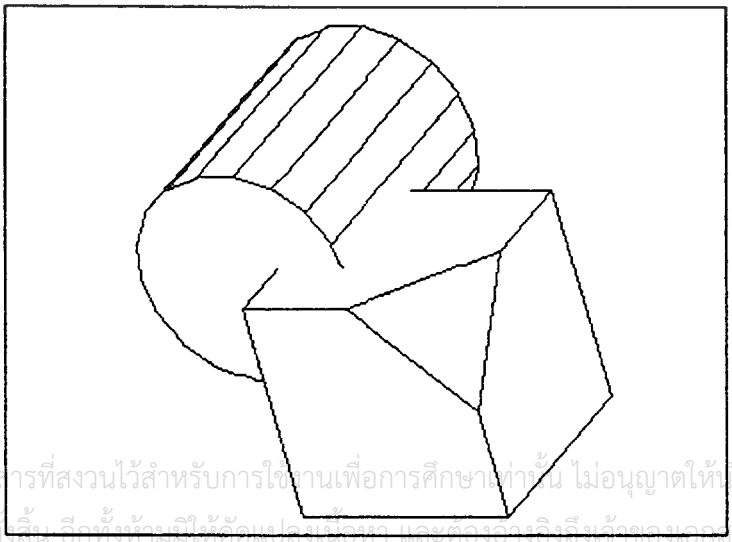
ภาพที่ 37

ขั้นตอนการแสดงภาพแบบโครงร่าง



ภาพที่ 38

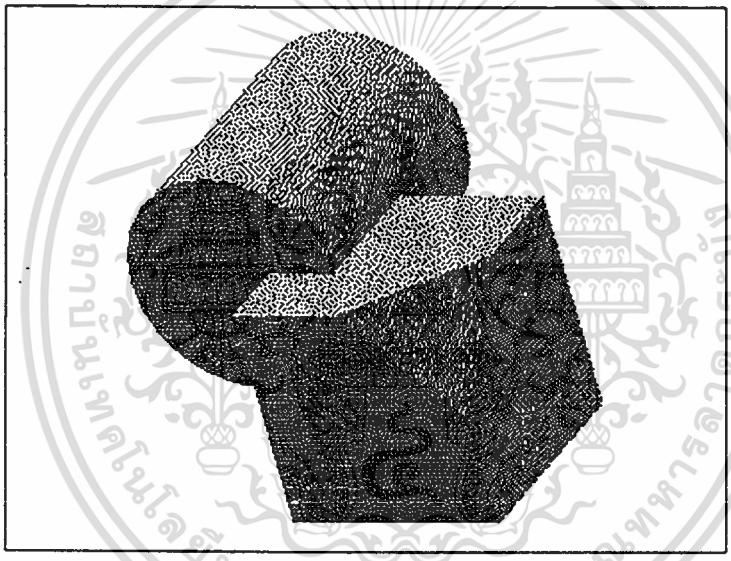
ภาพโครงร่างที่มีการตรวจสอบการบังกันของวัตถุภาพ



กระบวนการประมวลผลเพื่อแสดงภาพอีกแบบหนึ่งก็คือ การนำข้อมูลมาสร้างเป็นภาพวัตถุที่มีพื้นผิวทึบ ดังรูปที่ 39 สำหรับวิธีการที่ใช้ในการแสดงภาพพื้นผิวทึบ อาจจะใช้วิธีแสดงพื้นผิวแต่ละอัน ตามลำดับของระยะทางระหว่างวัตถุภาพกับฉากรับภาพ ตั้งแต่พื้นผิวที่อยู่ไกลที่สุดมาจนถึงพื้นผิวที่อยู่ใกล้ที่สุด

ภาพที่ 39

การแสดงภาพวัตถุแบบพื้นผิวทึบ

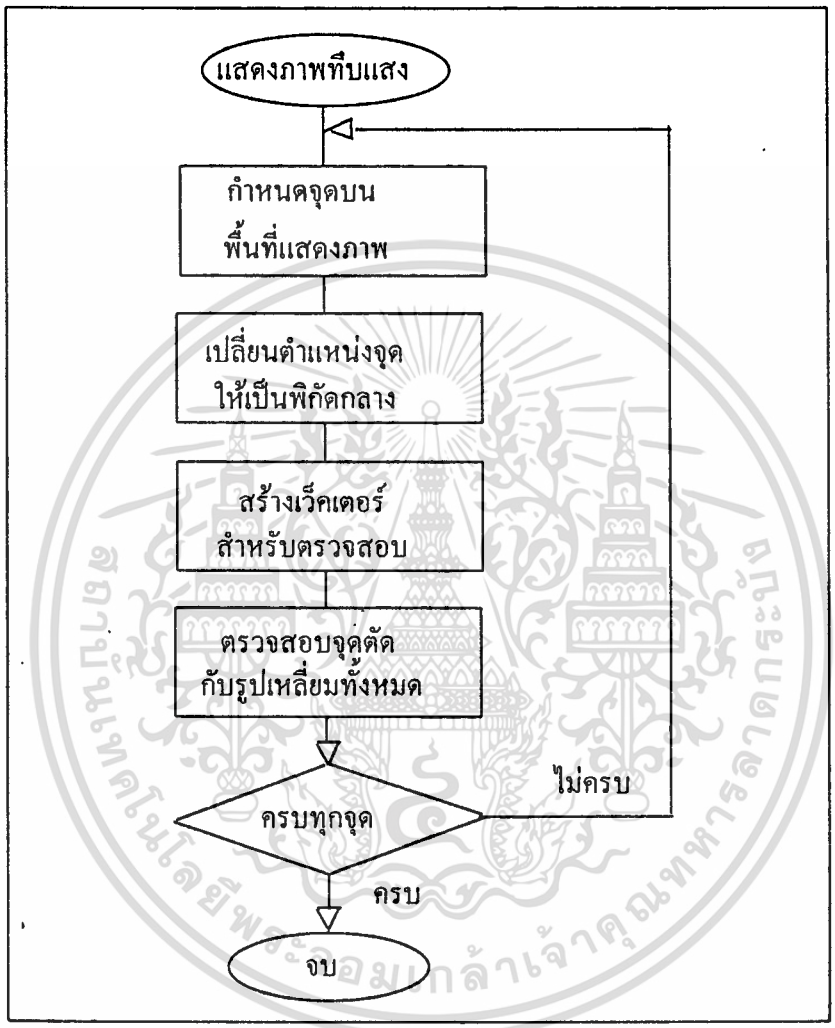


การแสดงพื้นผิวโดยเรียงลำดับตามระยะทางนั้น จะมีปัญหาในกรณีที่มีพื้นผิวตัดกันดังตัวอย่างของวัตถุภาพสองชิ้น สำหรับกระบวนการแสดงภาพพื้นผิวทึบในที่นี้ ดัดแปลงมาจากการตามทิศทางของแสง (Ray Tracing) [10] โดยการไล่ตรวจสอบตำแหน่งบนฉากรับภาพทีละจุด ตามขั้นตอนในภาพที่ 40

การตรวจสอบจะใช้วิธีมองผ่านจุดแต่ละจุดบนพื้นผิวแสดงภาพ โดยใช้เว็คเตอร์ตั้งแต่ตำแหน่งผู้มองภาพผ่านจุดเหล่านั้นลึกลงไปตามแนวแกน Z ดังภาพที่ 41 แล้วนำเว็คเตอร์นั้นไปตรวจสอบกับรูปเหลี่ยมทั้งหมดตามขั้นตอนในภาพที่ 42 เพื่อหาจุดตัดของเว็คเตอรกับรูปเหลี่ยมที่อยู่ใกล้ที่สุด ซึ่งจะเป็นจุดที่ปรากฏบนจอภาพ สำหรับสีของจุดนั้นจะได้จากสีของพื้นผิวนั้น ซึ่งจะต้องมีการกำหนดตรวจสอบมุมที่แสงตกกระทบพื้นผิวนั้น โดยนำทิศทางของแสงเทียบกับเว็คเตอร์ตั้งฉากกับพื้นผิว ทำให้ได้ระดับความเข้มสีของพื้นผิวนั้น

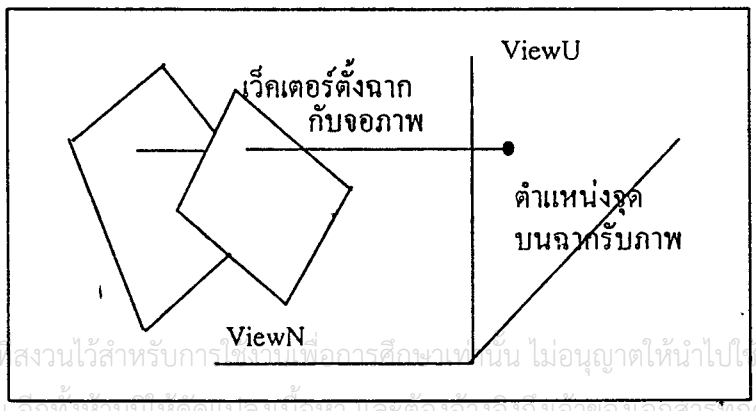
ภาพที่ 40

ขั้นตอนการแสดงภาพุแบบพื้นผิวทึบ



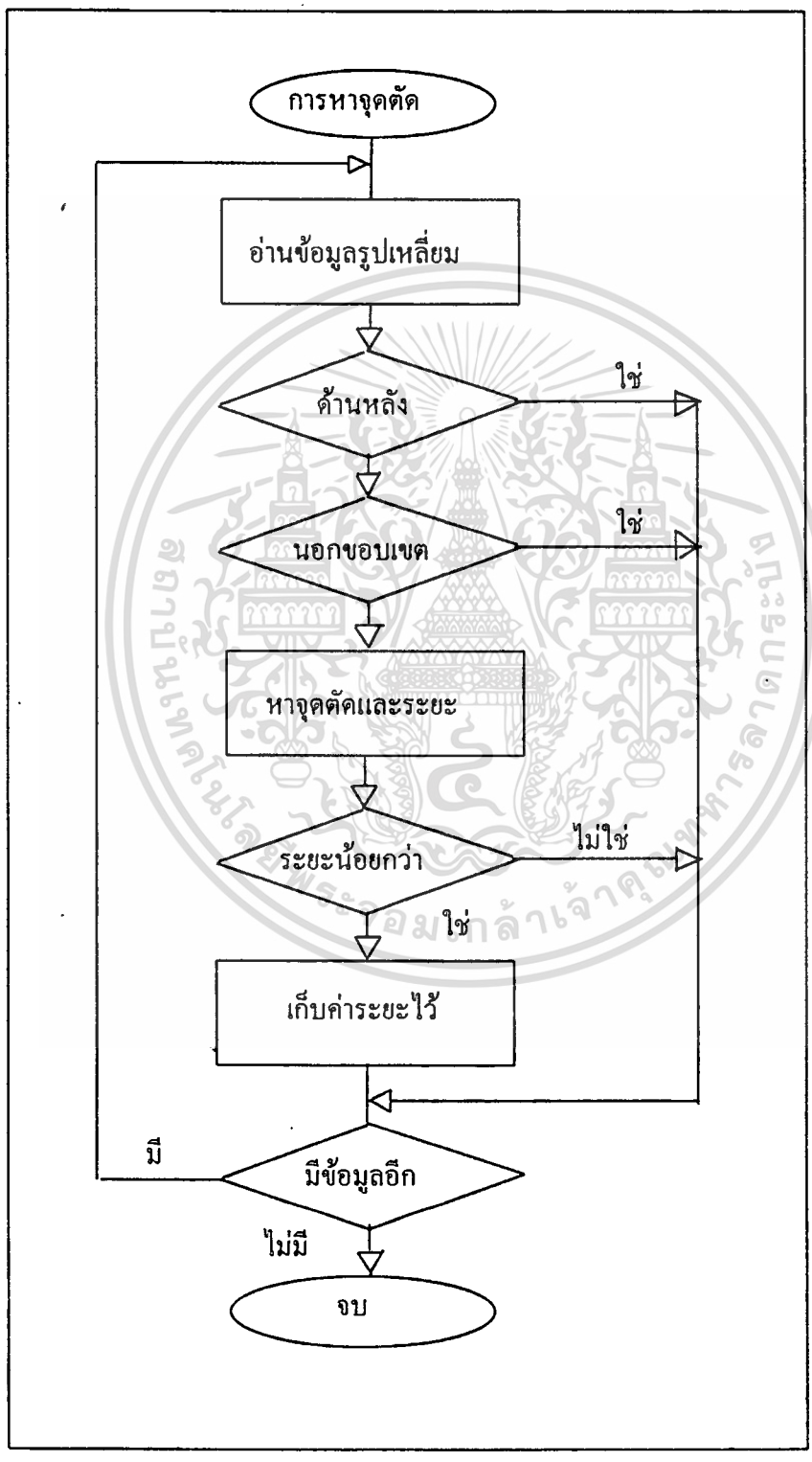
ภาพที่ 41

การหาพื้นผิวที่อยู่ใกล้จุดบนฉากรับภาพมากที่สุด



ภาพที่ 42

การหาระยะจุดตัดกับรูปเหลี่ยมที่อยู่ใกล้สุด



กระบวนการประมวลผลภาพเป็นกระบวนการภายในของระบบภาพ ซึ่งผู้เขียนโปรแกรม จะต้องเลือกกระบวนการที่เหมาะสมใส่เข้าไปในระบบภาพ ในส่วนของการใช้งาน ผู้ใช้มีหน้าที่เตรียมข้อมูลภาพ เมื่อได้ข้อมูลภาพแล้ว ผู้ใช้ก็สามารถควบคุมการแสดงภาพในแบบต่าง ๆ ได้ โดยใช้ข้อมูลวัตถุภาพชุดเดิม นอกจากการเปลี่ยนคุณสมบัติของวัตถุภาพ และทิศทางการมองภาพแล้ว ผู้ใช้ก็สามารถกำหนดลักษณะของภาพที่ต้องการ โดยใช้ส่วนการทำงานต่าง ๆ ในตารางที่ 12 ไม่ว่าจะเป็นการเลือกวิธีการแสดงภาพ หรือการกำหนดทิศทางของแสง แล้วสั่งให้แสดงภาพออกมา

ตารางที่ 12

ส่วนการทำงานสำหรับควบคุมการแสดงภาพ

Clear_Display ( )	ให้ลบภาพเดิมก่อนแสดงภาพใหม่
Make_Picture ( )	ให้นำข้อมูลมาแสดงภาพ
Set_BackColor (color)	กำหนดสี color ให้ฉากหลัง
Set_ForeColor (color)	กำหนดสี color ให้วัตถุภาพ
Hide_BackFace ( )	ไม่ให้แสดงพื้นผิวด้านหลัง
Show_BackFace ( )	ให้แสดงพื้นผิวด้านหลัง
Wire_Display ( )	ให้แสดงภาพแบบโครงลวด
Solid_Display ( )	ให้แสดงภาพแบบพื้นผิวทึบ
Light_Direction (direction)	กำหนดทิศทาง direction ของแสง

ส่วนการทำงานทั้งหมดในส่วนของการสร้างภาพ ตั้งแต่สร้างวัตถุภาพ การจัดตำแหน่งและมุมของวัตถุภาพ ตลอดจนการกำหนดทิศทางการฉายภาพ มีไว้เพื่อให้ส่วนใช้งานนำไปกำหนดลักษณะของภาพที่ต้องการ การใช้งานก็จะมีอยู่หลายระดับ ผู้ใช้อาจจะใช้คำสั่งพื้นฐานโดยตรง หรือ นำไปสร้างเป็นส่วนการทำงานในระดับสูงขึ้นไป เพื่อให้ได้ส่วนการทำงานใช้กับงานที่ต้องการได้เลย

บทที่ 6
ส่วนใช้งาน

ส่วนการทำงานที่เป็นพื้นฐานสำหรับการใช้งานในระบบภาพ ทั้งส่วนสำหรับสร้างภาพสองมิติและส่วนสำหรับสร้างภาพสามมิติ สามารถนำไปประยุกต์ใช้ในงานต่าง ๆ ได้มากมาย โดยโครงสร้างหลักของส่วนใช้งานจะประกอบด้วยขั้นตอนดังนี้

เตรียมระบบภาพ
เตรียมข้อมูลภาพ
แสดงภาพ
ออกจากระบบภาพ

ขั้นตอนแรกของส่วนใช้งานคือ การเตรียมระบบภาพให้พร้อมสำหรับการแสดงภาพ เริ่มจากการเข้าสู่ระบบภาพ การเตรียมพื้นที่แสดงภาพ และเตรียมการแสดงภาพในระบบสามมิติ ดังนี้

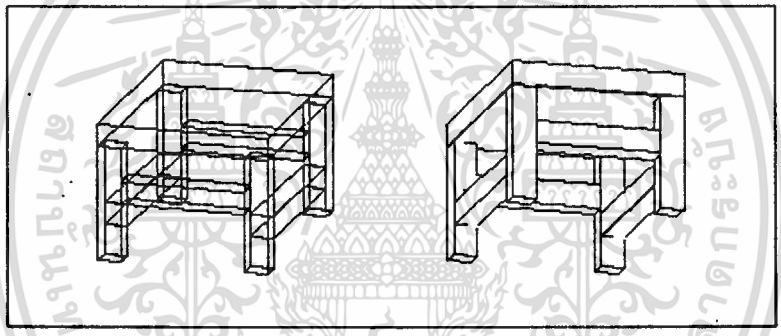
```
On_Graph (1, 0x5D);
Init_Window ( );
Init_3D ( );
```

ในตัวอย่างนี้เป็นการเตรียมระบบภาพแบบพื้นฐาน ซึ่งอาจจะปรับเปลี่ยนได้ตามต้องการ โดยใช้คำสั่งพื้นฐานต่าง ๆ เช่น กำหนดพื้นที่แสดงภาพโดยคำสั่ง **Set_Window( )** หรือ กำหนดตำแหน่งของผู้มองภาพด้วยคำสั่ง **View_Position( )** หลังจากเสร็จสิ้นการทำงาน ก็ออกจากระบบภาพด้วยคำสั่ง **Off_Graph( )** แต่ขั้นตอนหลักของส่วนใช้งานจริง ๆ แล้วจะประกอบด้วยขั้นตอนการเตรียมข้อมูลภาพ และ ขั้นตอนการแสดงภาพ

การเตรียมข้อมูล

สำหรับกระบวนการเตรียมข้อมูลภาพหรือการสร้างข้อมูลภาพ เป็นกระบวนการที่ใช้เวลาค่อนข้างมาก ยิ่งภาพที่มีองค์ประกอบซับซ้อนมากเท่าไหร่ การเตรียมข้อมูลก็จะยิ่งยากขึ้นเท่านั้น ดังตัวอย่างในภาพที่ 43 เป็นภาพที่แสดงรายละเอียดของโต๊ะสี่เหลี่ยมตัวหนึ่ง โต๊ะตัวนี้ประกอบด้วย พื้นโต๊ะ ขาสี่ข้าง และคานเชื่อมระหว่างขาทั้งสี่ รวมเป็นชิ้นส่วนทั้งหมด 9 ชิ้น

