แบบจำลองการส่องสว่างของวัตถุในคอมพิวเตอร์กราฟิก

OBJECTS' ILLUMINATION MODELS IN COMPUTER GRAPHICS

นายชฎิล แก้วปลั่ง

MR.CHADIN KEOPLUNG

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2539

ISBN 974-621-758-5

OBJECTS' ILLUMINATION MODELS IN COMPUTER GRAPHICS

CHADIN KEOPLUNG

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 974-621-758-5

หัวข้อวิทยานิพนธ์	แบบจำลองการส่องสว่างของวัตถุในคอมพิวเตอร์กราฟิก
นักศึกษา	นายชฎิล แก้วปลั่ง
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.ดร.กิตติ ไพฑูรย์วัฒนกิจ
ระดับการศึกษา	วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า
ภาควิชา	วิศวกรรมคอมพิวเตอร์
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.	2539

บทคัดย่อ

แบบจำลองการส่องสว่างของวัตถุในคอมพิวเตอร์กราฟิก เป็นการสร้างภาพจำลองของวัตถุที่เกิดจากการที่มีแสงจากต้นกำเนิดแสงใดๆมาตกกระทบที่พื้นผิวของวัตถุ ภาพจำลองการส่องสว่างที่ได้เป็นภาพจำลองในระบบกราฟิก 3 มิติ โดยอาศัยหลักการจำลองภาพวัตถุด้วยวิธี Ray Tracing และ Radiosity ในการสร้างภาพจำลอง และรวมถึงการศึกษาลักษณะของการเกิดเงาของวัตถุจากต้นกำเนิดแสงที่มีลักษณะการกระจายค่าความเข้มของแสงในแบบต่างๆ เพื่อทำให้เกิดเงาในลักษณะ Soft shadow หรือ เงาแบบเรียบ แทนการเกิดเงาแบบ Hard edge shadow หรือ เงาแบบขอบแข็งซึ่งเป็นลักษณะของเงาที่เกิดจากต้นกำเนิดแสงแบบจุด (Point light source) ทั้งนี้เพื่อเป็นการปรับปรุงคุณสมบัติของแบบจำลองการส่องสว่างของวัตถุให้มีความเหมือนจริงมากขึ้น

THESIS TITLE	OBJECTS' ILLUMINATION MODELS IN COMPUTER GRAPHICS
STUDENT	MR.CHADIN KEOPLUNG
THESIS ADVISOR	ASSOC. PROF. DR. KITTI PAITOONWATANAKIT
LEVEL OF STUDY	MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
DEPARTMENT	COMPUTER ENGINEERING
	KING MONGKUT'S INSTITUTE OF TECHNOLOGY
	LADKRABANG
YEAR	1996

Abstract

Objects' illumination models are the studying of the property of objects which are projected by light from any light source. The intensity of light, which reflects from the surface of objects, illustrats the effect on property of the objects. The shadow provides another result of light sources in which hard edge shadow is created by any point light source. When the intensity of light sources is distributed in different models, the soft shadow which will be more realistic has been created. The images of objects' illumination models are created by classical Ray Tracing and Radiosity schemes in three dimensional graphics.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยความช่วยเหลือจากบุคคลหลายฝ่าย ผู้วิจัยขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ ที่นี้ โดยเฉพาะอย่างยิ่ง รองศาสตราจารย์ ดร.กิตติ ไพฑูรย์วัฒนกิจที่ได้กรุณาให้ความอนุเคราะห์ในการใช้อุปกรณ์ระบบไมโครคอมพิวเตอร์ และให้คำแนะนำรวมทั้งตรวจสอบความถูกต้องตลอดการดำเนินการวิจัย

ชฎิล แก้วปลั่ง

สารบัญ

หน้า

บทที่ หน้า

บทที่ หน้า

บทที่ หน้า

สารบัญตาราง

สารบัญภาพ

หน้า

หน้า

หน้า

หน้า

สารบัญภาพ (ต่อ)