ภาพที่ 43
ภาพแสดงชิ้นส่วนทั้งหมดของโต๊ะตัวหนึ่ง



การเตรียมข้อมูลอาจจะให้ผู้ใช้ป้อนเข้ามาเอง โดยส่วนใช้งานต้องมีส่วนการทำงานสำหรับรับข้อมูลจากผู้ใช้มาเปลี่ยนเป็นคำสั่งสร้างภาพ ตัวอย่างเช่น ส่วนของการเตรียมข้อมูลของรูปหลายเหลี่ยมจะมีลักษณะดังนี้

```
Vector    *node;
int       point=0, lastpoint=0;
node = (Vector *) malloc (MAXNODE*sizeof(Vector));
while (!lastpoint) {
        lastpoint = Get_Node (node[point]);          //รับข้อมูล
        point = point + 1;                           //จำนวนจุด
}
if (point>0)     Polygon_Abs (point, node);          //เก็บข้อมูล
```

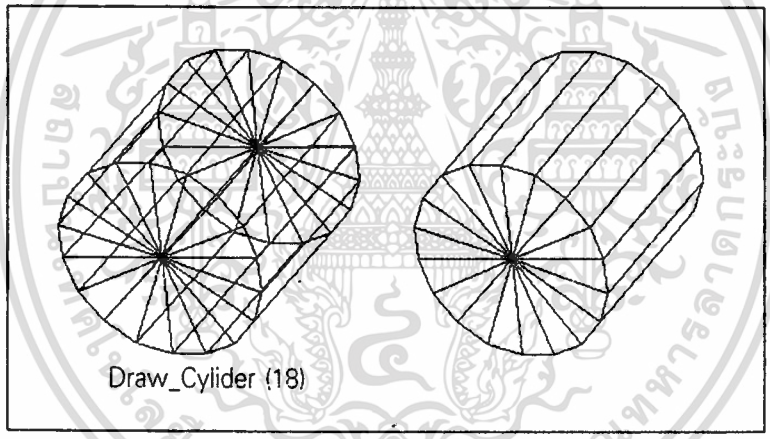
Get_Node () คือ ส่วนที่ทำหน้าที่รับข้อมูลจากผู้ใช้

ความยากง่ายของกระบวนการป้อนข้อมูลนั้นจะขึ้นอยู่กับส่วนรับข้อมูล ผู้ที่เขียนส่วนใช้งานจะต้องกำหนดวิธีการและอุปกรณ์ที่ใช้ในการป้อนข้อมูลให้เหมาะสมกับงานมากที่สุด อาจจะเป็นการรับข้อมูลทางแป้นพิมพ์ การกำหนดตำแหน่งโดยใช้ mouse หรือถ้าเป็นวัตถุซึ่งมีรูปทรงที่ซับซ้อนก็จะใช้ อุปกรณ์อ่านค่าตำแหน่งจุดบนวัตถุจริงเลยก็ได้

การเตรียมข้อมูลภาพของวัตถุที่มีรูปทรงเรขาคณิต ค่าตำแหน่งต่าง ๆ ก็จะหาได้จากการคำนวณ ดังตัวอย่างวัตถุรูปทรงกระบอกในภาพที่ 44 จะใช้วิธีคำนวณตำแหน่งมุมของรูปสี่เหลี่ยมผืนผ้าเล็ก ๆ ที่มาเรียงต่อกันเป็นพื้นผิวของทรงกระบอก

ภาพที่ 44

วัตถุภาพที่ได้จากการคำนวณ



เมื่อกำหนดจำนวนพื้นผิวสี่เหลี่ยม (num) ก็สามารถคำนวณตำแหน่งต่าง ๆ ของรูปทรงกระบอกที่มีรัศมีเท่ากับ 0.5 และความยาวเท่ากับ 1 ได้ดังนี้

```
void  Draw_Cylinder (int num)
{
        float x0, y0, x1,y1,degree;
        Vector p[4], p1[3], p2[3];
        x0 = 0.5;
        y0 = 0;
        Do_Vector (0, 0,-0.5, p1[0]);                    // จุดกลางพื้นผิวหัวท้าย
        Do_Vector (0, 0, 0.5, p2[0]);
```

```
        degree = 360/num;
        while  (degree <= 360) {
                x1 = 0.5*Cos (degree);                    // คำนวณตำแหน่ง
                y1 = 0.5*Sin (degree);
                Copy_Vector (x0, y0, -0.5, p[0]);
                Copy_Vector (x1, y1,  0.5, p[1]);
                Copy_Vector (x1, y1,  0.5, p[2]);
                Copy_Vector (x0, y0, -0.5, p[3]);
                Polygon_Abs (4, p[0]);          // สร้างรูปสี่เหลี่ยม
                Copy_Vector (p[0], p1[1]);
                Copy_Vector (p[3], p1[2]);
                Polygon_Abs (3, p1[0]);                   // พื้นผิวหัวท้าย
                Copy_Vector (p[2], p2[1]);
                Copy_Vector (p[1], p2[2]);
                Polygon_Abs (3, p2[0]);
                degree = degree + 360.0/num;
                x0 = x1; y0 = y1;
        }
}
```

นอกจากการคำนวณแล้ว ก็อาจจะเขียนเป็นส่วนการทำงาน Load_Object( ) สำหรับอ่านข้อมูลจากแฟ้มข้อมูลวัตถุภาพ ซึ่งจะต้องมีการกำหนดรูปแบบการเก็บข้อมูลเพื่อความสะดวกในการเตรียมแฟ้มข้อมูล ซึ่งประกอบด้วย คุณสมบัติของวัตถุ ข้อมูลตำแหน่งจุดทั้งหมด และตำแหน่งของพื้นผิวต่าง ๆ ดังตัวอย่างในภาพที่ 45 เป็นภาพที่ได้จากข้อมูลของรูปเหลี่ยมลูกบาศก์ตัดมุมในแฟ้ม "CUBE.DAT" ซึ่งมีรายละเอียดดังนี้

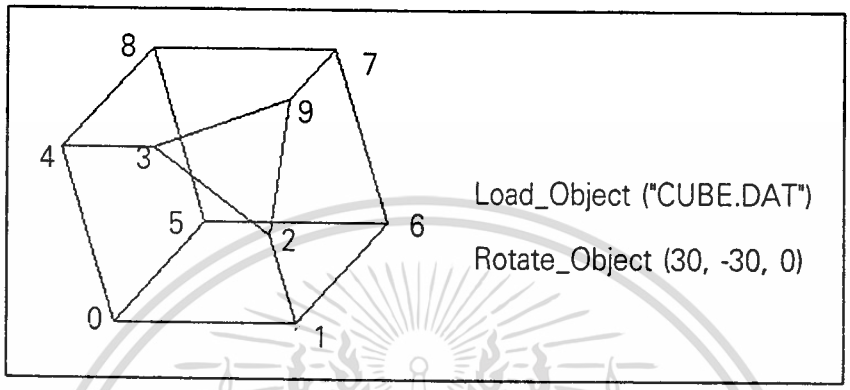
```
OBJECT
NODE        10      2       0       0       0       // จำนวนจุด,ขนาด
            -1      -1      -1                      // ตำแหน่งจุด
```

```
1       -1      -1
1       0       -1
0       1       -1
-1      1       -1
-1      -1      1
1       -1      1
1       1       1
-1      1       1
1       1       0

POLYGON 7                                       // จำนวนหน้า
        5       0       1       2       3       4       // หน้า 1
        4       8       7       6       5               // หน้า 2
        4       0       4       8       5
        5       4       3       9       7       8
        3       3       2       9
        5       2       1       6       7       9
        4       1       0       5       6

COLOR   15                                      // สีของวัตถุ

END OBJECT
```

ส่วนแรกของแฟ้มคือ NODE เป็นส่วนของข้อมูลกำหนดตำแหน่งจุดที่มีอยู่ในวัตถุภาพนั้น ในบรรทัดแรกจะเป็นจำนวนจุด ขนาดของวัตถุ และจุดศูนย์กลางของวัตถุ บรรทัดต่อมาก็จะเป็นค่าบอกตำแหน่งของจุดทั้งหมด ส่วน POLYGON เป็นส่วนที่กำหนดว่า พื้นผิวแต่ละอันประกอบด้วยจุดใดบ้าง ตัวเลขถัดมาคือ จำนวนพื้นผิวทั้งหมด สำหรับข้อมูลในบรรทัดต่อมา ตัวเลขตัวแรกคือจำนวนมุมของพื้นผิวที่เป็นรูปหลายเหลี่ยม ตามด้วยเลขลำดับของจุดที่กำหนดไว้ในส่วน NODE ปิดท้ายด้วย COLOR ซึ่งกำหนดค่าสีของวัตถุ

ภาพที่ 45

ภาพจากข้อมูลในแฟ้มข้อมูลวัตถุภาพ



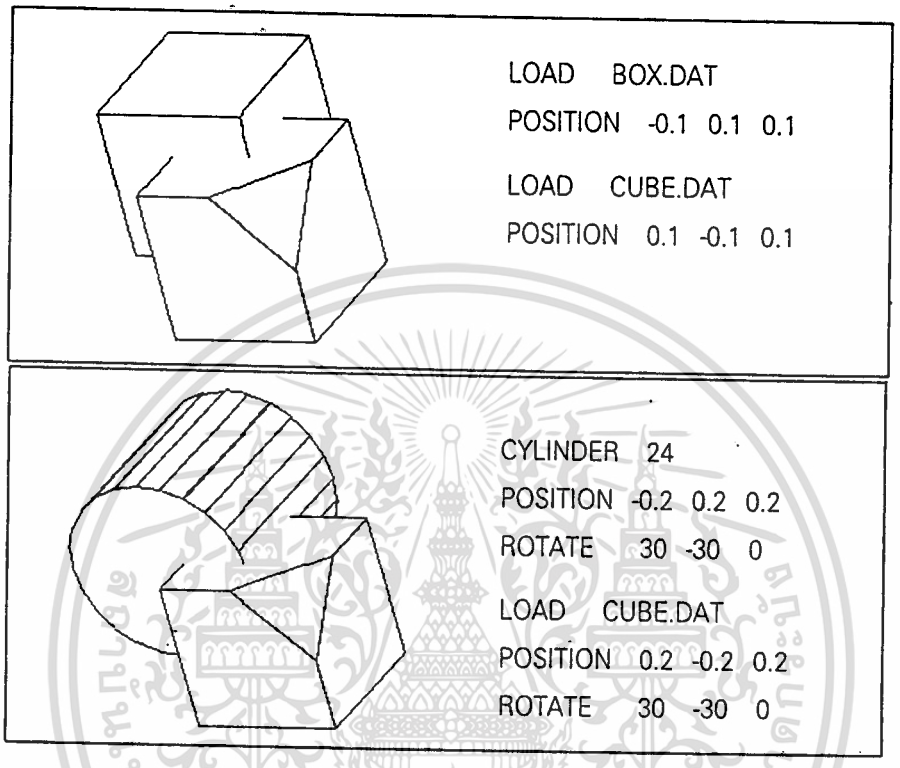
ประโยชน์ของการแยกข้อมูลออกเป็นส่วน NODE และ POLYGON ก็คือ ช่วยลดการซ้ำซ้อนเนื่องจากการใช้ตำแหน่งร่วมกันของพื้นผิวต่าง ๆ เป็นการป้องกันความสับสนในกรณีที่มีข้อมูลจำนวนมาก และการตรวจสอบแก้ไขก็ทำได้ง่าย ตัวอย่างส่วนการทำงานสำหรับการเรียกใช้แฟ้มข้อมูลภาพจะเป็นดังนี้

```
void Load_Object (char *s)
{
        FILE    *fo;
        char    cmd[16];
        Vector  *v, *p, cg;
        int     i,j,k,nv,nf,nn, color;
        float   size;
        if ((fo=fopen(s,"rt"))==NULL) return;
        fscanf (fo, "%s", cmd);                          // ตรวจสอบแฟ้ม
        if (strcmp(cmd,"OBJECT")) {
                fclose (fo);    return;  }
        New_Object ();                                   // เตรียมเนื้อที่
        do {
                fscanf (fo,"%s", cmd);
```

```
        if (!strcmp(cmd,"COLOR")) {                    // ข้อมูลสี
                fscanf(fo,"%d",&color);
                Color_Object (color);
        }
        if (!strcmp(cmd,"NODE")) {                     // ข้อมูลตำแหน่ง
                fscanf(fo,"%d %f %f %f %f",
                        &nv, &size, &cg[0],&cg[1],&cg[2]);
                if (size!=0) size = 1.0 / size;
                v=(Vector *) malloc(nv*sizeof(Vector));
                for (i=0; i<nv; i++) {
                        fscanf (fo, "%f %f %f",&v[i][0], &v[i][1], &v[i][2]);
                        Sub_Vector (v[i], cg, v[i]);
                        Scale_Vector (size, v[i], v[i]);
                }
        }
        if (!strcmp(cmd,"POLYGON")) {          //ข้อมูลพื้นผิว
                fscanf(fo,"%d",&nf);
                for (i=0; i<nf; i++) {
                        fscanf(fo, "%d", &nn);
                        p =(Vector*) malloc (nn*sizeof(Vector));
                        for (j=0; j<nn; j++) {
                                fscanf(fo, "%d", &k);
                                Copy_Vector(v[k], p[j]);
                        }
                        Polygon_Abs (nn, p[0]);  // สร้างรูปเหลี่ยม
                        free(p);
                }
        }
    } while (strcmp(cmd,"END"));                       // จบแฟ้มข้อมูล
    free (v); fclose (fo);
}
```

ภาพที่ 46

ภาพที่ได้จากการใช้แฟ้มองค์ประกอบภาพ



เมื่ออ่านข้อมูลเบื้องต้นของวัตถุภาพแล้ว ก็ต้องกำหนดตำแหน่ง ขนาด และมุมของวัตถุนั้น โดยใช้คำสั่งพื้นฐานดังตัวอย่างต่อไปนี้

```
Load_Object ("CUBE.DAT");              // อ่านข้อมูลจากแฟ้ม
Translate_Object (-0.2, 0.2, 0.2);     // กำหนดตำแหน่ง
Scale_Object (0.8, 0.8, 0.8);          // กำหนดขนาด
Rotate_Object (30,-30, 0);             // กำหนดมุม
```

ในกรณีที่ต้องการความสะดวกในการใช้งานอีกระดับหนึ่ง ก็อาจจะสร้างเป็นแฟ้มองค์ประกอบภาพ ซึ่งประกอบด้วยคำสั่งกำหนดที่มาของวัตถุภาพ ตัวอย่างเช่น

```
LOAD      CUBE.DAT              // กำหนดวัตถุภาพ
POSITION  -0.2   0.2   0.2     // กำหนดตำแแหน่ง
SCALE     0.8    0.8   0.8     // กำหนดขนาด
ROTATE    30     -30   0       // กำหนดมุม
```

ในการใช้แฟ้มองค์ประกอบภาพ สามารถรวมวัตถุภาพจากที่มาต่าง ๆ กันได้ ดังตัวอย่างในภาพที่ 46 จะมีทั้งวัตถุภาพที่ได้จากแฟ้มข้อมูลภาพ และวัตถุภาพที่ได้จากการคำนวณ โดยส่วนการทำงานสำหรับอ่านแฟ้มองค์ประกอบภาพจะมีลักษณะดังนี้

```
void Get_Image (char *s)
{
        FILE    *fs;
        char    cmd[16].  name[12];
        float   x,y,z;
        int     vl,vr,vt,vb,n;
        if ((fs=fopen(s,"rt"))==NULL) return;
        fscanf (fs, "%s", cmd);
        if (strcmp(cmd,"IMAGE")) {                    //ตรวจสอบแฟ้มภาพ
                fclose (fs); return; }
        do {
                fscanf (fs,"%s", cmd);
                if (!strcmp(cmd,"LOAD")) {
                        fscanf (fs, "%s", name);
                        Load_Object (name);           // อ่านข้อมูลภาพ
                }
                if (!strcmp(cmd,"CYLINDER")) {
                        fscanf (fs, "%d", &n;
                        Draw_Cylinder (n);            //รูปทรงกระบอก
                }
                if (!strcmp(cmd,"POSITION")) {
                        fscanf (fs, "%f %f %f",&x, &y, &z);
                        Translate_Object (x, y, z);   // กำหนดตำแหน่ง
                }
                if (!strcmp(cmd,"ROTATION")) {
                        fscanf (fs, "%f %f %f",&x, &y, &z);
```

```
                Rotate_Object (x, y, z);                    // หมุนวัตถุ
            }
            if (!strcmp(cmd,"SCALE")) {
                fscanf (fs, "%f %f %f",&x, &y, &z);
                Scale_Object (x, y, z);                     // เปลี่ยนขนาดวัตถุ
            }
        } while (strcmp(cmd,"END"));                        // จบแฟ้มภาพ
        fclose (fs);
    }
```

ในกรณีของภาพโต๊ะสี่เหลี่ยมซึ่งประกอบด้วยชิ้นส่วนทั้งหมด 9 ชิ้น แต่ละชิ้นมีรูปทรงเหมือนกัน แต่มีขนาดต่างกัน อาจจะใช้วิธีดึงข้อมูลของวัตถุรูปสี่เหลี่ยมลูกบาศก์ในแฟ้ม BOX.DAT มาทำการเปลี่ยนขนาด และกำหนดตำแหน่งให้ถูกต้องสำหรับชิ้นส่วนแต่ละชิ้น ตัวอย่างเช่น ชิ้นส่วนของพื้นโต๊ะก็จะเป็นดังนี้

```
LOAD        BOX.DAT                     // พื้นโต๊ะ
SCALE       0.6     0.08    0:8
POSITION    0.0     0.2     0.0
LOAD        BOX.DAT                     // ขาโต๊ะ
SCALE       0.08    0.4     0.08
POSITION    0.26    0.04    0.36
```

เมื่อกำหนดที่มาของข้อมูลภาพได้แล้ว ขั้นตอนการทำงานหลักของส่วนใช้งานจะเป็นดังนี้

```
Get_Image ( );                          // เตรียมข้อมูลภาพ
while (endloop==0) {
    Clear_Display ( );                  // ลบจอภาพ
    Make_Picture ( );                   // แสดงภาพ
    endloop=View_Adjust ( );            // ปรับวิธีแสดงภาพ
}
```