หน้า

บทที่ 1
บทนำ

ความเป็นมาและความสำคัญของปัญหา

การสร้างภาพจำลองของวัตถุโดยอาศัยวิธีการทางคอมพิวเตอร์กราฟิกมีหลายวิธีซึ่งแต่ละวิธีจะให้ผลที่มีคุณสมบัติแตกต่างกัน เช่น ความหยาบของภาพ แสง (Light) และเงา (Shadow) ที่ปรากฏบนภาพจำลอง คุณสมบัติของภาพจำลองที่สร้างขึ้นนั้น จะขึ้นอยู่กับความต้องการของผู้ใช้ว่าต้องการจะนำไปใช้ในเรื่องใด การสร้างภาพจำลองที่มีลักษณะเหมือนจริงจะสามารถนำมาใช้ในบางเรื่องเท่านั้น เช่น การออกแบบเพื่อใช้ในงานทางด้านวิทยาศาสตร์หรือทางการแพทย์ ซึ่งจะต้องคำนึงถึงความถูกต้องของภาพจำลองที่ได้มาก ว่ามีความเหมือนกับวัตถุต้นแบบเพียงใด การพิจารณาเรื่องแสงและเงาของภาพจำลองที่สร้างขึ้นก็เป็นเรื่องที่สำคัญมากอีกเรื่องหนึ่งที่จะต้องพิจารณา เนื่องจากแสงและเงาจะเป็นสิ่งที่แสดงให้เห็นว่าภาพจำลองมีมิติหรือรูปร่างเป็นอย่างไร

วัตถุประสงค์ของการวิจัย

โดยทั่วไปแล้วแบบจำลองการส่องสว่างของวัตถุที่ใช้ในการสร้างภาพจำลองที่เหมือนจริงจะมีวิธีต่างๆ แยกกันไปตามวิธีที่ใช้ในการจำลองภาพ เงาของวัตถุที่เกิดขึ้นในภาพจำลองก็จะมีลักษณะที่แตกต่างกัน เงาของภาพจำลองที่เกิดขึ้นนั้นโดยมากจะมีลักษณะเป็นเงาแบบขอบแข็ง ซึ่งมีค่าความเข้มของแสง (Intensity) ในบริเวณที่เป็นเงาเท่ากันทั้งหมด แต่เงาที่เกิดขึ้นกับวัตถุจริงนั้นจะมีลักษณะที่แตกต่างออกไป คือ จะมีค่าความเข้มของแสงในส่วนที่เป็นบริเวณของเงาไม่เท่ากัน และค่าความเข้มของแสงในบริเวณเงานี้จะมีการเรียงตัวกันของค่าความเข้มของแสงในสัดส่วนที่เหมาะสม คือที่บริเวณนอกสุดของส่วนที่เป็นเงาจะมีค่าความเข้มของแสงมากที่สุดเมื่อเปรียบเทียบกับค่าความเข้มของแสงในส่วนที่เป็นเงา และค่าความเข้มของแสงของเงาจะมีค่าลดลงเมื่ออยู่ในบริเวณตำแหน่งที่ใกล้เคียงจุดศูนย์กลางของเงา ดังนั้นการสร้างภาพจำลองของเงาเพื่อให้เกิดเงาตามลักษณะที่กล่าวมา จึงต้องอาศัยวิธีการบางอย่างที่แตกต่างไปจากวิธีการจำลองภาพแบบเดิมมาช่วยในการจำลองภาพ คือการเปลี่ยนแปลงคุณสมบัติบางอย่างของต้นกำเนิดแสง (Light source) เช่นการเปลี่ยนแปลงตำแหน่งส่องสว่าง หรือการเปลี่ยนแปลงค่าความเข้มของแสงของต้นกำเนิดแสง

แนวคิดที่ใช้ในการวิจัย

วิธีการสร้างภาพจำลองของวัตถุให้มีความเหมือนจริงนั้นสามารถแยกเป็น 2 หลักการคือ วิธีการสร้างภาพจำลองแบบ Radiosity ที่อาศัยคุณสมบัติการแลกเปลี่ยนพลังงานความร้อนระหว่างวัตถุมาประยุกต์ในการสร้างภาพจำลอง และวิธีการสร้างภาพจำลองแบบ Ray Tracing ที่อาศัย

หลักการของการพิจารณาทิศทางของลำแสงจากต้นกำเนิดแสงที่มาตกกระทบกับวัตถุแล้วสะท้อนมาสู่ผู้สังเกต

การสร้างภาพจำลองในส่วนที่เป็นเงานั้นจะแยกออกเป็น 2 ส่วนคือ การสร้างเงาแบบขอบแข็งจากต้นกำเนิดแสงแบบจุด และการสร้างเงาแบบเรียบจากต้นกำเนิดแสงแบบจุดโดยการกระจายตำแหน่งส่องสว่างและการเปลี่ยนแปลงค่าความเข้มของแสงของต้นกำเนิดแสง โดยอาศัยวิธีการสร้างภาพจำลองแบบ Ray Tracing

ขอบเขตของการวิจัยและวิธีที่ใช้ในการวิจัย

การวิจัยนี้เป็นการสร้างภาพจำลองที่เหมือนจริงของวัตถุโดยอาศัยวิธีการต่างๆ ภาพจำลองที่สร้างขึ้นนั้นสร้างบนเครื่องไมโครคอมพิวเตอร์ 80486/100 Mhz. และใช้ภาษา C ในการสร้างโปรแกรม

วิธีการต่างๆที่ใช้ในการสร้างภาพจำลองในการวิจัยนี้ จะรวมถึงการสร้างแบบจำลองในระบบ 3 มิติ การกำหนดระดับความสัมพันธ์ของสีที่ใช้ การกำหนดลักษณะของพื้นผิวและลวดลายบนพื้นผิวของวัตถุและหลักการการสร้างภาพจำลองโดยวิธี Radiosity และวิธี Ray Tracing ที่ใช้ในการสร้างภาพจำลองของเงาแบบเรียบ โดยการเปลี่ยนแปลงคุณสมบัติของต้นกำเนิดแสง

วิทยานิพนธ์ฉบับนี้มีวัตถุประสงค์ที่จะพัฒนาการจำลองภาพของวัตถุให้ดียิ่งขึ้นเพื่อประโยชน์ในการจำลองภาพต่างๆ รายละเอียดของวิทยานิพนธ์นี้สามารถแยกออกได้เป็นส่วนๆ ตามพื้นฐานของวิธีการจำลองภาพคือ จะกล่าวถึงระบบคอมพิวเตอร์กราฟิกที่เป็นพื้นฐานและการพัฒนาระบบคอมพิวเตอร์กราฟิก การกำหนดคุณสมบัติต่างๆของของวัตถุในระบบกราฟิก 2 มิติ และระบบ 3 มิติเพื่อใช้ในการสร้างแบบจำลองของวัตถุ การกำหนดคุณสมบัติของแสงและแบบจำลองของสีแบบต่างๆ รวมถึงวิธีการจัดลำดับค่าความเข้มของแสงซึ่งจะต้องนำมาใช้ในการกำหนดค่าความเข้มของวัตถุในภาพจำลอง รวมถึงหลักการพื้นฐานของการสร้างภาพจำลองโดยวิธี Radiosity การสร้างภาพจำลองโดยวิธี Ray Tracing และการสร้างภาพจำลองของเงาแบบเรียบในแบบต่างๆกัน