ส่วน View_Adjust ( ) คือ ส่วนควบคุมรูปแบบการแสดงภาพ ซึ่งจะพูดถึงในส่วนต่อไป

ส่วนควบคุมการแสดงภาพ

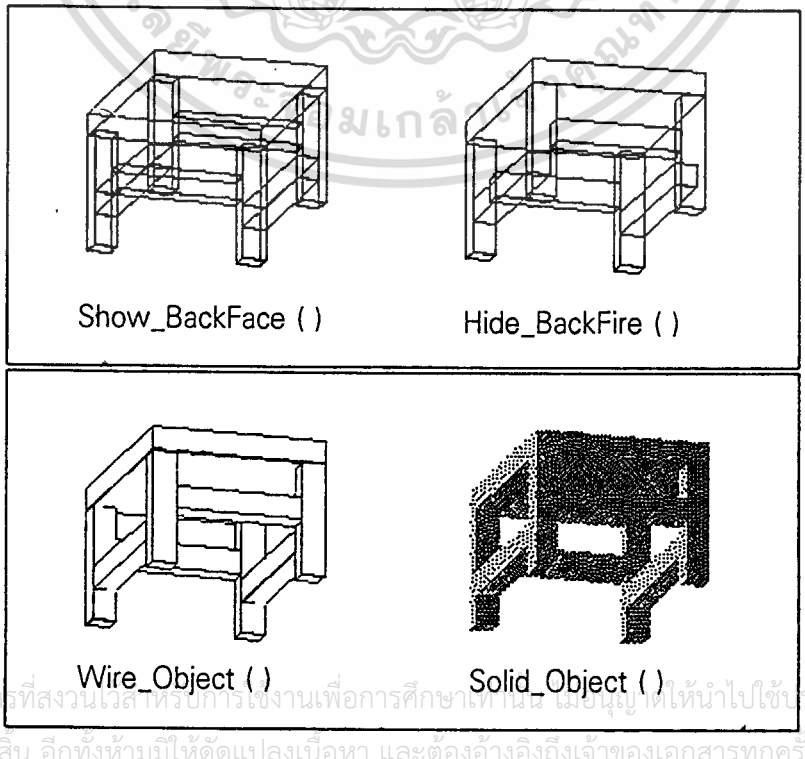
เมื่อผ่านขั้นตอนการเตรียมข้อมูล จนได้ข้อมูลภาพซึ่งมีการกำหนดขนาด ตำแหน่ง และมุมของวัตถุภาพทั้งหมดแล้ว ก็มาถึงส่วนที่สองของการใช้งาน นั่นคือ ส่วนควบคุมการแสดงภาพ ซึ่งกำหนดรูปแบบของภาพก่อนการแสดงภาพ ดังนี้

```
endloop=View_Adjust ();                    // ควบคุมการแสดงภาพ
if (!endloop) {
        Clear_Display ();                  // ลบพื้นที่แสดงภาพ
        Make_Picture ();                   // แสดงภาพ
}
```

การกำหนดรูปแบบของภาพ จะเริ่มจากการการกำหนดลักษณะของภาพ โดยใช้คำสั่งกำหนดรูปแบบการแสดงภาพต่าง ๆ ซึ่งจะให้ภาพดังตัวอย่างในภาพที่ 47

ภาพที่ 47

ภาพเปรียบเทียบรูปแบบการแสดงภาพ



ตัวอย่างของส่วนการทำงาน View_Adjust ( ) สำหรับควบคุมการแสดงภาพเป็นดังนี้

```
int View_Adjust ()                          // ส่วนควบคุมการแสดงภาพ
{
  static int solid=0, clip=0, hide=0;
  char c;
  while (True) {                            // รอคำสั่ง
       while (!kbhit());                    // ตรวจสอบแป้นพิมพ์
       c=getch();                           // รับคำสั่ง
       if (c==13) return (1);               // จบการทำงาน
       if ((c=='s')||(c=='S')) {            // เปลี่ยนรูปแบบการแสดงภาพ
            if (solid)
                 {Wire_Object (); solid=0;}
            else {Solid_Object (); solid=1;}
       }
       if ((c=='h')||(c=='H')) {            // ควบคุมการแสดงพื้นผิวด้านหลัง
            if (hide)
                 {Show_BackFace (); hide=0;}
            else {Hide_BackFace (); hide=1;}
       }
}
```

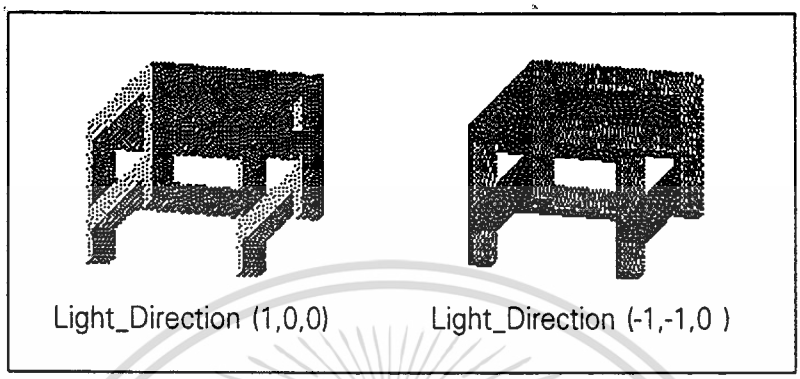
ในส่วนการทำงานนี้ สามารถใส่คำสั่งควบคุมการแสดงภาพอื่น ๆ ได้ในทำนองเดียวกัน ตัวอย่าง เช่น การกำหนดทิศทางของแสงในการแสดงภาพพื้นผิวทึบ ก็จะใช้คำสั่ง

```
Light_Direction (-1,-1,0)
```

เป็นคำสั่งกำหนดให้แสงมาจากทางด้านบนซ้ายของภาพ ซึ่งจะได้เป็นภาพทางด้านขวาของภาพที่ 48 ส่วนภาพทางด้านซ้ายเป็นภาพที่กำหนดให้แสงมาทางด้านขวาของภาพ

ภาพที่ 48

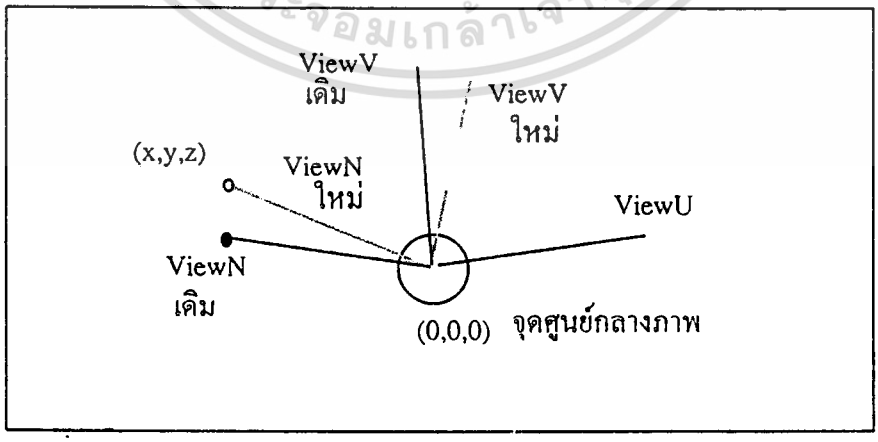
ภาพแสดงการเปลี่ยนทิศทางของแสง



ขั้นตอนสุดท้ายก่อนที่จะสั่งให้แสดงภาพก็คือ การกำหนดตำแหน่งของผู้มองภาพ โดยการปรับค่าตำแหน่งของผู้มองภาพ และทิศทางการมองภาพ เหมือนกับการควบคุมกล้องถ่ายภาพ ในกรณีที่ต้องการให้มีการควบคุมจากผู้ใช้โดยตรง อาจจะใช้วิธีหมุนตำแหน่งของผู้มองภาพไปรอบวัตถุภาพ แล้วปรับทิศทางการมองภาพให้มองเข้าหาจุดศูนย์กลางของภาพตลอดเวลา ภาพที่ 49 แสดงการหาทิศทางของการมองภาพ เมื่อมีการเปลี่ยนตำแหน่งผู้มองภาพในแนวตั้ง

ภาพที่ 49

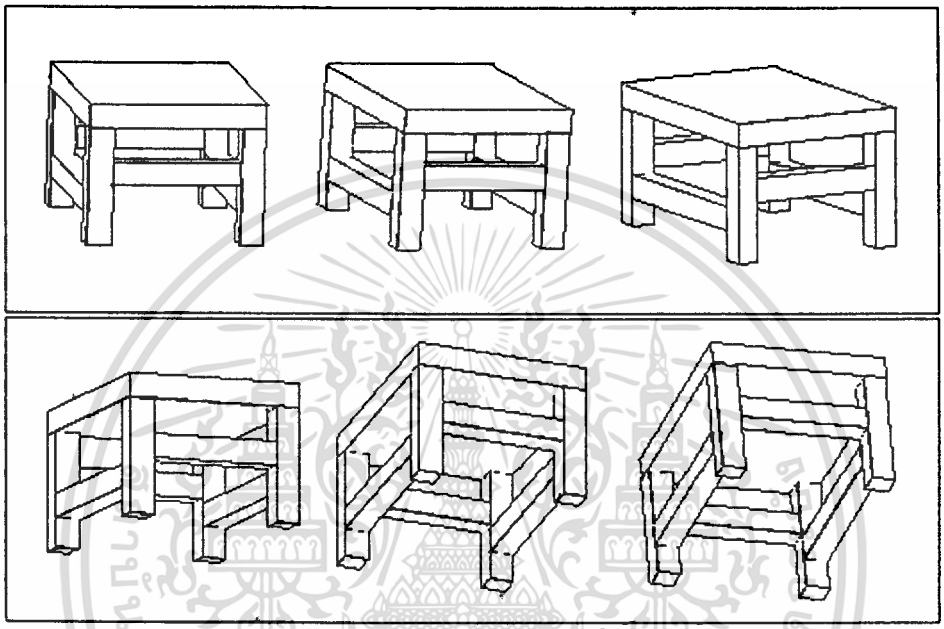
การหาทิศทางของการมองภาพ



ภาพที่ 50 เป็นภาพที่ได้จากการเปลี่ยนตำแหน่งของผู้มองภาพ โดยหมุนไปรอบแกน X และแกน Y แล้วมองไปที่ตำแหน่ง (0,0,0)

ภาพที่ 50

ภาพแสดงการเปลี่ยนตำแหน่งของผู้มองภาพ



การหาตำแหน่งและทิศทางการมองภาพ เมื่อกำหนดมุมรอบแกน x และแกน Y เป็น deg1 และ deg2 ตามลำดับ จะทำได้ดังนี้

```
y = Sin(deg2);
x = Sin(deg1)*Cos(deg2);
z = -Cos(deg1)*Cos(deg2);
View_Position (x, y, z);                    // ตำแหน่งผู้มองภาพ
Do_Vector (-x, -y, -z, v);                  // ทิศทางของผู้มองภาพ
Copy_Vector (v, ViewN);
if (HorChange)                              // เปลี่ยนมุมในแนวนอน
        Cross_Vector (ViewN, ViewV, ViewU);
if (VerChange)                              // เปลี่ยนมุมในแนวตั้ง
        Cross_Vector (ViewU, ViewN, ViewV);
```

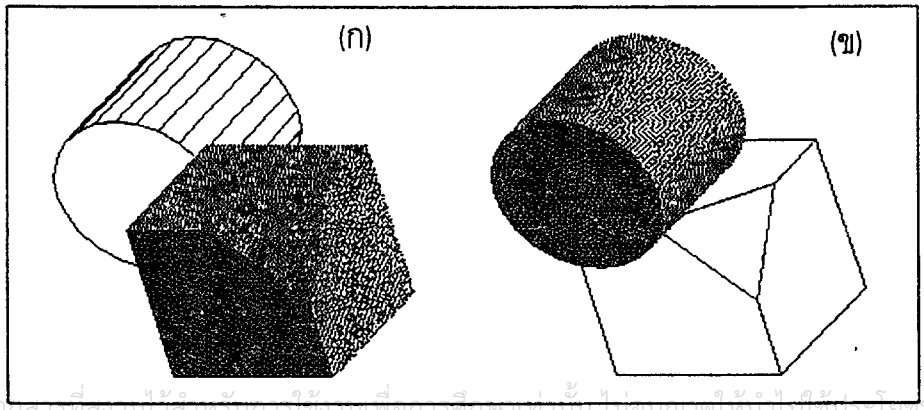
ในกรณีที่ไม่ต้องการควบคุมโดยตรง ก็อาจจะใช้แฟ้มควบคุมการแสดงภาพได้เช่นเดียวกัน การใช้งานเริ่มจากการอ่านข้อมูลจากแฟ้มภาพ ในตัวอย่างนี้จะใช้แฟ้มภาพที่ประกอบด้วยวัตถุภาพสองอย่าง ได้แก่ รูปทรงกระบอก และ รูปเหลี่ยมตัดมุม ดังนี้

```
IMAGE
CYLINDER    24                              // คำนวณรูปทรงกระบอก
        POSITION    -0.2    0.2     0.2
        ROTATION    30      -30     0
        SCALE       1       1       1

LOAD    CUBE.DAT                            // อ่านข้อมูลจากแฟ้มวัตถุ
        POSITION    0.2     -0.3    -0.2
        ROTATION    30      -30     0
        SCALE       0.8     0.8     0.8
END IMAGE
```

ตัวอย่างในภาพที่ 51 แสดงการใช้แฟ้มควบคุมการแสดงภาพ นำแฟ้มภาพซึ่งมีวัตถุภาพสองอย่างมาแสดงสองครั้ง ครั้งแรกให้แสดงวัตถุภาพชิ้นที่หนึ่ง ครั้งที่สองให้แสดงวัตถุภาพชิ้นที่สอง แต่ละครั้งกำหนดรูปแบบการแสดงภาพต่างกัน

ภาพที่ 51

ภาพแสดงการใช้แฟ้มควบคุมการแสดงภาพ



ในการแสดงภาพโดยใช้แฟ้มควบคุมการแสดงภาพ ส่วนแรกจะเป็นส่วนของการกำหนดรูปแบบการแสดงภาพ เช่น การกำหนดตารางสีที่ใช้ กำหนดพื้นที่แสดงภาพ ตามด้วยข้อมูลควบคุมการแสดงภาพอื่น ๆ จากนั้น ก็จะเป็นการกำหนดขั้นตอนการแสดงภาพ ดังตัวอย่างต่อไปนี้ เป็นขั้นตอนการแสดงภาพที่ 51 (ก)

```
DISPLAY
        PALETTE     GREY.PAL
        WINDOW      150  100  400  350     //พื้นที่แสดงภาพ
        VIEW        0    0    -1          //ตำแหน่งผู้มองภาพ
                    0    0    1           //ทิศทางการมอง
                    0    1    0           //แนวตั้ง
        LIGHT       -1   -1   0           //ทิศทางของแสง
        WIRE                             //แสดงภาพแบบโครงลวด
        HIDE        2                    //ไม่แสดงวัตถุหมายเลข 2
        CLEAR                            //ลบพื้นที่แสดงภาพ
        MAKE                             //แสดงภาพ
        SOLID                            //แสดงภาพแบบพื้นผิวทึบ
        SHOW        2                    //แสดงวัตถุหมายเลข 2
        HIDE        1                    //ไม่แสดงวัตถุหมายเลข 1
        MAKE                             //แสดงภาพ
END DISPLAY
```

การเปลี่ยนจากการควบคุมโดยตรงมาเป็นการควบคุมด้วยแฟ้มแสดงภาพ ก็ต้องเปลี่ยนจากส่วนการทำงาน View_Adjust () ในส่วนการทำงานหลัก มาใช้ส่วนการทำงาน Load_Display () ซึ่งมีรายละเอียดดังนี้

```
void Load_Display (char *s)
{
        FILE    *fs;
        char    cmd[16],prm[16];
```

```
float   x,y,z;
int     vl,vr,vt,vb;n,color;

if ((fs=fopen(s,"rt"))==NULL) return;          // เปิดแฟ้มควบคุม
fscanf (fs, "%s", cmd);
if (strcmp(cmd,"DISPLAY")) {fclose(fs); return;}
do {
        fscanf (fs,"%s", cmd);
        if (!strcmp(cmd,"WINDOW")) {          // กำหนดพื้นที่แสดงภาพ
                fscanf (fs, "%d %d %d %d",&vl, &vt, &vr, &vb);
                Set_Window (vl, vt, vr, vb);
        }
        if (!strcmp(cmd,"PALETTE")) {         // เลือกตารางสี
                fscanf (fs, "%s",prm);
                Set_Color (prm);
        }
        if (!strcmp(cmd,"LIGHT")) {           // กำหนดทิศทางแสง
                fscanf (fs, "%f %f %f",&x, &y, &z);
                Light_Direction (x, y, z);
        }
        if (!strcmp(cmd,"OBJECT")) {          // เลือกวัตถุภาพ
                fscanf (fs, "%d", &n);   Select_Object (n);
        }
        if (!strcmp(cmd,"VIEW")) {            // กำหนดมุมมองภาพ
                fscanf (fs, "%f %f %f",&x, &y, &z);
                View_Position (x, y, z);
                fscanf (fs, "%f %f %f",&x, &y, &z);
                View_Direction (x, y, z);
                fscanf (fs, "%f %f %f",&x, &y, &z);
                View_Up (x, y, z);
```

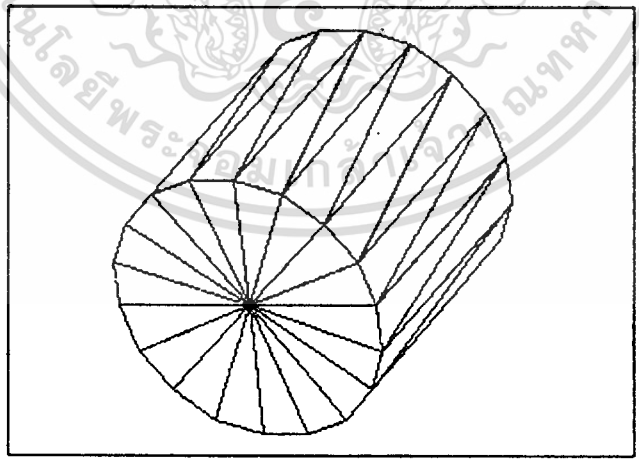
```
            }
            if (!strcmp(cmd,"SOLID"))  Solid_Object ();          //ลักษณะภาพ
            if (!strcmp(cmd,"WIRE"))   Wire_Object ();
            if (!strcmp(cmd,"HIDE"))   Hide_Object ();
            if (!strcmp(cmd,"SHOW"))   Show_Object ();
            if (!strcmp(cmd,"CLEAR"))  Clear_Display ();         //ลบพื้นที่
            if (!strcmp(cmd,"MAKE"))   Make_Picture ();          //แสดงภาพ
        } while (strcmp(cmd,"END"));                             //จบการแสดงภาพ
        fclose (fs);
}
```

การแยกแฟ้มข้อมูลออกจากแฟ้มแสดงภาพ เพื่อให้ได้ส่วนเตรียมข้อมูลและส่วนการแสดงภาพที่เป็นอิสระต่อกัน สามารถเลือกกระบวนการให้เหมาะกับการใช้งาน และแก้ไขเพิ่มเติมได้โดยไม่มีผลกระทบถึงกัน

บทที่ 7
การสร้างภาพพื้นผิววัตถุ

จากการกำหนดให้ระบบภาพรับข้อมูลภาพเป็นคำสั่งสร้างรูปหลายเหลี่ยม ดังนั้นส่วนใช้งานจะต้องทำหน้าที่ทำข้อมูลของพื้นผิววัตถุให้เป็นรูปเหลี่ยม เนื่องจากต้องมีการเก็บข้อมูลของรูปเหลี่ยมทั้งหมดในขั้นตอนการประมวลผล จากตัวอย่างก่อนหน้านี้ มีการใช้ทั้งรูปสามเหลี่ยม รูปสี่เหลี่ยมและรูปหลายเหลี่ยม ซึ่งจะเป็นปัญหาในการกำหนดโครงสร้างข้อมูล เนื่องจากต้องกำหนดให้มีเนื้อที่พอสำหรับการเก็บรูปเหลี่ยมที่มีจำนวนเหลี่ยมมากที่สุด โดยทั่วไป จะใช้รูปสามเหลี่ยมเป็นองค์ประกอบพื้นฐานของพื้นผิวต่าง ๆ ไม่ว่าจะเป็นพื้นผิวเรียบ หรือ พื้วผิวโค้ง เพื่อประสิทธิภาพในการเก็บข้อมูล ดังตัวอย่างในภาพที่ 52 พื้นผิวด้านข้างซึ่งเป็นรูปสี่เหลี่ยมจะถูกแบ่งเป็นรูปสามเหลี่ยมสองรูป ส่วนพื้นผิวหัวท้ายซึ่งเป็นรูปหลายเหลี่ยมก็จะถูกแบ่งจากจุดศูนย์กลาง