บทที่ 2
ความรู้พื้นฐานเกี่ยวกับคอมพิวเตอร์กราฟิก

การพัฒนาอย่างมีประสิทธิภาพในด้านต่างๆ ไม่ว่าจะเป็นทางด้านวิทยาศาสตร์ ธุรกิจหรืองานด้านใดๆก็ตาม ต้องสามารถสื่อสารกับผู้รับให้สามารถเข้าใจได้โดยง่าย ในปัจจุบันเครื่องคอมพิวเตอร์ได้ถูกนำมาใช้เพื่อการนี้มากขึ้น เนื่องจากคุณสมบัติในด้านต่างๆ ของคอมพิวเตอร์ เช่น ประสิทธิภาพในการคำนวณ การแสดงผล และอื่นๆ และที่สำคัญคือวิธีการสื่อสาร ซึ่งใช้ภาพในการแสดงผลลัพธ์ ดังนั้นระบบคอมพิวเตอร์กราฟิก (Computer Graphics) จึงถูกนำมาใช้ในการช่วยแสดงผลของแบบจำลองของงานต่างๆไม่ว่าจะเป็น รูปกราฟ หรือภาพจำลองของวัตถุต่างๆ จากที่กล่าวข้างต้นสามารถที่จะให้คำจำกัดความง่ายๆของระบบคอมพิวเตอร์กราฟิก คือระบบงานใดๆที่ใช้เครื่องคอมพิวเตอร์และอุปกรณ์ต่างๆที่เกี่ยวข้องมาช่วยในการประมวลผลข้อมูลแล้วแสดงผลเป็นภาพกราฟิกหรือในทางตรงกันข้ามที่นำข้อมูลจากภาพทางกราฟิกมาช่วยในการประมวลผลเพื่อที่จะนำผลที่ได้มาใช้ในงานต่างๆ

ภาพที่ 1

แสดง เครื่องไมโครคอมพิวเตอร์

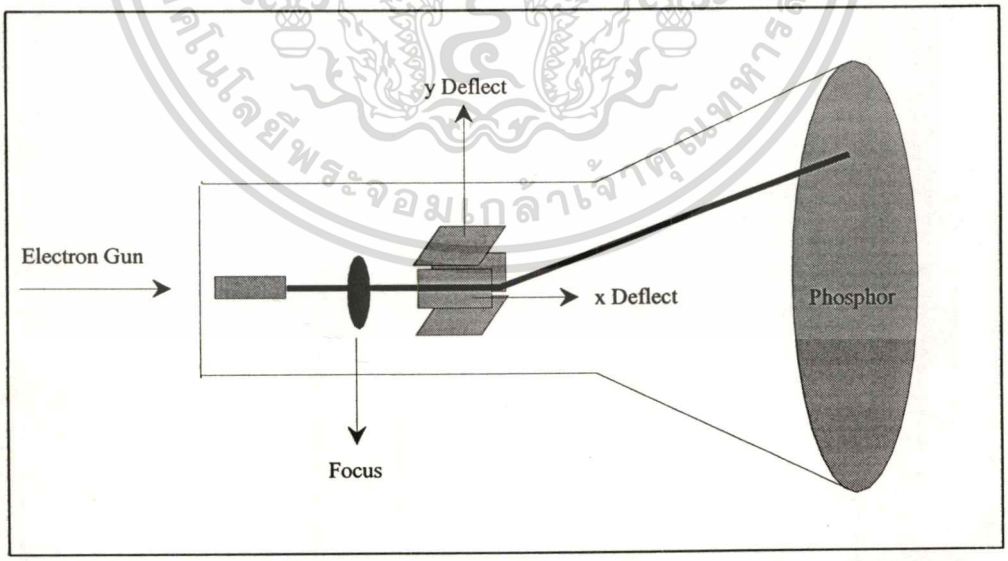
จากคำจำกัดความข้างต้น จะเห็นว่าการพัฒนาระบบคอมพิวเตอร์กราฟิกนั้นไม่กำหนดว่าเป็นการพัฒนาทางด้าน ซอฟร์แวร์ (Software) หรือ ขั้นตอนวิธี (Algorithm) ในการจำลองภาพต่างๆเท่านั้น แต่ยังรวมถึงงานด้านการออกแบบระบบ ฮาร์ดแวร์ (Hardware) เช่น จอภาพ หน่วยความจำที่ใช้ในการแสดงผล หรือ หน่วยประมวลผลข้อมูล ด้วย ซึ่งปัจจุบันนี้เครื่องคอมพิวเตอร์ได้ถูกพัฒนาให้มีประสิทธิภาพสูงขึ้นโดยที่มีขนาดเล็กเมื่อเทียบจากที่ผ่านมา ภาพที่ 1 แสดงลักษณะของเครื่องไมโครคอมพิวเตอร์ (Micro Computer) ที่มีประสิทธิภาพในการทำงานสูงและมีขนาดเล็ก

การพัฒนาระบบคอมพิวเตอร์กราฟิก

วิวัฒนาการของระบบคอมพิวเตอร์กราฟิก[1] สามารถที่จะสรุปได้ดังนี้

คศ. 1950-1960 มีการพัฒนาการควบคุมทิศทางของลำอิเลคตรอน (Electron beam) ในหลอดลำแสงแคโทด (Cathode Ray tube; CRT) ดังแสดงในภาพที่ 2 เมื่อลำของอิเลคตรอนกระทบกับสารฟอสเฟอร์ (Phosphor) ที่เคลือบที่บริเวณผิวของหลอดภาพจะทำให้เกิดการเรืองแสงขึ้นและทิศทางของอิเลคตรอนจะถูกควบคุมให้มีการเปลี่ยนแปลงโดยแผ่นที่ใช้ในการควบคุมทิศทาง 2 คู่ ทำให้สามารถที่จะควบคุมตำแหน่งการเรืองแสงของจอภาพ CRT ได้ในตำแหน่งที่ต้องการ แต่การเรืองแสงของจอภาพนั้นจะเกิดขึ้นในช่วงเวลาที่ไม่นาน

ภาพที่ 2

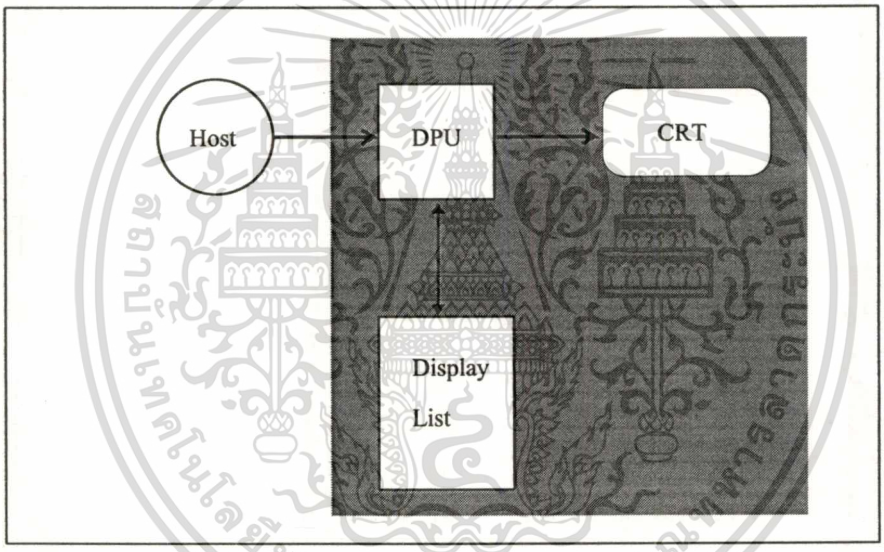


แสดง หลอดลำแสงคาโทด

คศ. 1960-1970 ได้พัฒนาหลอดภาพชนิดใหม่ที่เรียกว่า DVST (Direct View Storage Tube) หลอดภาพชนิดนี้มีคุณสมบัติที่ดีกว่าหลอดภาพแบบ CRT เพราะจะให้เวลาในการส่องสว่างของจุดบนจอภาพได้นานกว่า แต่ก็ยังไม่เพียงพอกับความต้องการ จากนั้นได้มีการพัฒนาอุปกรณ์ที่ใช้ในเป็นตัวประมวลผลการแสดงผล (Display processor) ช่วยในการแสดงผลของภาพบนจอคอมพิวเตอร์ โดยภาพกราฟิกที่จำลองขึ้นจะถูกสร้างและประมวลผลที่เครื่องคอมพิวเตอร์แม่ (Host computer) และข้อมูลที่ได้จะถูกเก็บไว้ในแฟ้มข้อมูลของการแสดงผล (Display file)