ภาพที่ 52
การแบ่งพื้นที่รูปเหลี่ยมเป็นรูปสามเหลี่ยม

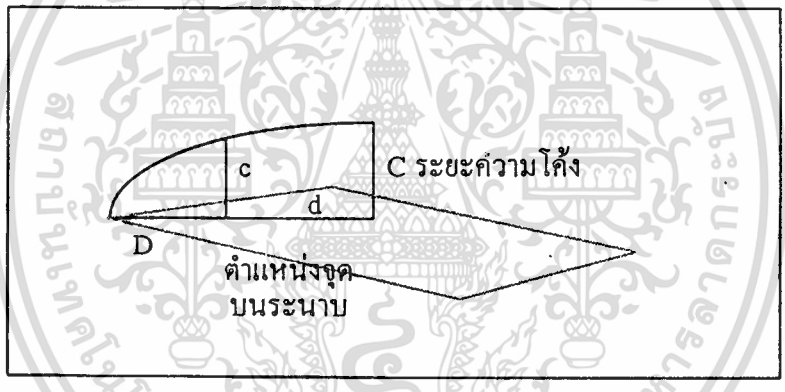


คุณสมบัติอย่างหนึ่งที่ทำให้ได้ภาพของวัตถุใกล้เคียงกับวัตถุจริงก็คือ ลักษณะของภาพพื้นผิววัตถุ ซึ่งมีองค์ประกอบตั้งแต่ รูปทรงของพื้นผิว (Form) ลวดลาย (Texture) การสะท้อนแสง (Reflection) แต่สิ่งสำคัญที่สุดในการแสดงภาพแบบโครงร่าง ก็คือ รูปทรงของพื้นผิว ซึ่งในที่นี้ จะแสดงการสร้างพื้นผิวสองแบบ คือ พื้นผิวโค้ง และ พื้นผิวไม่เรียบ

การสร้างภาพพื้นผิวโค้ง

ในกรณีของพื้นผิวโค้ง มีวิธีการสร้างได้หลายวิธี อาจจะใช้วิธีลากเส้นโค้งจากข้อมูลตำแหน่งจุดบนผิวโค้งโดยใช้ B_Splines Curve [11] แต่ในที่นี้ จะแสดงวิธีสร้างพื้นผิวโค้งจากพื้นราบใด ๆ โดยการกำหนดระยะความโค้งจากระนาบเดิมและตำแหน่งที่มีความโค้งสูงสุด ซึ่งโดยทั่วไปจะใช้จุดกึ่งกลางของพื้นผิวนั้น ดังภาพที่ 53

ภาพที่ 53
การสร้างพื้นผิวโค้ง



จากตำแหน่งบนพื้นราบ สามารถจะหาตำแหน่งบนพื้นผิวโค้งได้จากสมการที่ (9) ซึ่งเป็นสมการของรูปวงรี ดังนี้

$$(d/D)^2 + (c/C)^2 = 1 \qquad (9)$$

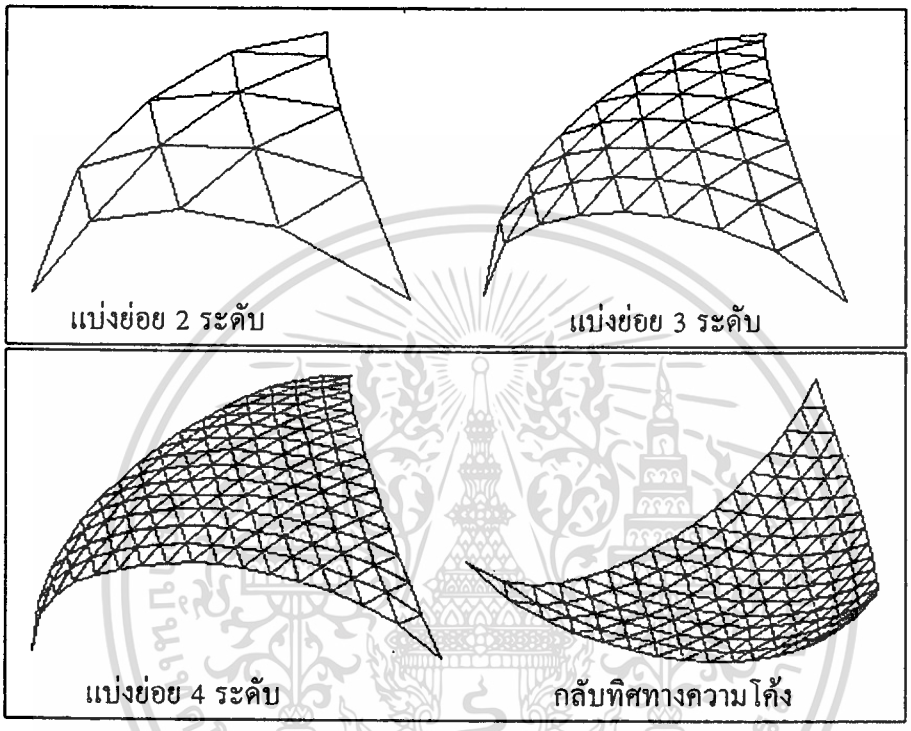
โดยที่ d เป็นระยะห่างจากศูนย์กลางของพื้นผิวเรียบ D เป็นระยะกว้างสุดของพื้นผิวโค้ง และ C เป็นระยะห่างของพื้นผิวโค้งจากระนาบเดิม ดังนั้น จะได้ระยะ c ดังสมการที่ (10)

$$c = \mathrm{sqrt}\,(1 - (d/D)^2) * C \qquad (10)$$

และถ้าเปลี่ยนจากระยะ d ให้เป็น -d ก็สามารถหาพื้นผิวโค้งที่โค้งกลับไปในทิศทางตรงกันข้ามได้ เมื่อทราบวิธีการหาระยะทางจากจุดบนระนาบไปยังพื้นผิวโค้งแล้ว ขั้นตอนการสร้างพื้นผิวโค้งเริ่มจากการแบ่งย่อยพื้นผิวราบเป็นรูปสามเหลี่ยมซึ่งเป็นองค์ประกอบหลัก แล้วนำรูปสามเหลี่ยมแต่ละรูปมาสร้างเป็นพื้นผิวโค้ง ดังภาพที่ 54

ภาพที่ 54

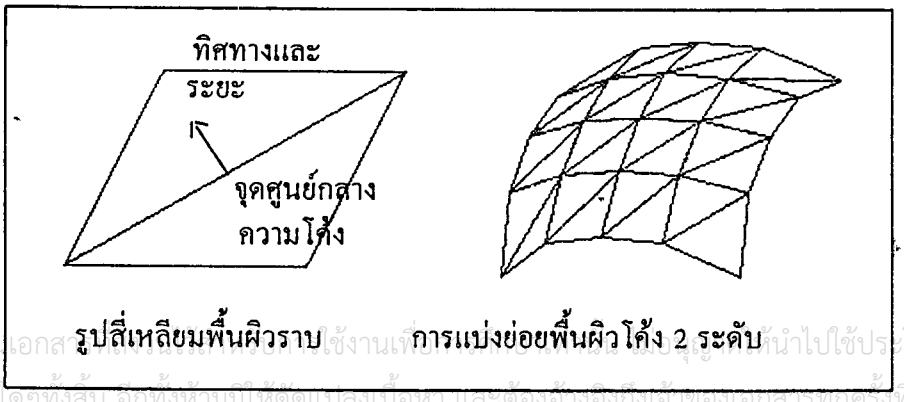
ภาพของพื้นผิวโค้งจากรูปสามเหลี่ยม



ในการสร้างพื้นผิวโค้งจากรูปสามเหลี่ยม จะเริ่มจากการแบ่งย่อยพื้นที่ออกเป็นรูปสามเหลี่ยมเล็ก ๆ กระบวนการในการแบ่งย่อยรูปสามเหลี่ยมจะใช้วิธีเชื่อมจุดแบ่งครึ่งด้านทั้งสามเข้าด้วยกันได้เป็นรูปสามเหลี่ยมสี่รูป จากรูปสามเหลี่ยมย่อยแต่ละรูปก็จะแบ่งย่อยด้วยวิธีเดียวกันซ้ำจนถึงระดับที่ต้องการ

ภาพที่ 55

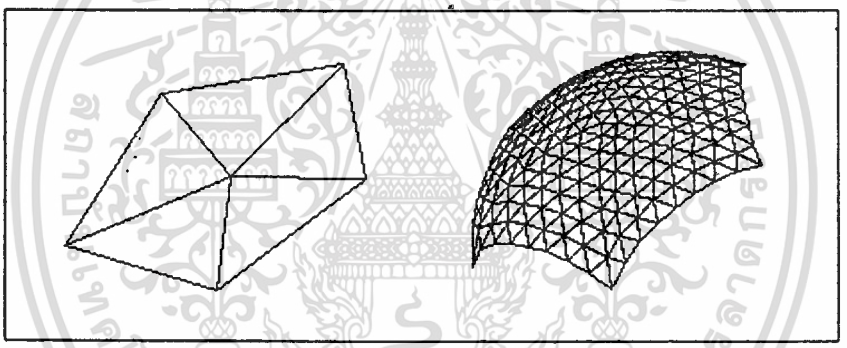
การสร้างภาพพื้นผิวโค้งจากรูปสี่เหลี่ยมพื้นเรียบ



งานขั้นตอนต่อคือ เปลี่ยนเป็นตำแหน่งของรูปสามเหลี่ยมบนพื้นผิวราบให้เป็นตำแหน่งบนพื้นผิวโค้ง แล้วหาระยะห่างของตำแหน่งจุดในรูปสามเหลี่ยมย่อยบนพื้นราบไปยังส่วนโค้งที่ต้องการตามวิธีการข้างต้น โดยใช้จุดกึ่งกลางของพื้นผิวเป็นจุดศูนย์กลางความโค้ง

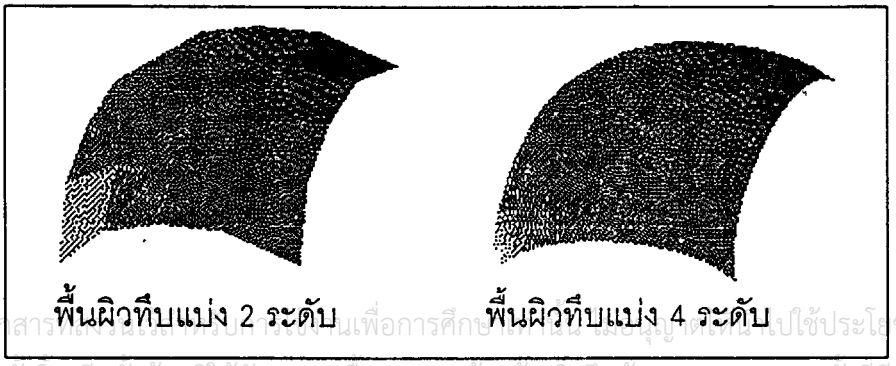
ในกรณีของพื้นที่รูปสี่เหลี่ยมก็จะแบ่งเป็นรูปสามเหลี่ยมสองรูป แต่จะใช้จุดศูนย์กลางของรูปสี่เหลี่ยมเป็นหลักในการหาระยะความโค้ง ซึ่งจะได้ผลลัพธ์ดังภาพที่ 55 ถ้าเป็นรูปเหลี่ยมทั่วไป จะใช้วิธีแบ่งเป็นรูปสามเหลี่ยมโดยใช้จุดศูนย์กลางของรูปเหลี่ยมนั้นเป็นหลัก แล้วใช้จุดนั้นเป็นจุดศูนย์กลางความโค้งดังภาพที่ 56

ภาพที่ 56
ภาพของพื้นผิวโค้งจากรูปหลายเหลี่ยมทั่วไป



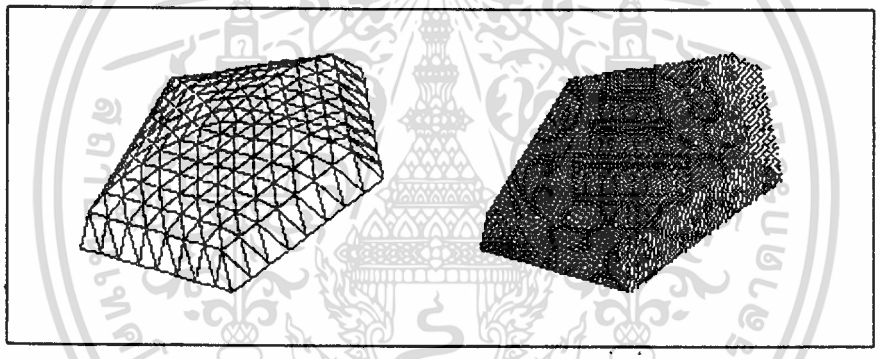
ที่กล่าวมาแล้ว เป็นขั้นตอนของการเตรียมข้อมูลภาพ ในส่วนของการแสดงภาพ นอกจากจะแสดงภาพแบบโครงร่างแล้ว ก็สามารถแสดงภาพของส่วนโค้งเป็นพื้นผิวทึบได้ดังภาพที่ 57 จะเห็นได้ว่าถ้าโครงร่างดีก็จะได้ภาพพื้นผิวทึบดีด้วย

ภาพที่ 57
ภาพพื้นผิวโค้งแบบทึบ



จากตัวอย่างที่ผ่านมา จะใช้ความกว้างของพื้นผิวคงที่การคำนวณหาระยะส่วนโค้ง ทำให้ขอบของพื้นผิวส่วนที่กว้างน้อยกว่า จะยกสูงขึ้นจากพื้นผิวเดิม ในกรณีที่ต้องการให้ขอบของพื้นผิวใหม่สัมผัสกับขอบของพื้นผิวเดิม จะต้องใช้ความกว้างของขอบจริงในทิศทางจากจุดศูนย์กลางมายังจุดที่จะหาระยะส่วนโค้ง ภาพที่ได้ก็จะเป็นดังภาพที่ 58

ภาพที่ 58
พื้นผิวโค้งขอบเรียบ

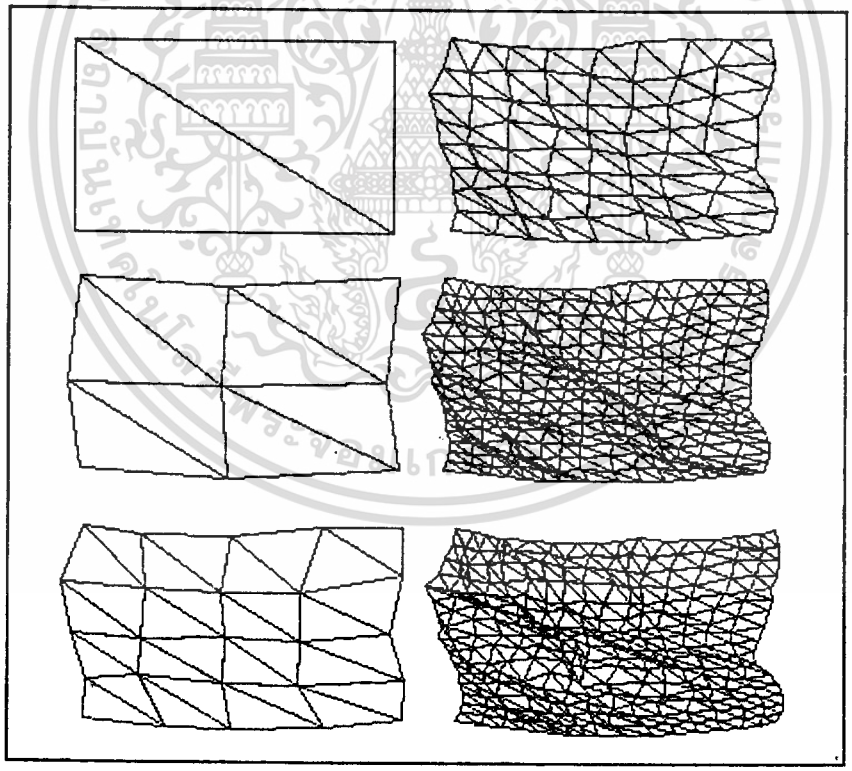


การใช้รัศมีความโค้งต่างกันจะทำให้ความโค้งของพื้นผิวแต่ละส่วนไม่ต่อเนื่อง แต่การที่ขอบพื้นผิวโค้งเป็นเส้นตรงที่อยู่บนระนาบเดียวกัน ทำให้สะดวกแก่การนำพื้นผิวนี้ไปเชื่อมต่อกับพื้นผิวอื่นมีขอบเป็นเส้นตรงเช่นเดียวกัน

การสร้างภาพพื้นผิวไม่เรียบด้วย Fractal Surfaces

การสร้างภาพจำลองของพื้นผิวอีกชนิดหนึ่ง ได้แก่ ภาพของพื้นผิวที่ไม่เรียบ ซึ่งสามารถสร้างโดยใช้ Fractal Surfaces [12-13] ด้วยการแบ่งย่อยพื้นผิวเป็นรูปสามเหลี่ยมเล็ก ๆ แล้วย้ายตำแหน่งจุดไม่ให้อยู่ในระนาบเดิมเหมือนกับการสร้างพื้นผิวโค้ง แต่ใช้ค่าเบี่ยงเบนเป็นค่าสุ่ม รูปสามเหลี่ยมเล็ก ๆ แต่ละรูปจึงมีขนาดและระนาบแน่นอน ดังภาพที่ 59 เป็นรูปพื้นผิวขรุขระที่ได้จากข้อมูลรูปสี่เหลี่ยมรูปเดียว

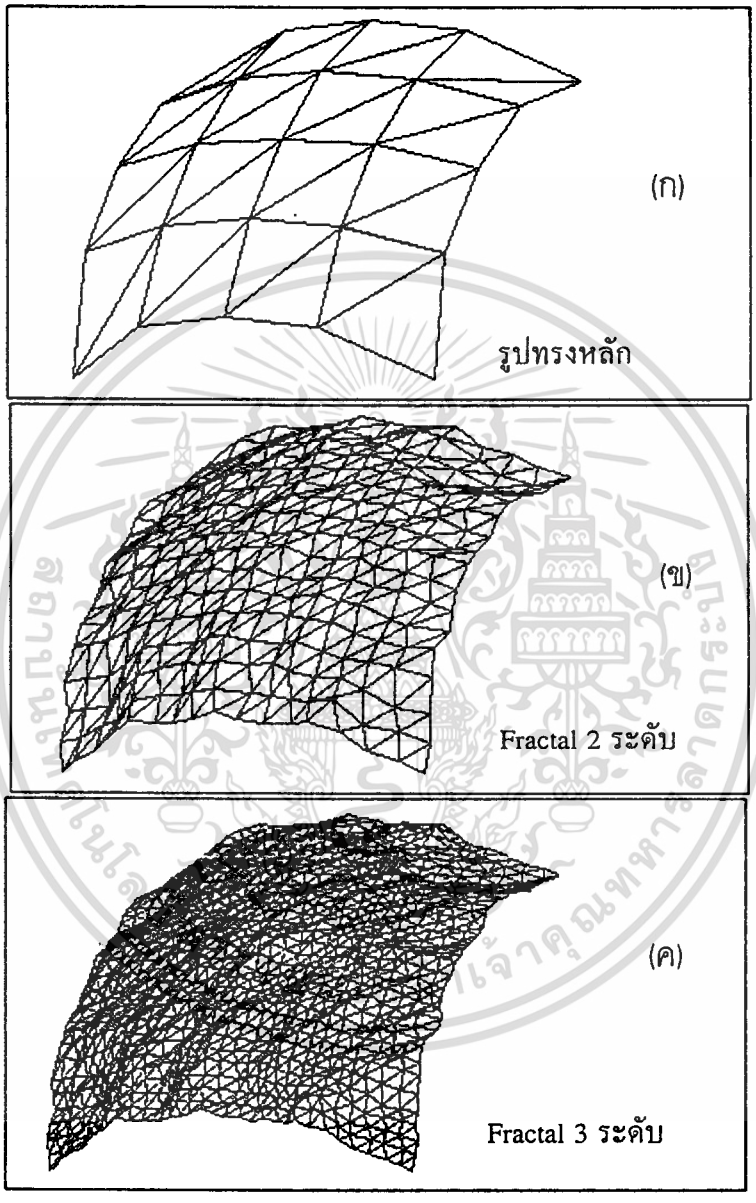
ภาพที่ 59
พื้นผิวไม่เรียบที่จำลองโดยใช้ Fractal Surfaces



ในการสร้างภาพจำลองของวัตถุใด ๆ จะต้องกำหนดรูปทรงของวัตถุก่อน เมื่อได้รูปทรงของวัตถุที่ต้องการแล้วจึงจะสร้างพื้นผิวที่ต้องการบนพื้นผิวแต่ละด้าน ดังตัวอย่างในภาพที่ 60 จะเริ่มจากการสร้างรูปทรงหลักเป็นพื้นผิวโค้งดังภาพที่ 60 (ก) จากนั้นก็จะนำรูปสามเหลี่ยมที่เป็นองค์ประกอบของพื้นผิวโค้งนั้น ไปสร้างเป็น Fractal Surfaces อีกทีหนึ่ง ซึ่งจะได้ผลลัพธ์ดังภาพที่ 60 (ข) และ (ค)

ภาพที่ 60

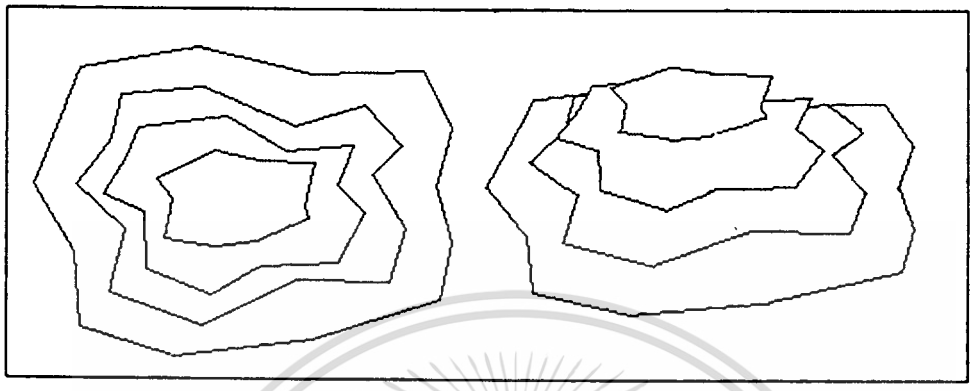
การสร้างพื้นผิวไม่เรียบบนรูปทรงหลัก



การกำหนดรูปทรงของวัตถุภาพ นอกจากจะใช้วิธีการคำนวณแล้ว สำหรับวัตถุซึ่งรูปทรงสามารถแทนด้วยสูตรคำนวณ ก็จะใช้วิธีอ่านข้อมูลตำแหน่งจุดจากวัตถุจริง ๆ หรือถ้าไม่มีวัตถุจริงอยู่ก็สามารถสร้างหุ่นจำลองของวัตถุนั้นขึ้นมาก่อน แล้วหาตำแหน่งจุดบนหุ่นจำลองนั้นอีกทีหนึ่ง ดังตัวอย่างในภาพที่ 61 เป็นตัวอย่างของข้อมูลบอกระดับ ซึ่งจะได้จากการสุ่มตำแหน่งบนวัตถุจริง แล้วนำตำแหน่งนั้นไปสร้างเป็นภาพของวัตถุอีกทีหนึ่ง

ภาพที่ 61

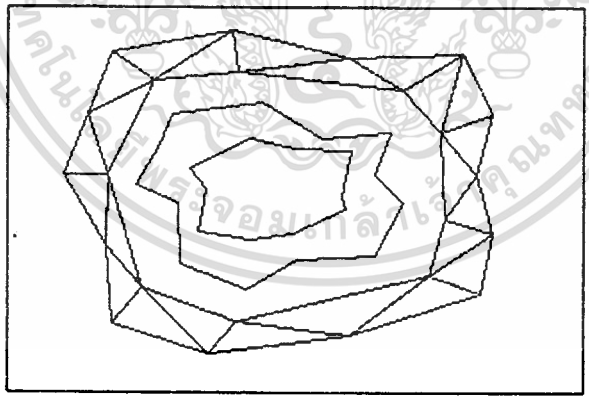
ข้อมูลบอกระดับ



การสร้างภาพพื้นผิวจากข้อมูลบอกระดับ ด้วยการเชื่อมต่อระหว่างข้อมูลสองระดับด้วยรูปสามเหลี่ยม โดยเชื่อมจุดสองจุดในระดับที่หนึ่งกับจุดในระดับที่สองซึ่งอยู่ในตำแหน่งที่ใกล้ที่สุด ดังภาพที่ 62

ภาพที่ 62

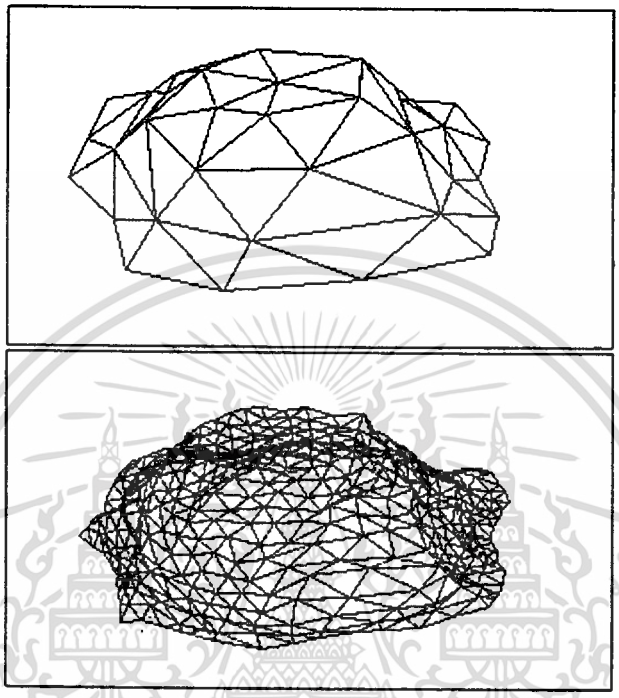
การสร้างพื้นผิวเชื่อมต่อระหว่างข้อมูลสองระดับ



เมื่อเชื่อมต่อไล่ระดับต่อเนื่องกันจากระดับล่างสุดจนถึงระดับบนสุด ก็จะได้พื้นผิวที่มีองค์ประกอบภาพเป็นรูปสามเหลี่ยมทั้งหมด แต่ถ้าข้อมูลตำแหน่งบนวัตถุที่สุ่มมามีจำนวนไม่มาก ภาพที่ได้ก็จะเป็นพียงภาพโครงร่างของวัตถุซึ่งไม่มีรายละเอียด ตัวอย่างเช่น การสร้างภาพภูมิประเทศ โดยการหาค่าตำแหน่งจากเส้นบอกระดับ (contour line) บนแผนที่ แล้วนำมาสร้างเป็นภาพโครงร่างโดยวิธีข้างต้น เมื่อใส่พื้นผิว Fractal ลงบนรูปสามเหลี่ยมซึ่งเป็นองค์ประกอบของโครงร่างที่ได้ จะได้ภาพดังตัวอย่างในภาพที่ 63

ภาพที่ 63

การสร้างภาพวัตถุจากข้อมูลบอกระดับ



ปัญหาของการสร้างภาพของการสร้างภาพจากข้อมูลของวัตถุ จะอยู่ที่ขั้นตอนการหาตำแหน่งจุดบนพื้นผิววัตถุ ซึ่งเป็นขั้นตอนที่เปลี่ยนไปตามสภาพการใช้งาน ต้องใช้ความละเอียดในการออกแบบมาก โดยเฉพาะอย่างยิ่ง ถ้าต้องการภาพที่มีรายละเอียดเหมือนวัตถุจริงซึ่งจะต้องใช้จำนวนจุดข้อมูลมาก จากตัวอย่างข้างบน จะเห็นว่าภาพที่ได้มีปัญหาเรื่องความต่อเนื่องของพื้นผิว ซึ่งเป็นผลสืบเนื่องมาการเตรียมข้อมูลไม่ดีพอ ทำให้รูปสามเหลี่ยมที่ได้มีขนาดต่างกัน อาจจะเปลี่ยนมาใช้วิธีการแบ่งพื้นที่โดยกำหนดขนาดของสามเหลี่ยมย่อย แทนการกำหนดจำนวนครั้งของการแบ่งพื้นที่ (ภาคผนวก จ)

ในกรณีที่องค์ประกอบภาพมีมากเกินไป จนทำให้เกิดปัญหาในการเก็บข้อมูล หรือใช้เวลาในการประมวลผลมากเกินไป อาจจะต้องปรับโครงสร้างโดยลดจำนวนองค์ประกอบย่อยของพื้นผิวลง โดยไม่ให้กระทบต่อรูปทรงของวัตถุ หรือให้มีผลน้อยที่สุด ซึ่งจะต้องเลือกเอาระหว่างความถูกต้องของรูปทรงกับความเร็วในการแสดงภาพ

บทที่ 8
บทสรุป

ในการออกแบบระบบสร้างภาพ จะต้องเริ่มจากการกำหนดลักษณะของภาพที่ต้อง การในที่นี้ กำหนดให้เป็นภาพวัตถุในระบบสามมิติ ที่มีองค์ประกอบของภาพเป็นเส้นตรงทั้งหมด โดยแทนพื้นผิวของวัตถุภาพด้วยรูปเหลี่ยมพื้นราบ ในกรณีที่พื้นผิววัตถุไม่ใช่พื้นผิวราบธรรมดา เช่น พื้นผิวโค้งหรือพื้นผิวไม่เรียบ ก็จะแบ่งพื้นผิวเหล่านั้นออกเป็นรูปสามเหลี่ยมเล็ก ๆ ถ้าใช้จำนวนของพื้นที่ย่อยมากพอ จะได้ภาพพื้นผิวที่มีความต่อเนื่องใกล้เคียงกับพื้นผิวจริง

เมื่อได้ลักษณะของภาพแล้ว อันดับต่อไปจะต้องกำหนดรูปแบบของข้อมูลภาพ ในที่นี้จะกำหนดให้หน่วยพื้นฐานของข้อมูลประกอบด้วย รหัสคำสั่งและข้อมูลตำแหน่ง ส่วนของรหัสคำสั่งจะเป็นข้อมูลขนาดหนึ่งไบท์ซึ่งแทนคำสั่งได้ถึง 256 คำสั่ง เปิดโอกาสให้เพิ่มคำสั่งการทำงานได้ โดยไม่กระทบต่อโครงสร้างส่วนใหญ่ ส่วนข้อมูลตำแหน่งจะใช้ตัวแปรแบบ float เก็บข้อมูลที่อยู่ในพิกัดกลางสำหรับค่าบนแกนทั้งสามในรูปของเว็คเตอร์

ปัญหาที่พบในกรณีที่มีการสร้างภาพโดยใช้องค์ประกอบย่อยจำนวนมาก ก็คือ การใช้หน่วยความจำ เพราะเครื่องคอมพิวเตอร์ในระบบของไอบีเอ็ม จะมีปัญหาในเรื่องของหน่วยความจำปกติมีจำนวนจำกัด ถ้าข้อมูลมีจำนวนมากก็สามารถเก็บข้อมูลในแฟ้มข้อมูลได้ ทั้งนี้ผู้ออกแบบระบบจะต้องพิจารณาจากลักษณะการใช้งานว่าจะต้องใช้เนื้อที่เก็บข้อมูลมากน้อยแค่ไหน สำหรับระบบที่ออกแบบไว้ในที่นี้ เป็นระบบที่เก็บข้อมูลได้ขนาดปานกลาง โดยใช้หน่วยความจำ XMS ซึ่งมีขนาดใหญ่กว่าหน่วยความจำปกติมาช่วยในการเก็บข้อมูลบางส่วน แต่ก็มีบางส่วนที่ใช้หน่วยความจำปกติอยู่

ในการสร้างระบบแสดงภาพ ส่วนสำคัญจะอยู่ที่ขั้นตอนการประมวลผลภาพ และปัญหาในการออกแบบระบบแสดงภาพ ไม่ได้อยู่ที่การค้นหาทฤษฎีใหม่ในการแสดงภาพ ซึ่งมีผู้คิดค้นวิธีการต่าง ๆ ไว้มากมาย แต่ไม่มีวิธีไหนที่ครอบคลุมการทำงานได้ทุกอย่าง ตัวอย่างเช่น วิธี Ray Tracing จะสร้างภาพได้ดี แต่ต้องใช้เวลาในการประมวลผลมากจนไม่สามารถนำใช้กับงานที่ต้องการความเร็วได้ ทั้งนี้จะขึ้นอยู่กับการเลือกเอาวิธีการที่เหมาะสมมาใช้กับงานที่ต้องการ ในที่นี้ได้เลือกเอาวิธีง่าย ๆ ไม่ซับซ้อน และใช้เวลาในการประมวลผลไม่มาก เหมาะกับการแสดงภาพที่ไม่ต้องการความเหมือนจริงมากนัก ตัวอย่างเช่น การแสดงภาพตัว

อักษรในแบบสามมิติ หรือ การแสดงแบบจำลองวัตถุทางวิศวกรรม ซึ่งสามารถแทนวัตถุภาพด้วยข้อมูลตำแหน่งจุดในระบบสามมิติ

ระบบโปรแกรมโดยทั่วไป จะแบ่งส่วนการทำงานสำหรับการใช้งานได้เป็นสองระดับ ได้แก่ ส่วนการทำงานสำหรับผู้ใช้งานทั่วไป และส่วนการทำงานสำหรับผู้พัฒนาระบบ ปัญหาในการเขียนส่วนการทำงานแต่ละอันก็คือ ต้องครอบคลุมการใช้งานให้ได้มากที่สุด โดยไม่ให้สูญเสียความกระชับและความเร็วในการทำงาน ในขณะที่ความต้องการของผู้ใช้งานทั่วไปต้องการส่วนการทำงานที่ใช้งานได้ง่าย มีเงื่อนไขหรือข้อกำหนดที่ผู้ใช้ต้องตัดสินใจน้อยที่สุด ในส่วนนี้สามารถแก้ไขได้โดยเขียนส่วนการทำงานซึ่งเรียกใช้ส่วนการทำงานเดิม โดยกำหนดเงื่อนไขเฉพาะกิจ ตัวอย่างเช่น ส่วนการทำงาน Set_View ( ) ต้องการค่ากำหนดขอบเขตพื้นที่แสดงภาพ ซึ่งเป็นปัญหาสำหรับผู้ใช้บางคน จึงต้องสร้างส่วนการทำงาน Init_View ( ) ซึ่งผู้ใช้ไม่ต้องใส่ค่าใด ๆ แต่ส่วนการทำงานนี้จะไปเรียกใช้ส่วนการทำงาน Set_View ( ) อีกทีหนึ่ง โดยกำหนดพื้นที่แสดงภาพให้เต็มจอภาพ

ข้อดีอย่างหนึ่งของการสร้างระบบขึ้นมาเองทั้งหมด ก็คือ ในการพัฒนาระบบจะมีผลกระทบต่อโครงสร้างส่วนไหนของระบบบ้าง สำหรับการพัฒนาระบบจะแยกออกเป็นสองส่วน ได้แก่ ส่วนของการใช้งาน และ ส่วนของระบบแสดงภาพ โดยทั่วไป การพัฒนาส่วนใช้งานจะไม่มีผลกระทบต่อโครงสร้างของระบบภาพ ตัวอย่างเช่น การปรับปรุงโครงสร้างขององค์ประกอบย่อยของพื้นผิวไม่เรียบ ให้มีความต่อเนื่องของพื้นผิวในกรณีที่แสดงภาพแบบโครงร่าง หรือลดจำนวนองค์ประกอบลงเพื่อเพิ่มความเร็วในการแสดงภาพแบบพื้นผิวทึบ โดยรักษารูปทรงเดิมของพื้นผิวนั้นไว้ให้มากที่สุด หรือในการสร้างภาพตัวอักษรระบบสามมิติ อาจจะพัฒนาระบบเตรียมข้อมูล เพื่อสร้างองค์ประกอบของตัวอักษรแต่ละตัวเก็บไว้ในแฟ้มภาพซึ่งจะนำมาใช้ในภายหลังได้ แต่ในส่วนของการพัฒนาระบบแสดงภาพ อาจจะต้องมีการปรับโครงสร้างบางส่วนของระบบภาพ เช่น ปรับปรุงส่วนแสดงภาพให้สามารถนำข้อมูลตัวอักษรในแฟ้มภาพจากระบบเตรียมข้อมูลภายนอกมาใช้ได้ หรือปรับโครงสร้างข้อมูลให้เก็บองค์ประกอบภาพได้มากขึ้น หรือเพิ่มคุณสมบัติวัตถุภาพ เพื่อปรับปรุงกระบวนการประมวลผลภาพให้สามารถแสดงภาพที่ซับซ้อนมากขึ้น เช่น การหาเงาของวัตถุ หรือแสงสะท้อนของพื้นผิวอื่นได้

เอกสารอ้างอิง

[1] Harrington, Steven, **Computer Graphics, A Programming Approach,** McGraw-Hill, Singapore, pp. 397-441, 1987.

[2] กิตติ ไพฑูรย์วัฒนกิจ และ สุวัฒน์ ศรีธนะรัตน์,""**การสร้างภาพในระบบสามมิติด้วยวิธี Interactive**",การประชุมวิชาการทางวิศวกรรมประจำปี 2533, วิศวกรรมสถานแห่งประเทศไทย, หน้า 1-8.