ดังนั้นเครื่องคอมพิวเตอร์แม่จะถูกใช้ในการสร้างภาพจำลองเพียงครั้งเดียวและภาพที่ถูกเก็บในแฟ้มข้อมูลของการแสดงผลจะถูกนำมาประมวลผลโดย หน่วยแสดงผล หรือ DPU (Display Processing Unit) จากนั้นผลลัพธ์ที่ได้จะถูกนำมาแสดงผลที่จอภาพ ดังแสดงในภาพที่ 3

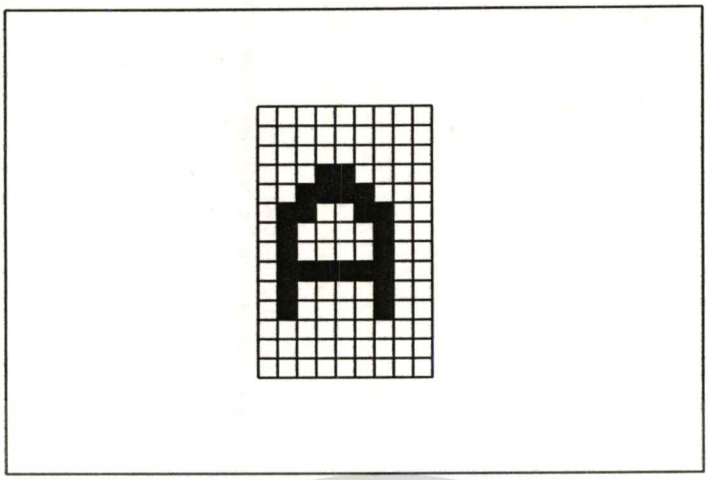
ภาพที่ 3



แสดง สถาปัตยกรรมตัวประมวลผลการแสดงผล

คศ. 1970-1980 มีการพัฒนาระบบคอมพิวเตอร์กราฟิกแบบใหม่เรียกว่า Raster graphics โดยอาศัยผลของวิวัฒนาการของหน่วยความจำแบบ Solid state ทำให้ต้นทุนในการผลิตต่ำลง ในระบบคอมพิวเตอร์กราฟิกแบบนี้ข้อมูลของภาพจะถูกเก็บในรูปชุดของจำนวนจุดภาพ (Pixel) ต่างๆ ของภาพ ดังแสดงในภาพที่ 4 เป็นการเก็บภาพของตัวอักษร แทนการเก็บภาพแบบเดิมที่เก็บลักษณะของเส้นต่างๆที่ประกอบเป็นภาพ[2] จุดต่างๆที่ถูกสร้างขึ้นมาจะถูกเก็บไว้ในหน่วยความจำที่เรียกว่า Frame buffer การแสดงผลทางจอภาพสามารถทำได้โดย การกำหนดให้จุดบนจอภาพมืดหรือสว่างตามข้อมูลที่ได้จาก Frame buffer

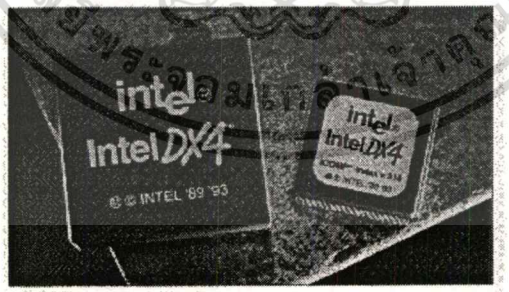
ภาพที่ 4



แสดง ชุดของจุดของตัวอักษร A

คศ. 1980 คอมพิวเตอร์ได้ถูกพัฒนาอย่างรวดเร็วทั้งในด้านประสิทธิภาพที่สูงขึ้นและขนาดที่เล็กลง ตัวประมวลผลส่วนกลาง (Central processing unit) ที่ใช้ในปัจจุบันเป็นอุปกรณ์ที่ผลิตขึ้นมาจากขบวนการของ Solid state ทำให้มีขนาดเล็กมาก ตัวประมวลผลส่วนกลางเพียงตัวเดียวมีประสิทธิภาพเทียบได้กับหลอดสูญญากาศที่ใช้ในเครื่องคอมพิวเตอร์สมัยแรกๆเป็นจำนวนหลายพันตัว ภาพที่ 5 แสดงหน่วยประมวลผลส่วนกลางที่ใช้ในเครื่องคอมพิวเตอร์