[3] Rimmer Steve, **Supercharged Bitmapped Graphics,** Windcrest/McGraw-Hill, USA, pp. 337-505, 1992.

[4] Williams Dave, **Programmer's Technical Reference**, John Wiley & Sons, Singapore, pp. 105-107, 1990.

[5] Foley James D., **Computer Graphics, Principles and Practice,** Addison-Wesley, USA, pp. 67-91, 1992.

[6] Hill Francis S. Jr, **Computer Graphics,** Macmillan, Singapore, pp. 327-380, 1990.

[7] Sproull Robert F. and Sutherland W.R. and Ullner Michael K, **Device Independent Graphics**, pp. 345-401, 1989.

[8] Stevens Roger T.and Watkins Steven D., **Advance Graphics Programming in C and C++**, Prentice Hall, England, pp. 17-39, 1991.

[9] Foley James D., **Computer Graphics, Principles and Practice,** Addison-Wesley, USA, pp. 649-717, 1992.

[10] Stevens Roger T., **Fractal Programming and Ray Tracing with C++**, Prentice Hall, England, pp. 215-224, 1991.

[11] Rankin John R., **Computer Graphics Software Construction,** Prentice Hall, Australia, pp. 106-109, pp. 259-262, 1989

[12] กิตติ ไพฑูรย์วัฒนกิจ และ สุวัฒน์ ศรีธนะรัตน์, "**การแสดงภาพภูมิประเทศด้วย Fractal Surfaces**", การประชุมวิชาการทางวิศวกรรมไฟฟ้าครั้งที่ 13, 2533, หน้า 638-645.

[13] กิตติ ไพฑูรย์วัฒนกิจ และ สุวัฒน์ ศรีธนะรัตน์, "**การทำ Fractal Surfaces ด้วยวิธีเก็บโครงสร้างของ Fractal Points**", การประชุมวิชาการทางวิศวกรรมประจำปี 2534, วิศวกรรมสถานแห่งประเทศไทย, หน้า 71-78.

ภาคผนวก ก

ส่วนควบคุมการแสดงภาพ (Graphics Display Driver)

ส่วนสำคัญที่ทำให้แผงวงจรควบคุมการแสดงภาพแบบ VGA แต่ละชนิดมีความแตกต่างกัน ได้แก่ การอ้างอิงตำแหน่งของหน่วยความจำภาพ (Screen Buffer) ซึ่งถูก แบ่งออกเป็นหน้า (page) หน้าละ 64K การหาตำแหน่งของจุดภาพจะต้องหาว่าอยู่ในหน้าไหน และ หาตำแหน่งออฟเซ็ทในหน้านั้น ปัญหาก็คือ แผงวงจรของแต่ละบริษัทมีวิธีในการเก็บข้อมูลภาพต่างกัน วิธีการหาหน้าของหน่วยความจำภาพและตำแหน่งของจุดภาพก็แตกต่างกัน ตัวอย่างเช่น แผงวงจรของ TRIDENT จะใช้วิธีเก็บข้อมูลภาพเก็บเรียงติดต่อกันทีละเส้นจนเต็มเนื้อที่ 64K เส้นภาพเส้นสุดท้ายของหน้าก็จะกินเนื้อที่ข้ามไปอีกหน้าหนึ่ง จะต้องแบ่งเส้นภาพนั้นออกเป็นสองส่วน ในภาค ผนวกนี้เป็นตัวอย่างของส่วนควบคุมที่ใช้วิธีควบคุมโดยตรง โดยใช้รีจิสเตอร์ควบคุมในการเลือกหน้าของหน่วยความจำภาพ

```
;ส่วนควบคุมการแสดงภาพขนาด 640x480 จุดสำหรับ TRIDENT
_AOFF       EQU   6                 ;ตำแหน่งออฟเซ็ทของพารามิเตอร์
VGA_MODE    EQU   05DH              ;หมายเลขแบบการแสดงภาพ
VGA_WIDE    EQU   640               ;จำนวนจุดในแนวนอน
VGA_DEEP    EQU   480               ;จำนวนจุดในแนวตั้ง
SCREENSEG   EQU   0A000H            ;จุดเริ่มต้นของหน่วยความจำภาพ

CODE        SEGMENT PARA PUBLIC 'CODE'
        ASSUME      CS:CODE
        DB      'SCDRV_01'              ;ข้อความตรวจสอบ
        ORG     0008H
HEADER      PROC  NEAR
        DW      VGA_ON                  ;ตำแหน่งออฟเซ็ทของส่วนการทำงาน
        DW      VGA_OFF
```

```
        DW      CLEAR_SCREEN
        DW      PUT_POINT
        DW      PUT_LINE
        DW      VGA_WIDTH
        DW      VGA_DEPTH
HEADER  ENDP

VGA_ON  PROC                    ;เข้าสู่ระบบการแสดงภาพ
        MOV     AX,VGA_MODE
        INT     10H
        RETF
VGA_ON  ENDP

VGA_OFF PROC                    ;ออกจากการแสดงภาพ
        MOV     AX,0003H
        INT     10H
        RETF
VGA_OFF ENDP


CLEAR_SCREEN    PROC            ;ลบจอภาพ
        PUSH    BP
        MOV     BP,SP
        PUSH    DI
        MOV     AX,SCREENSEG            ;ตำแหน่งเริ่มต้น
        MOV     ES,AX
        MOV     AX,[BP + _AOFF + 0]     ;สีพื้น
        MOV     AH,AL
        MOV     BX,0
CLR1:   MOV     CS:[BANK], BX           ;เลือกหน้า
```

```
        CALL  SET_BANK
        MOV   CX,08000H                 ;ใส่ข้อมูลสีทั้งหน้า
        MOV   DI,0
        REPNE STOSW
        INC   BX                        ;หน้าต่อไป
        CMP   BX,5
        JNE   CLR1
        POP   DI
        POP   BP
        RETF
CLEAR_SCREEN        ENDP

PUT_POINT   PROC                        ;ใส่ข้อมูลจุดภาพ
        PUSH  BP
        MOV   BP,SP
        PUSH  ES
        MOV   AX,SCREENSEG
        MOV   ES,AX
        MOV   CX,[BP + _AOFF + 0]       ;ตำแหน่งแนวนอน
        MOV   BX,[BP + _AOFF + 2]       ;ตำแหน่งแนวตั้ง
        CALL  GET_ADDR                  ;คำนวณตำแหน่ง
        MOV   BX,AX
        CALL  SET_BANK                  ;เลือกหน้า
        MOV   AX,[BP + _AOFF + 4]       ;สี
        MOV   ES:[BX],AL                ;ใส่ค่าสี
        POP   ES
        POP   BP
        RETF
PUT_POINT   ENDP
```

```
PUT_LINE    PROC                          ;ใส่ข้อมูลเส้นภาพ
        PUSH  BP
        MOV   BP,SP
        PUSH  DS
        PUSH  ES
        PUSH  SI
        PUSH  DI
        CMP   WORD PTR [BP + _AOFF + 6],VGA_DEEP
        JGE   PUTL4
        MOV   AX,SCREENSEG
        MOV   ES,AX
        MOV   SI,[BP + _AOFF + 0]     ;ตำแหน่งของข้อมูลเส้นภาพ
        MOV   DS,[BP + _AOFF + 2]
        MOV   CX,[BP + _AOFF + 8]     ;จำนวนไบท์
        MOV   AX,VGA_WIDE             ;ตรวจสอบเนื้อที่เหลือบนเส้น
        SUB   AX,[BP + _AOFF + 4]
        CMP   AX,CX
        JGE   PUTL1
        MOV   CX,AX                   ;ปรับขนาดข้อมูลให้เท่ากับเนื้อที่
PUTL1:  PUSH  CX
        MOV   CX,[BP + _AOFF + 4]     ;ตำแหน่งบนเส้นภาพ
        MOV   BX,[BP + _AOFF + 6]     ;ตำแหน่งเส้นภาพ
        CALL  GET_ADDR                ;คำน่วณตำแหน่ง
        MOV   DI,AX
        CALL  SET_BANK                ;เลือกหน้า
        POP   CX
        ADD   AX,CX                   ;ตรวจสอบว่าข้ามหน้าหรือไม่
        JC    PUTL2
        JMP   PUTL3
PUTL2:  SUB   CX,AX                   ;แบ่งข้อมูลเป็นสองส่วน
```

```
        CLD
        REPNE MOVSB                     ;ใส่ข้อมูลส่วนแรก
        INC   CS:[BANK]                 ;เปลี่ยนหน้า
        CALL  SET_BANK
        MOV   CX,AX
        MOV   DI,0
PUTL3:  CLD
        REPNE MOVSB                     ;ใส่ข้อมูลส่วนที่เหลือทั้งหมด
PUTL4:  POP   DI
        POP   SI
        POP   ES
        POP   DS
        POP   BP
        RETF
PUT_LINE    ENDP

SCREEN_WIDTH    PROC                    ;จำนวนจุดแนวนอน
        MOV   AX, SCREEN_WIDE
        RETF
SCREEN_WIDTH            ENDP

SCREEN_WIDTH    PROC                    ;จำนวนจุดแนวตั้ง
        MOV   AX, SCREEN_DEEP
        RETF
SCREEN_WIDTH            ENDP

SET_BANK    PROC  NEAR                  ;เลือกหน้าหน่วยความจำภาพ
        PUSH  AX
        PUSH  DX
        MOV   AX,CS:[BANK]              ;หมายเลขหน้าใหม่
```

```
        CMP   AX,CS:[OLDB]          ;ตรวจสอบว่าอยู่หน้าเดิมหรือไม่
        JE    BNK1
        CLI
        MOV   CS:[OLDB],AX          ;เปลี่ยนหน้า
        MOV   AH,AL
        XOR   AH,2
        MOV   DX,3C4H
        MOV   AL,0EH
        OUT   DX,AX
        STI
BNK1:   POP   DX
        POP   AX
        RET
SET_BANK    ENDP

GET_ADDR    PROC  NEAR              ; คำนวณตำแหน่งจุด
        PUSH  DX
        MOV   AX,VGA_WIDE
        MUL   BX                    ;ความกว้างxตำแหน่งแถว
        ADD   AX,CX                 ;บวกตำแหน่งคอลัมน์
        JNC   ADR1                  ;ได้ค่าออฟเซ็ทใน AX
        INC   DX                    ;และเลขหน้าใน DX
ADR1:   MOV   CS:[BANK],DX
        POP   DX
        RET
GET_ADDR    ENDP
BANK  DW    0000H                   ;เลขหน้าใหม่
OLDB  DW    00FFH                   ;เลขหน้าเดิม
CODE  ENDS
```

END

ส่วนการทำงานสำหรับเรียกส่วนควบคุมนี้มาใช้งานจะมีลักษณะดังนี้

```
#include <stdio.h>
#include <alloc.h>
#include <mem.h>
#include <dos.h>
int  far (*On_Graph)     (void);          // ตัวแปรชี้ตำแหน่ง
void far (*Off_Graph)    (void);          // ส่วนการทำงาน (Function Pointers)
void far (*Clear_Screen)(int color);
void far (*Put_Point)     (int x, int y, int color, int mode);
void far (*Put_Line)      (char *p, int x, int y, int n);
int  far (*Screen_Width)(void);
int  far (*Screen_Depth)(void);

int Load_Driver ( )              //ดึงส่วนควบคุมจากแฟ้มมาไว้ในหน่วยความจำ
{
        FILE    *fo;
        unsigned int  *screen;
        unsigned char b[8];
        unsigned int  n, sgm;
        if  ((fp=fopen("SCREEN.DRV") == NULL)       return (1);
        fseek (fp,0L,SEEK_END);
        n = ftell (fp);                          //หาขนาด
        rewind (fp);
        fread (b,1,8,fp);                        //อ่านข้อมูล 8 ไบท์แรก
        if (!memcmp(b,"SCDRV_01",8)) {           //ตรวจสอบข้อความ
        screen = (unsigned *)malloc(n+8);        //จองเนื้อที่
        sgm = (long)screen>>16;
```

```
        screen = (unsigned *)MK_FP (sgm, 8);
        if (fread(screen,n-8,1,fp)!=1)              //อ่านข้อมูลส่วนควบคุมทั้งหมด
                free (screen); fclose(fp); return (1);
        }
/* ใส่ค่าตำแหน่งสำหรับส่วนการทำงานแต่ละอันในส่วนควบคุม    */
        On_Graph        = (int far(*)())     MK_FP(sgm, screen[0]);
        Off_Graph       = (void far(*)())    MK_FP(sgm, screen[1]);
        Clear_Screen    = (void far(*)(int)) MK_FP(sgm, screen[2]);
        Put_Point       = (void far(*)(int,int,int,int)) MK_FP(sgm, screen[3]);
        Put_Line        = (void far(*)(char*,int,int,int)) MK_FP(sgm, screen[5]);
        Screen_Width    = (int far(*)()) MK_FP(sgm, screen[10]);
        Screen_Depth    = (int far(*)()) MK_FP(sgm, screen[11]);
        On_Graph ( );
        fclose (fp);
        return (0);
}

void Unload_Driver ()              //ยกเลิกการใช้ส่วนควบคุม
{
        if (On_Graph!=NULL)  free ((void *)On_Graph);
        On_Graph = NULL;
}
```

ข้อเสียของการเขียนโปรแกรมโดยใช้ส่วนควบคุมแยกส่วนที่มีลักษณะเฉพาะ ก็คือ ความสะดวกในการนำไปปรับเปลี่ยนเพื่อให้ใช้กับระบบที่ต่างกันได้ ซึ่งจะเป็นหน้าที่ของผู้ที่จะเขียนส่วนใช้งาน ที่จะอำนวยความสะดวกในการเลือกส่วนควบคุมให้เหมาะสมกับระบบที่จะใช้งาน

ภาคผนวก ข

ส่วนควบคุมการใช้หน่วยความจำพิเศษ

หน่วยความจำ XMS เป็นหน่วยความจำที่มีลักษณะเฉพาะสำหรับคอมพิวเตอร์ในระบบของไอบีเอ็ม ขั้นตอนการใช้งานจะผิดไปจากการใช้หน่วยความจำปกติ ในภาคผนวกนี้จะแสดงตัวอย่างการเขียนส่วนการทำงานสำหรับใช้หน่วยความจำ XMS

```
#include <stdio.h>
#include <dos.h>
#include <alloc.h>
#include  "h\xtramem.h"
struct  Xmove {                          // โครงสร้างควบคุมการส่งข้อมูล
        unsigned  long   Length;         // จำนวนไบท์
        unsigned  int    Shandle;        // หน่วยความจำต้นทาง
        unsigned  long   Saddress;       // ตำแหน่ง
        unsigned  int    Dhandle;        // หน่วยความจำปลายทาง
        unsigned  long   Daddress;       // ตำแหน่ง
} Xm;
int  far  (*Xms_Call) (void);            // ตัวแปรชี้ตำแหน่งของส่วนควบคุม

int  Xms_Alloc (unsigned n, unsigned bytes)
{
        union REGS r;
        struct SREGS s;
        r.x.ax = 0x4300;                 // ตรวจสอบส่วนควบคุม
        int86(0x2f,&r,&r);
        if (r.h.al != 0x80) return(0);
```

```
        r.x.ax = 0x4310;
        int86x(0x2f,&r,&r,&s);
        Xms_Call = MK_FP(s.es, r.x.bx);          // ตำแหน่งของส่วนควบคุม
        kbytes = 1 + (long)n * bytes / 1024;     // คำนวณเนื้อที่เป็น Kbyte
        _AH = 9;                                 // จองเนื้อที่
        _DX = kbytes;
        if (Xms_Call())  h = _DX;                // หมายเลขของเนื้อที่ที่จองได้
        return (h);
}

int Xms_Move ()
{
        _SI = FP_OFF(&Xm);                       // ตำแหน่งของค่าควบคุม
        _DS = FP_SEG(&Xm);
        _AH=0x0B;                                // ส่งข้อมูล
       return  (Xms_Call( ));
}

int   Xms_Read (Xhandle h, unsigned long p, unsigned long n , char *m)
{
        Xm.Length=n;
        Xm.Shandle=h;                            // หน่วยความจำ XMS
        Xm.Saddress=p;
        Xm.Dhandle=0;                            // หน่วยความจำปกติ
        Xm.Daddress=(long)m;
        return  (Xms_Move( ));
}

int  Xms_Write (Xhandle h, unsigned long p, unsigned long n , char *m)
{
```

```
        Xm.Length=n;
        Xm.Shandle=0;                          // หน่วยความจำปกติ
        Xm.Saddress=(long)m;
        Xm.Dhandle=h;                          // หน่วยความจำ XMS
        Xm.Daddress=p;
        return  (Xms_Move( ));
}

int Xms_Free (Xhandle handle)
{
        if (handle != -1) {
            _AH = 0x0A;                        // ยกเลิกการใช้เนื้อที่
            _DX = h;
            Xms_Call ();
            return(1);
        }
        return(0);
}
```