ภาพที่ 5



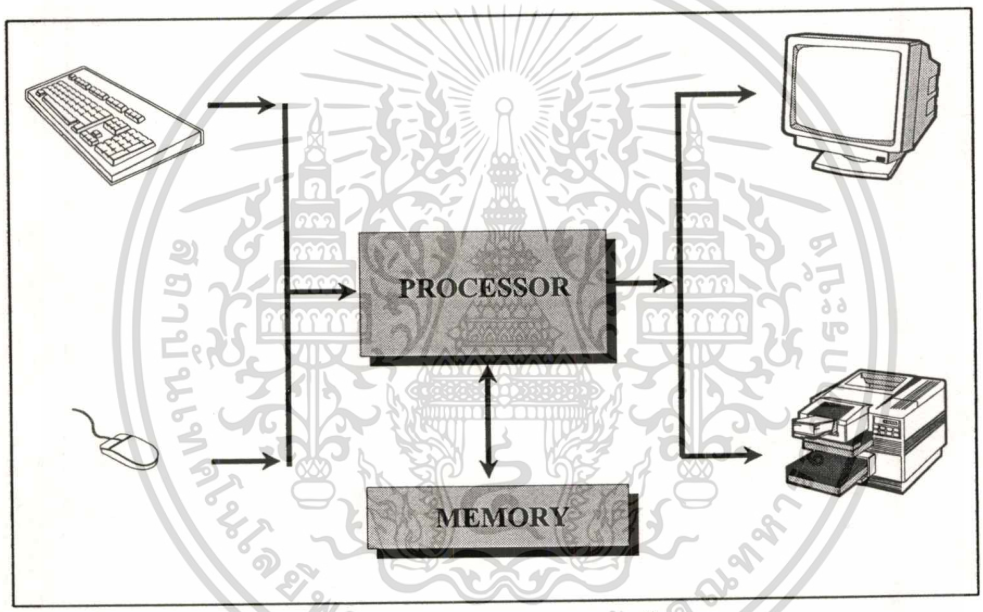
แสดง ตัวประมวลผลส่วนกลาง

ระบบคอมพิวเตอร์กราฟิกพื้นฐาน

ระบบคอมพิวเตอร์กราฟิกที่ใช้งานในปัจจุบันนั้นจะมีส่วนประกอบพื้นฐาน[3]ที่เป็นหลักอยู่ 4 ส่วนด้วยกันดังแสดงในภาพที่ 6 คือ

1. ตัวประมวลผล (Processor)
2. หน่วยความจำ (Memory)
3. อุปกรณ์แสดงผล (Output device)
4. อุปกรณ์รับข้อมูล (Input device)

ภาพที่ 6



แสดง ระบบคอมพิวเตอร์กราฟิกพื้นฐาน

ตัวประมวลผล อุปกรณ์ส่วนตัวประมวลผลเป็นส่วนที่ใช้ในการประมวลผลข้อมูลต่างๆ ที่ใช้ในการจำลองภาพ เช่น การคำนวณคุณสมบัติต่างๆของวัตถุในภาพจำลอง หรือใช้ในการแปรคำสั่งต่างๆที่ได้รับจากโปรแกรมควบคุม แล้วนำมาใช้ในการสร้างภาพจำลอง

หน่วยความจำ อุปกรณ์หน่วยความจำจะทำหน้าที่เป็นส่วนที่ใช้ในการเก็บหรือพักข้อมูลต่างๆ เพื่อใช้ในการคำนวณ การแสดงผล หรือ การรับข้อมูล ในอุปกรณ์ส่วนนี้จะไม่มีการคำนวณเลย

อุปกรณ์แสดงผล ใช้ในการแสดงผลของข้อมูลที่ได้จากการประมวลผล หรือ แสดงผลของข้อมูลที่รับ เพื่อให้ผู้ใช้ทราบ ผลของข้อมูลต่างๆอาจเป็นรูปภาพหรือตัวอักษรก็ได้ อุปกรณ์เหล่านี้ได้แก่ จอภาพ (Monitor) เครื่องพิมพ์ (Printer) เครื่องลงจุด (Plotter) หรืออุปกรณ์ใดๆที่สามารถเป็นสื่อทำให้ผู้ใช้เข้าใจในผลลัพธ์ที่ได้