ตัวอย่างต่อไปนี้เป็นการใช้หน่วยความจำ XMS ในการเก็บข้อมูลภาพที่อ่านมาจากแฟ้มภาพแบบ BMP แล้วนำภาพที่เก็บไว้มาแสดงบนจอภาพซ้ำกันหลาย ๆ ครั้ง

```
#include <stdlib.h>
#include <stdio.h>
#include <alloc.h>
#include "xtramem.c"
#include "drivers.c"
typedef struct {                           // header ของแฟ้มภาพแบบ BMP
        char id[2];
        long filesize;
```

```
        int  reserved[2];
        long headersize;
        long infosize;
        long width;
        long depth;
        int  plane;
        int  bits;
} BMPHEAD;

int  BmpWidth, BmpDepth, BmpBytes, BmpBits;
int  Buff=0;
int  Load_Bmp (char *name);
int  Get_Image (FILE *fp);
void Ega_2_Vga (char *p1, char *p2, int width);

Load_Bmp (char *name)                    //เปิดแฟ้มข้อมูลภาพแบบ BMP   {
        FILE *fp;
        if ((fp=fopen(name,"rb"))!=NULL) {
                if (!Get_Header(fp)) {
                        if (Get_Image (fp)) {
                                fclose (fp);  return (1);
                        }
                }
        }
        fclose (fp);  return (0);
}

Get_Header(FILE *fp)                     // ตรวจสอบแฟ้มภาพ
{
        BMPHEAD bmp;
```

```
        char  palette [768];
        int   i,n;
        fread ((char*)&bmp,1,sizeof(BMPHEAD),fp);
        if (!memcmp(bmp.id,"BM",2)) {            // ตรวจสอบ header
                if ((bmp.bits<4)||(bmp.bits>8))    return (1);
                BmpBits  = bmp.bits;
                BmpWidth = (int)bmp.width;
                BmpDepth = (int)bmp.depth;
                if (BmpBits==8)  BmpBytes=BmpWidth;
                        else     BmpBytes=(BmpWidth+7)/8 << 2;
                if (BmpBytes & 0x0003) {
                        BmpBytes |= 0x0003;
                        ++BmpBytes;
                }
                fseek (fp, 54L, SEEK_SET);
                n=1<<BmpBits;
                for (i=0; i<n; i++) {                // ตารางสี
                        palette[(i*3)+2] = fgetc(fp)>>2;
                        palette[(i*3)+1] = fgetc(fp)>>2;
                        palette[(i*3)]   = fgetc(fp)>>2;    fgetc(fp);
                }
                Set_Palette (palette,0,256);
                fseek (fp, bmp.headersize, SEEK_SET);
                return (0);
        }
        return (1);
}

Get_Image (FILE *fp)
{
```

```
        char *p1,*p2;
        unsigned size;
        int  i,n;
        Buff=Xms_Alloc(BmpDepth, BmpWidth);          // เนื้อที่สำหรับภาพทั้งหมด
        if (Buff==0) return(1);
        p1 =(char*)malloc(BmpBytes);                 // เนื้อที่สำหรับภาพ 1 เส้น
        p2 =(char*)malloc(BmpWidth);
        for (i=0; i<BmpDepth; i++) {                 // อ่านข้อมูลภาพ
                if (fread(p1, BmpBytes, 1, fp)) {
                        if (BmpBits==8)
                                Xms_Write (Buff,BmpDepth-1-i,BmpWidth, p1);
                        else {
                                Ega_2_Vga (p1, p2,BmpWidth);
                                Xms_Write (Buff,BmpDepth-1-i,BmpWidth, p2);
                        }
                }
        }
        free(p1); free(p2);  return(0);
}

void  Ega_2_Vga (char *p1, char *p2, int width)      //แปลงข้อมูล EGA
{
        int i,j;
        for (i=0,j=0;i<width;j++) {
                p2[i++]=(p1[j] >> 4) & 0x0f;
                p2[i++]=p1[j] & 0x0f;
        }
}

void  Display_Buffer (int x, int y)                  // แสดงภาพ
```

```
{
        int i;
        char *p;
        p=(char*)malloc(BmpWidth);
        for (i=0;i<BmpDepth;i++) {
                Xms_Read (Buff,i,BmpWidth, p);
                Put_Line (p,x,y+i,BmpWidth);
        }
        free(p);
}

void main ()
{
        int    x,y,xm,ym,i;
        if (On_Graph(1,0x5D)) return;                    // เข้าระบบภาพ
        if (!Load_Bmp ("\\windows\\leaves.bmp")) {       // อ่านข้อมูลภาพ
                Off_Graph ();  return;
        }
        ym=Screen_Depth() - BmpDepth;
        xm=Screen_Width() - BmpWidth;
        randomize ();
        for (i=0; i<500; i++) {                          // แสดงภาพ 500 ครั้ง
                x=random(xm);                            // สุ่มตำแหน่ง
                y=random(ym);
                Display_Buffer (x,y);                    // แสดงภาพ
        }
        Xms_Free (Buff);                                 // ยกเลิกเนื้อที่
        getch ();
        Off_Graph();                                     //ออกจากระบบภาพ
}
```

ภาคผนวก ค
ส่วนการทำงานพื้นฐานของระบบภาพ

ในภาคผนวกนี้เป็นรายชื่อของส่วนการทำงานที่จำเป็นในระบบภาพ ซึ่งผู้ใช้สามารถเรียกใช้งานได้เลย ตั้งแต่ส่วนการทำงานที่ติดต่อกับจอภาพจนถึงส่วนการทำงานในระบบสามมิติ

ส่วนควบคุมการทำงานของจอภาพ

On_Graph	(type, mode)	เข้าสู่ระบบภาพ mode คือ แบบแสดงภาพ type มีค่าเท่ากับ 1,2,3 สำหรับการแสดงภาพขนาด 640x480, 800x600 และ 1024x768 ตามลำดับ
Off_Graph	( )	ออกจากระบบภาพ
Clear_Screen	(color)	ลบจอภาพด้วยสี color (0 - 255)
Put_Point	(x,y,color)	แสดงจุดสี color ที่ตำแหน่ง x,y บนจอภาพ
Get_Point	(x,y)	อ่านข้อมูลจุดภาพที่ตำแหน่ง x,y ของจอภาพ ค่าที่ส่งกลับมาก็คือ ค่าสีที่ตำแหน่งจุดนั้น
Put_line	(p,x,y,n)	นำข้อมูลจากหน่วยความจำที่ตำแหน่ง p จำนวน n ไบท์ไปใส่บนเส้นภาพที่ y เริ่มตั้งแต่จุดที่ x
Get_Line	(p,x,y,n)	อ่านข้อมูล n จุดตั้งแต่จุดที่ x บนเส้นภาพที่ y มาไว้ที่หน่วยความจำภาพที่ตำแหน่ง p
Fill_Line	(x,y,n,color)	ลากเส้นแนวนอนสี color ยาว n จุด เริ่มจากจุด (x,y)
Set_Palette	(p,s,n)	นำข้อมูลจากตำแหน่ง p ไปใส่ในตารางสี n สีเริ่มต้นจากสีที่ s
Get_Palette	(p,s,n)	อ่านข้อมูล n สีจากตารางสีตั้งแต่สีที่ s มาไว้ที่หน่วยความจำตำแหน่ง p
Screen_Width	( )	จำนวนจุดภาพในแนวนอน
Screen_Depth	( )	จำนวนจุดภาพในแนวตั้ง

ส่วนแสดงภาพในพิกัดจอภาพ

Draw_Point	(x,y,color)	แสดงจุดสี color ที่ตำแหน่ง (x,y) โดยมีการตรวจสอบว่าอยู่ภายในขอบเขตที่กำหนดไว้
Draw_Line	(x1,y1,x2,y2,c)	ลากเส้นตรงสี c จากตำแหน่ง (x1,y1) ไปยังตำแหน่ง (x2,y2)
Set_View	(L,T,R,B)	กำหนดพื้นที่การแสดงภาพโดยที่ (L,T) เป็นตำแหน่งมุมบนซ้ายและ (R,B) เป็นตำแหน่งมุมล่างขวา
Clear_View	(color)	ลบพื้นที่แสดงภาพด้วยสี color
Init_Screen	( )	กำหนดให้แสดงภาพเต็มจอภาพ

ข้อมูลตำแหน่งเป็นตำแหน่งจุดบนจอภาพ ค่าสีจะมีค่าตั้งแต่ 0 ถึง 255

ส่วนการแสดงภาพในระบบพิกัดกลาง 2 มิติ

Draw_Point_2D	(x,y,color)	แสดงจุดสี color ที่ตำแหน่ง (x,y)
Draw_Line_2D	(x1,y1,x2,y2,c)	ลากเส้นตรงสี color จากตำแหน่ง (x1,y1) ไปยังตำแหน่ง (x2,y2)
Set_Window	(x1,y1,x2,y2,c)	กำหนดพื้นที่แสดงภาพจากตำแหน่ง (L,T) ถึงตำแหน่ง (R,B)
Init_Window	( )	กำหนดพื้นที่แสดงภาพเต็มจอภาพ

ตำแหน่งจุดที่ใช้ในส่วนนี้จะเป็นค่าในพิกัดกลาง โดยในส่วนของพื้นที่แสดงภาพจะมีค่าตั้งแต่ -1.0 ถึง 1.0 ยกเว้นคำสั่ง Init_Window ( ) ซึ่งจะเป็นการกำหนดพื้นที่บนจอภาพเหมือนคำสั่ง Init_View( ) แต่ในส่วนการทำงานนี้จะมีการปรับค่าสำหรับการเปลี่ยนข้อมูลจากพิกัดกลางไปเป็นพิกัดจอภาพด้วย

ส่วนการแสดงภาพในระบบพิกัดกลาง 3 มิติ

Draw_Point_3D	(pt,color)	แสดงจุดภาพที่ตำแหน่ง pt
Draw_Line_3D	(pt1,pt2,color)	ลากเส้นตรงจากจุด pt1 ไปยังจุด pt2
View_Position	(pos)	กำหนดตำแหน่งผู้มองภาพ
View_Direction	(dir)	กำหนดทิศทางการมองภาพโดยเวิ์คเตอร์ dir
View_Angle	(phi,theta)	กำหนดทิศทางการมองภาพโดยใช้มุมเป็น phi องศา จากแกน Y และหมุน theta องศารอบแกน Y
View_Up	(up)	กำหนดเวิ์คเตอร์ up เป็นแนวตั้งของฉากรับภาพ
View_Distance	(d)	กำหนดระยะห่างของฉากรับภาพสำหรับการฉายภาพแบบมีระยะลึก
Perspective_On	( )	ให้แสดงภาพแบบมีความลึก
Perspective_Off	( )	ให้แสดงภาพแบบขนาน
Init_3D	( )	จัดเตรียมการแสดงภาพ 3 มิติ

ส่วนการทำงานในส่วนนี้ จะใช้ค่าตำแหน่งเป็นเวิ์คเตอร์ในระบบสามมิติ

ส่วนการทำงานสำหรับการเปลี่ยนพิกัด

Translate_3D	(t, mat)	สร้างแมตริกซ์ย้าย mat ตำแหน่งด้วยเวิ์คเตอร์ t
Scale_3D	(s, mat)	สร้างแมตริกซ์เปลี่ยนขนาด mat ด้วยเวิ์คเตอร์ s
Rotate_3D	(axis,theta,mat)	สร้างแมตริกซ์หมุนรอบแกน mat สำหรับการหมุนรอบแกน axis ด้วยมุม theta
Set_Tmatx	(s, t, r, mat)	สร้างแมตริกซ์ mat สำหรับการเปลี่ยนพิกัดรวม
Inv_Tmatx	(s, t, r, mat)	สร้างแมตริกซ์ mat สำหรับการเปลี่ยนย้อนกลับ
Transform	(a, b, mat)	เปลี่ยนพิกัดจากเวิ์คเตอร์ a เป็นเวิ์คเตอร์ b โดยใช้แมตริกซ์ mat ซึ่งเป็นแมตริกซ์ขนาด 4x4

s, และ t คือค่าเปลี่ยนขนาด และ ค่าย้ายตำแหน่งในแกนทั้งสาม ส่วน r เป็นค่าองศาสำหรับการหมุนรอบแกนทั้งสาม

ส่วนการทำงานพื้นฐานที่กล่าวมาแล้ว เป็นส่วนการทำงานสำหรับการแสดงภาพโดยตรง นั่นคือ เมื่อใช้คำสั่งเรียกส่วนการทำงานเหล่านี้ ก็จะได้ภาพออกมาที่จอภาพเลย แต่ในส่วนต่อไปนี้ เป็นส่วนการทำงานสำหรับการสร้างภาพที่ถูกเก็บไว้เพื่อรอการประมวลผล แล้วนำมาแสดงเป็นภาพในภายหลัง เริ่มจากการเตรียมเนื้อที่สำหรับการเก็บข้อมูลภาพจากคำสั่งสร้างภาพ

ส่วนการทำงานสำหรับการเก็บข้อมูล

New_Object	( )	จองที่สำหรับวัตถุภาพชิ้นใหม่
Deletel_Object	( )	ยกเลิกวัตถุภาพปัจจุบัน
New_Picture	( )	ยกเลิกข้อมูลภาพทั้งหมด
Hide_Object	( )	กำหนดให้ไม่แสดงวัตถุภาพ
Show_Object	( )	กำหนดให้แสดงวัตถุภาพ
Rotate_Object	(dx,dy,dz)	หมุนวัตถุภาพด้วยมุม dx, dy, dz รอบแกน x, y และ z ตามลำดับ
Scale_Object	(sx,sy,sz)	เปลี่ยนขนาดวัตถุภาพด้วยค่า sx, sy, sz สำหรับแกน x, y และ z ตามลำดับ
Trans_Object	(tx,ty,tz)	กำหนดตำแหน่งวัตถุภาพด้วยค่า tx, ty, tz สำหรับแกน x, y และ z ตามลำดับ
Color_Object	(color)	กำหนดสี color ให้วัตถุภาพ
First_Object	( )	เลือกวัตถุภาพลำดับแรก
Select_Object	(n)	เลือกวัตถุภาพลำดับที่ n
Next_Object	( )	เลือกวัตถุภาพลำดับต่อไป
Prev_Object	( )	เลือกวัตถุภาพลำดับก่อนหน้า

ข้อมูลภาพจะถูกแยกออกเป็นส่วน ๆ ซึ่งในที่นี้จะเรียกว่า Object และสามารถแยกการกำหนดคุณสมบัติของแต่ละส่วนได้

ส่วนการทำงานสำหรับการสร้างภาพ

Move_Abs	(p)	กำหนดตำแหน่งจริง p
Move_Rel	(pr)	กำหนดตำแหน่งสัมพัทธ์ pr
Line_Abs	(p)	ลากเส้นไปที่ตำแหน่งจริง p
Line_Rel	(pr)	ลากเส้นไปที่ตำแหน่งสัมพัทธ์ pr
Polygon_Abs	(n, p)	สร้างรูป n เหลี่ยมจากข้อมูลตำแหน่งจริง p
Polygon_Rel	(n, pr)	สร้างรูป n เหลี่ยมจากข้อมูลตำแหน่งสัมพัทธ์ pr

ส่วนการทำงานทั้งหมดในส่วนนี้ เป็นส่วนสร้างองค์ประกอบภาพซึ่งจะถูกเก็บไว้ในเนื้อที่ ซึ่งถูกกำหนดโดยส่วนเก็บข้อมูลภาพ

ส่วนการทำงานควบคุมการแสดงภาพ

Clear_Display	( )	ให้ลบภาพเดิมก่อนแสดงภาพใหม่
Make_Picture	( )	ให้นำข้อมูลมาแสดงภาพ
Set_BackColor	(color)	กำหนดสี color ให้ฉากหลัง
Set_ForeColor	(color)	กำหนดสี color ให้วัตถุภาพ
Hide_BackFace	( )	ไม่ให้แสดงพื้นผิวด้านหลัง
Show_BackFace	( )	ให้แสดงพื้นผิวด้านหลัง
Wire_Display	( )	ให้แสดงภาพแบบโครงลวด
Solid_Display	( )	ให้แสดงภาพแบบพื้นผิวทึบ
Light_Direction	(direction)	กำหนดทิศทาง direction ของแสง

ส่วนการทำงานในส่วนนี้ใช้สำหรับการกำหนดรูปแบบการประมวลผลภาพ เพื่อให้ได้ภาพที่มีลักษณะต่าง ๆ

ภาคผนวก ง

ตัวอย่างของส่วนการใช้งาน

ในภาคผนวกนี้เป็นตัวอย่างการใช้ส่วนการทำงานพื้นฐาน โดยเลือกเอาส่วนการใช้งานที่พูดถึงในบทที่ 6 ในส่วนที่เกี่ยวกับการใช้งานโดยแฟ้ม ตั้งแต่การเลือกส่วนควบคุม การเลือกวัตถุภาพ ไปจนถึงการกำหนดรูปแบบการแสดงภาพ

```
#include "h\graph_1.h"
#include "h\graph_2.h"

Init_Graph ( )                                  // เตรียมระบบแสดงภาพ
{
        char cmd[30], *s1, *s2;
        int   type=0, mode=0;
        FILE *fp;
        if ((fp=open("CONFIG.G3D","rt"))==NULL) return(0);
        do {                                    // อ่านแฟ้มกำหนดระบบภาพ
                fscanf (fp,"%s", cmd);
        } while (strcmp(cmd,"CONFIG"));
        do {                                    // อ่านข้อมูลกำหนดระบบภาพ
                fscanf (fp,"%s", cmd);
                if (!strcmp(cmd,"TYPE"))        fscanf(fp,"%d",&type);
                else if (!strcmp(cmd,"MODE"))   fscanf(fp,"%d",&mode);
                if ((mode>0)&&(type>0) break;
        }  while (!feof(fp));
        fclose (fp);
        if  ((mode==0)||(type==0) return (1);
```

```
        if (On_Graph (type, mode)) return (1);   // เข้าระบบภาพ
        Init_Window ( );                         // เตรียมพื้นที่แสดงภาพ
        Init_3D ();                              // เตรียมการแสดงภาพสามมิติ
        return (0);
}

void Exit_Graph (int status)                     // ออกจากระบบภาพ
{
        switch (status) {
                case 0: Off_Graph ( );
                        printf ("Program normally terminated");  break;
                case 1:
                        printf ("Incorrect arguments");  break
                case 2:
                        printf ("Init Graph error");   break;
                case 3:  Off_Graph ( );
                        printf ("Image File error"); break;
                case 4: Off_Graph ( );
                        printf ("Display File error");
        }
}

void Load_Object (char *s)                       // อ่านข้อมูลจากแฟ้มข้อมูลภาพ
{
        FILE    *fo;
        char    cmd[16];
        Vector  *v,  *p, cg;
        int     i,j,k,nv,nf,nn,lv, color;
        float   size=1;
        if ((fo=fopen(s,"rt"))==NULL) return;
```

```
fscanf (fo, "%s", cmd);
if (strcmp(cmd,"OBJECT")) {              // ตรวจสอบแฟ้มข้อมูล
        fclose (fo);
        return;
}
New_Object ( );                          // เตรียมเนื้อที่สำหรับข้อมูลภาพ
do {
        fscanf (fo,"%s", cmd);
        if (!strcmp(cmd,"COLOR")) {      // ข้อมูลสี
                fscanf(fo,"%d",&color);
                Color_Object (color);
        }
        if (!strcmp(cmd,"NODE")) {       // ข้อมูลตำแหน่ง
                fscanf(fo,"%d %f %f %f %f",
                        &nv,&size,&cg[0],&cg[1],&cg[2]);
                if (size!=0) size = 1.0 / size;
                v=(Vector *) malloc(nv*sizeof(Vector));
                for (i=0; i<nv; i++) {
                        fscanf (fo, "%f %f %f",&v[i][0], &v[i][1], &v[i][2]);
                        Sub_Vector (v[i], cg, v[i]);
                        Scale_Vector (size, v[i], v[i]);
                }
        }
        if (!strcmp(cmd,"POLYGON")) {    //ข้อมูลรูปเหลี่ยม
                fscanf(fo,"%d",&nf);
                for (i=0; i<nf; i++) {
                        fscanf(fo, "%d", &nn);
                        p =(Vector*) malloc (nn*sizeof(Vector));
                        for (j=0; j<nn; j++) {
                                fscanf(fo, "%d", &k);
```

```
                                Copy_Vector(v[k], p[j]);
                            }
                            Polygon_Abs (nn, p[0]);
                            free(p);
                        }
                    }
        } while (strcmp(cmd,"END"));
        free (v);
        fclose (fo);
}

void Load_Image (char *s)                    // อ่านข้อมูลกำหนดองค์ประกอบภาพ
{
        FILE    *fs;
        char    cmd[16];
        char    name[12];
        float   x,y,z;
        if ((fs=fopen(s,"rt"))==NULL) return;
        fscanf (fs, "%s", cmd);
        if (strcmp(cmd,"IMAGE")) return;
        do {
                fscanf (fs,"%s", cmd);
                if (!strcmp(cmd,"LOAD")) {
                        fscanf (fs, "%s", name);          // ชื่อแฟ้มข้อมูลภาพ
                        Load_Object (name);
                }
                if (!strcmp(cmd,"POSITION")) {            // ข้อมูลกำหนดตำแหน่ง
                        fscanf (fs, "%f %f %f",&x, &y, &z);
                        Translate_Object(x, y, z);
                }
```

```
            if (!strcmp(cmd,"ROTATION")) {              // ข้อมูลกำหนดมุม
                fscanf (fs, "%f %f %f",&x, &y, &z);
                Rotate_Object(x, y, z);
            }
            if (!strcmp(cmd,"SCALE")) {                 // ข้อมูลกำหนดขนาด
                fscanf (fs, "%f %f %f",&x, &y, &z);
                Scale_Object(x, y, z);
            }
        } while (strcmp(cmd,"END"));
        fclose (fs);
}

void Load_Display (char *s)                             // อ่านข้อมูลกำหนดการแสดงภาพ
{
        FILE    *fs;
        char    cmd[16];
        float   x,y,z;
        int     vl,vr,vt,vb;
        int     n,color;
        if ((fs=fopen(s,"rt"))==NULL) return;
        fscanf (fs, "%s", cmd);
        if (strcmp(cmd,"DISPLAY")) return;
        do {
            fscanf (fs,"%s", cmd);
            if (!strcmp(cmd,"LIGHT")) {                 // กำหนดแสง
                fscanf (fs, "%f %f %f",&x, &y, &z);
                Light_Direction (x, y, z);
            }
            if (!strcmp(cmd,"WINDOW")) {
                fscanf (fs, "%d %d %d %d",&vl, &vt, &vr, &vb);
```

```
            Set_"Window (vl, vt, vr, vb);
        }
        if (!strcmp(cmd,"VIEW")) {              // กำหนดมุมมองภาพ
            fscanf (fs, "%f %f %f",&x, &y, &z);
            View_Position (x, y, z);
            fscanf (fs, "%f %f %f",&x, &y, &z);
            View_Direction (x, y, z);
            fscanf (fs, "%f %f %f",&x, &y, &z);
            View_Up (x, y, z);
        }
        if (!strcmp(cmd,"OBJECT")) {            // เลือกวัตถุ
            fscanf (fs, "%d", &n);
            Select_Object (n);
        }
        if (!strcmp(cmd,"POSITION")) {          // ตำแหน่งวัตถุ
            fscanf (fs, "%f %f %f",&x, &y, &z);
            Translate_Object(x, y, z);
        }
        if (!strcmp(cmd,"ROTATION")) {                  // มุมของวัตถุ
            fscanf (fs, "%f %f %f",&x, &y, &z);
            Rotate_Object(x, y, z);
        }
        if (!strcmp(cmd,"SCALE")) {             // ขนาด
            fscanf (fs, "%f %f %f",&x, &y, &z);
            Scale_Object(x, y, z);
        }
        if (!strcmp(cmd,"FORECOLOR"))   {
            fscanf (fs,"%d",&color);
            Set_ForeColor (color);
        }
```

```
                if (!strcmp(cmd,"BACKCOLOR")) {
                        fscanf (fs,"%d",&color);
                        Set_BackColor (color);
                }
                if (!strcmp(cmd,"HIDE"))        Hide_Object ();
                if (!strcmp(cmd,"SHOW"))  Show_Object ();
                if (!strcmp(cmd,"GREY"))   Set_Color ("GREY.PAL");
                if (!strcmp(cmd,"BACK"))   Show_BackFace ();
                if (!strcmp(cmd,"NOBACK")) Hide_BackFace ();

                if (!strcmp(cmd,"SOLID"))   Solid_Object ();
                if (!strcmp(cmd,"WIRE"))    Wire_Object ();
                if (!strcmp(cmd,"CLEAR"))  Clear_Display ();
                if (!strcmp(cmd,"MAKE"))   Make_Picture ();
        } while (strcmp(cmd,"END"));
        fclose (fs);
}

void main ( )
{
        int   status=0;

        if  (argc<3)  status = 1;
        else     if  (Init_Graph ( ))  status = 2;
                else     if  (Load_Image (argv[1]))  status = 3;
                        else     if (Load_Display (argv[2]))  statuse = 4;
        Exit_Graph (status);
}
```

ในการใช้โปรแกรมนี้ ผู้ใช้เพียงแต่กำหนดข้อมูลในแฟ้มต่าง ๆ ให้ถูกต้อง ก็สามารถแสดงภาพที่ต้องการออกมาได้ เริ่มจากการกำหนดระบบจอภาพในแฟ้ม CONFIG.GSW ดังนี้

```
CONFIG
        TYPE        1
        MODE        93
END CONFIG
```

ขั้นต่อมาก็เตรียมแฟ้มเลือกวัตถุภาพ ตัวอย่างเช่น TEST.IMG ดังนี้

```
IMAGE
        LOAD  CUBE.DAT                  // เลือกแฟ้มวัตถุภาพ
        POSITION    0.2     -0.3    -0.2
        ROTATION    30      -30     0
        SCALE       0.8     0.8     0.8
END IMAGE
```

ขั้นตอนสุดท้ายได้แก่ การสร้างแฟ้มแสดงภาพ ตัวอย่างเช่น แฟ้ม TEST.DSP จะมีลักษณะดังนี้

```
DISPLAY
    PALETTE     GREY.PAL
    WINDOW      150  100  400  350    //พื้นที่แสดงภาพ
    VIEW        0    0    -1         //ตำแหน่งผู้มองภาพ
                0    0    1          //ทิศทางการมอง
                0    1    0          //แนวตั้ง
    WIRE                             //แสดงภาพแบบโครงลวด
    CLEAR                            //ลบพื้นที่แสดงภาพ
    MAKE                             //แสดงภาพ
END DISPLAY
```

ในส่วนของแฟ้มข้อมูลวัตถุภาพ "CUBE.DAT" จะมีลักษณะดังนี้

```
OBJECT
NODE        10      2       0       0       0       // จำนวนจุด,ขนาด
        -1      -1      -1                              // ตำแหน่งจุด
        1       -1      -1
        1       0       -1
        0       1       -1
        -1      1       -1
        -1      -1      1
        1       -1      1
        1       1       1
        -1      1       1
        1       1       0
POLYGON     7                                           //จำนวนหน้า
        5       0       1       2       3       4       //องค์ประกอบหน้า 1
        4       8       7       6       5
        4       0       4       8       5
        5       4       3       9       7       8
        3       3       2       9
        5       2       1       6       7       9
        4       1       0       5       6
COLOR15                                                 // สีของวัตถุ
END OBJECT
```

เมื่อเตรียมแฟ้มต่าง ๆ เรียบร้อยแล้ว ในการเรียกโปรแกรมมาทำงานก็จะระบุชื่อแฟ้มที่ต้องการดังนี้

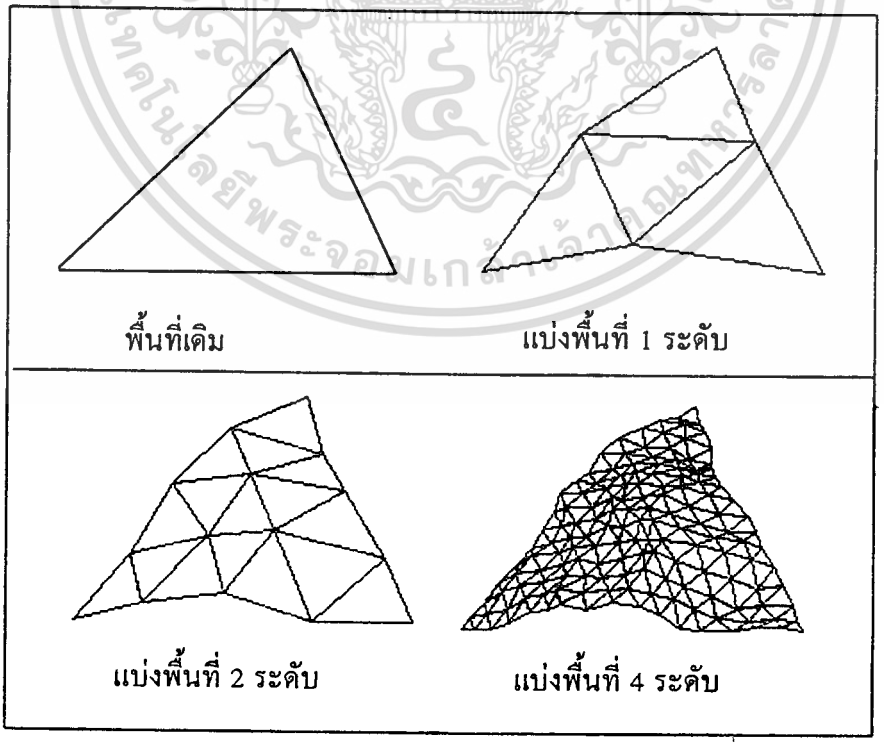
```
APPLY   TEST.IMG   TEST.DAT
```

โดยที่ APPLY ก็คือชื่อของส่วนใช้งานในตัวอย่างนี้

ภาคผนวก จ
การสร้างพื้นผิวไม่เรียบ (Fractal Surfaces)

การทำพื้นผิวไม่เรียบ (fractal surfaces) ด้วยการใช้รูปเหลี่ยมเล็ก ๆ จำนวนมากเรียงติดต่อกันอย่างไม่เป็นระเบียบ จะเริ่มจากการแบ่งพื้นที่สามเหลี่ยมเดิมที่มีขนาดใหญ่ออกเป็นสามเหลี่ยมเล็กสี่รูปด้วยการแบ่งครึ่งด้านแต่ละด้านของสามเหลี่ยม โดยให้จุดที่แบ่งเบี่ยงเบนออกจากเส้นเดิมเล็กน้อยเพื่อให้ สามเหลี่ยมที่ได้มีขนาดและรูปร่างที่แตกต่างกัน ดังภาพที่ 64 ด้วยวิธีการเรียกส่วนการทำงานเดิมซ้อนกัน ทำให้สามารถกำหนดระดับของการแบ่งสามเหลี่ยมแต่ละรูปลงไปกี่ระดับก็ได้ตามต้องการ

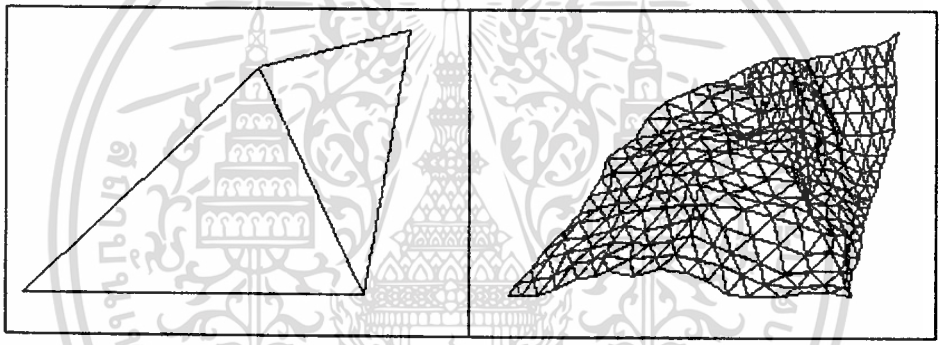
ภาพที่ 64
การแบ่งย่อยพื้นที่สามเหลี่ยมโดยกำหนดระดับ



ปัญหาของการแบ่งรูปสามเหลี่ยมโดยการกำหนดระดับการแบ่งพื้นที่ จะเกิดขึ้นเมื่อพื้นที่สามเหลี่ยมเดิมสองรูปซึ่งอยู่ติดกันมีขนาดที่ต่างกันมาก เมื่อแบ่งพื้นที่ออกไปโดยกำหนดจำนวนระดับการแบ่งเท่ากันแล้ว รูปสามเหลี่ยมย่อยของแต่ละพื้นที่ก็จะมีขนาดต่างกัน ทำให้ได้ภาพที่ไม่ต่อเนื่องดังภาพที่ 65

ภาพที่ 65

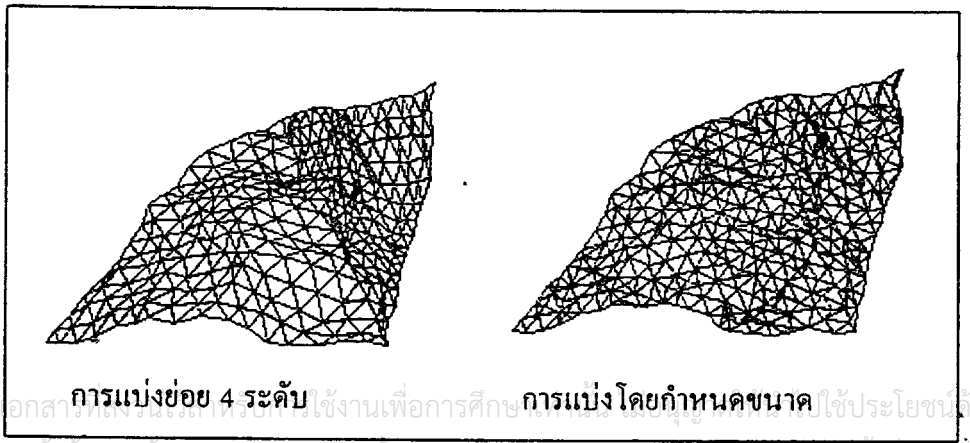
พื้นผิวที่เกิดจากรูปสามเหลี่ยมสองรูปที่มีขนาดต่างกัน



ในการแก้ปัญหานี้ จะใช้วิธีควบคุมขนาดของรูปสามเหลี่ยมย่อยแทนการกำหนดระดับการแบ่ง เมื่อนำภาพพื้นผิวที่ได้วิธีทั้งสองมาเปรียบเทียบกันดังภาพที่ 66 จะเห็นว่าภาพของพื้นผิวมีความต่อเนื่องกันมากกว่าภาพที่ได้จากวิธีแรก

ภาพที่ 66

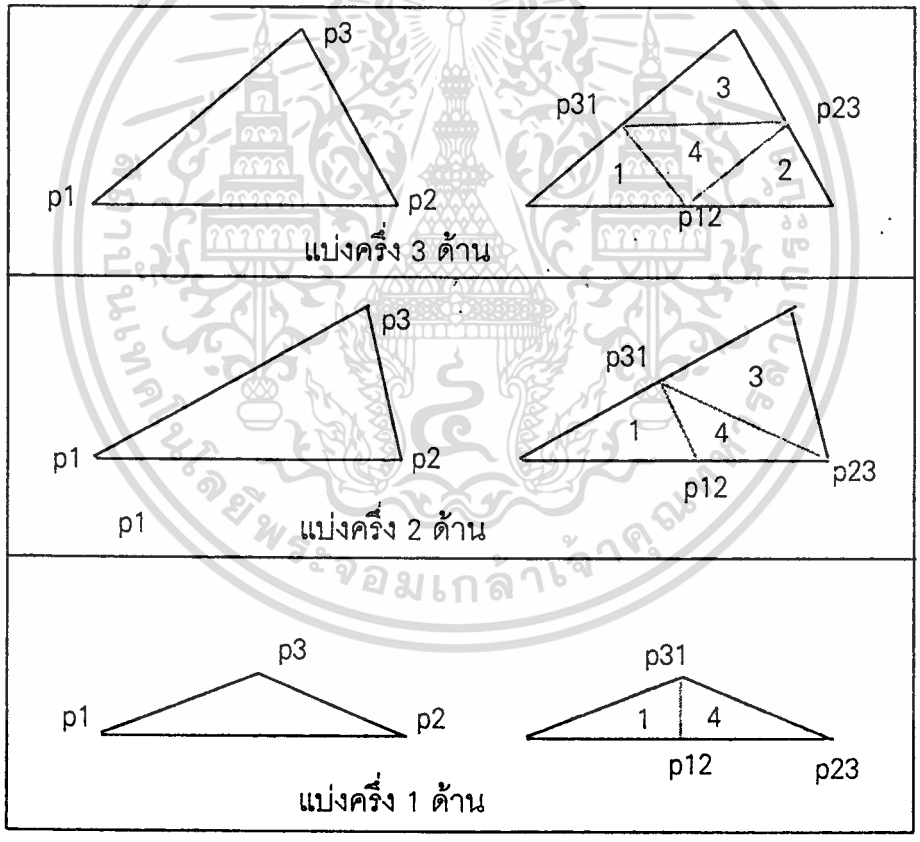
เปรียบเทียบภาพที่ได้จากวิธีการทั้งสอง



ในการแบ่งสามเหลี่ยมโดยกำหนดขนาด จากเดิมที่ต้องแบ่งครึ่งด้านทั้งสามของรูปสามเหลี่ยม ซึ่งจะได้รูปสามเหลี่ยมย่อย 4 รูป ก็จะแบ่งเฉพาะด้านที่มีความยาวมากกว่าระดับที่กำหนดไว้เท่านั้น จำนวนรูปสามเหลี่ยมย่อยจะขึ้นอยู่กับจำนวนด้านที่ถูกแบ่งครึ่งดังภาพที่ 67

ภาพที่ 67

การแบ่งพื้นที่สามเหลี่ยมโดยการตรวจสอบขนาด



ในส่วนของระยะเบี่ยงเบนของจุดแบ่งก็จะใช้ความยาวของด้านที่ถูกแบ่งเป็นตัวกำหนด โดยมีขั้นตอนการทำงานดังนี้

```
procedure Sub_Fractal (p1, p2, pt3, weight, size)
{
        d1 = Get_Distance (p1, p2)                    // หาความยาวด้านทั้งสาม
        d2 = Get_Distance (p2, p3)
        d3 = Get_Distance (p3, p1)

        if (d1 > size)
                p12 =  Sub_Divide (p1, p2, weight);     // แบ่งครึ่งด้าน
        else
                p12 = p1
        if (d2 > size)
                p23 =  Sub_Divide (p2, p3, weight);
        else
                p23 = p2
        if (d3 > size)
                p31 =  Sub_Divide (p3, p1, weight);
        else
                p31 = p3
        if  (p12=p1)                                  // ไม่แบ่งด้านที่หนึ่ง
                if  (p23=p2)                          // ไม่แบ่งด้านที่สอง
                        if  (p31=p3)                  // ไม่แบ่งด้านที่สาม
                                Draw_Triangle (p1, p2, p3);
                                return
                        else    Sub_Fractal (p1, p31, p12)
                else    Sub_Fractal (p2, p23, p31)
        else    Sub_Fractal (p3, p12, p31)
        Sub_Fractal (p12, p23, p31)
}
```

ด้วยวิธีการนี้นอกจากจะสามารถควบคุมขนาดของรูปสามเหลี่ยมย่อยไม่ให้เล็กมากเกินไป สำหรับความละเอียดของจอภาพ จนรายละเอียดของภาพเริ่มจะหายไปแล้ว รูปสามเหลี่ยมแต่ละรูปก็จะมีขนาดใกล้เคียงกัน

ประวัติผู้เขียน

ชื่อผู้เขียน	นายสุวัฒน์ ศรีธนะรัตน์
วัน เดือน ปีเกิด	วันที่ 20 พฤศจิกายน พ.ศ.2497
วุฒิการศึกษาระดับปริญญาตรี	วิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมไฟฟ้า
สถานที่สำเร็จการศึกษา	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีที่สำเร็จการศึกษา	2528
ประสบการณ์การทำงาน	วิศวกรไฟฟ้า โปรแกรมเมอร์ นักวิเคราะห์ระบบ
อาชีพปัจจุบัน	วิเคราะห์ระบบและเขียนโปรแกรม