อุปกรณ์รับข้อมูล เป็นอุปกรณ์ที่ใช้เป็นตัวรับข้อมูลจากส่วนต่างๆ แล้วส่งข้อมูลให้เครื่องคอมพิวเตอร์นำข้อมูลนั้นไปใช้งานต่างๆ เช่น แป้นพิมพ์ (Keyboard) หรือเครื่องกวาดตรวจ (Scanner)

สรุป

จากที่กล่าวไปแล้วนั้นจะเห็นว่า การพัฒนาของระบบคอมพิวเตอร์ในช่วงเวลาที่ผ่านมาเป็นไปอย่างรวดเร็ว เพื่อสนองตอบต่อความต้องการของผู้ใช้และ เพื่อเป็นการเพิ่มประสิทธิภาพของการทำงานต่างๆโดยใช้คอมพิวเตอร์ ส่วนวิธีการคำนวณต่างๆที่ใช้ในระบบคอมพิวเตอร์กราฟิกนั้นจะได้กล่าวถึงต่อไปเพื่อความเข้าใจในระบบคอมพิวเตอร์กราฟิกให้ดีขึ้น

บทที่ 3

กราฟิก 2 มิติ และ กราฟิก 3 มิติ

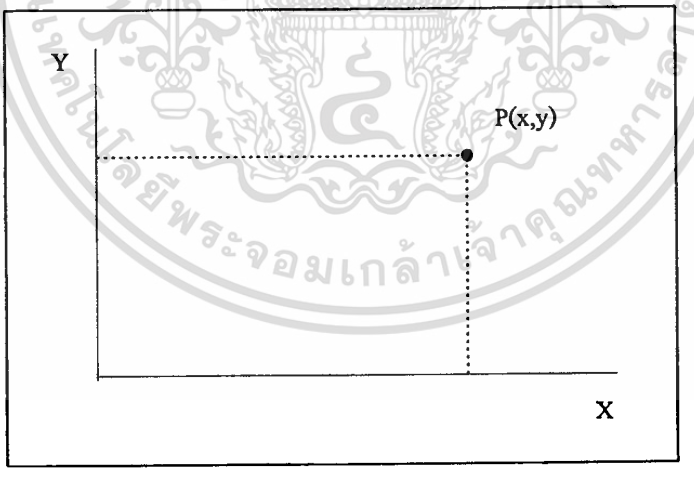
กราฟิก 2 มิติ

ระบบที่เป็นพื้นฐานของระบบคอมพิวเตอร์กราฟิก คือ ระบบกราฟิก 2 มิติ[4] เพราะภาพที่ได้จากระบบกราฟิก 2 มิติ นี้สามารถนำไปใช้ประโยชน์ได้มากมาย เช่น รูปกราฟต่าง ๆ ในระบบกราฟิก 2 มิตินี้ ตำแหน่งของข้อมูลต่างๆจะถูกแทนที่ได้โดยตัวแปร 2 ตัวด้วยกันคือ (x, y) เมื่อ x คือระยะทางในแนวนอนจากจุดกำเนิด (origin) ใด ๆ และ y คือระยะทางในแนวตั้งจากจุดกำเนิดเดียวกัน สิ่งที่สำคัญในระบบกราฟิก 2 มิติ คือ จุด (point), เส้นตรง (line) และรูปหลายเหลี่ยม (polygon) เนื่องจากสามารถที่จะนำสิ่งเหล่านี้ไปประกอบเพื่อให้เกิดเป็นภาพขึ้นมาตามความต้องการ

จุด

จุดในระบบกราฟิก 2 มิติจะถูกแสดงโดยพิกัด (coordinate) ของมันเองโดยค่า 2 ค่าด้วยกัน คือ P(x, y) ดังแสดงในภาพที่ 7

ภาพที่ 7

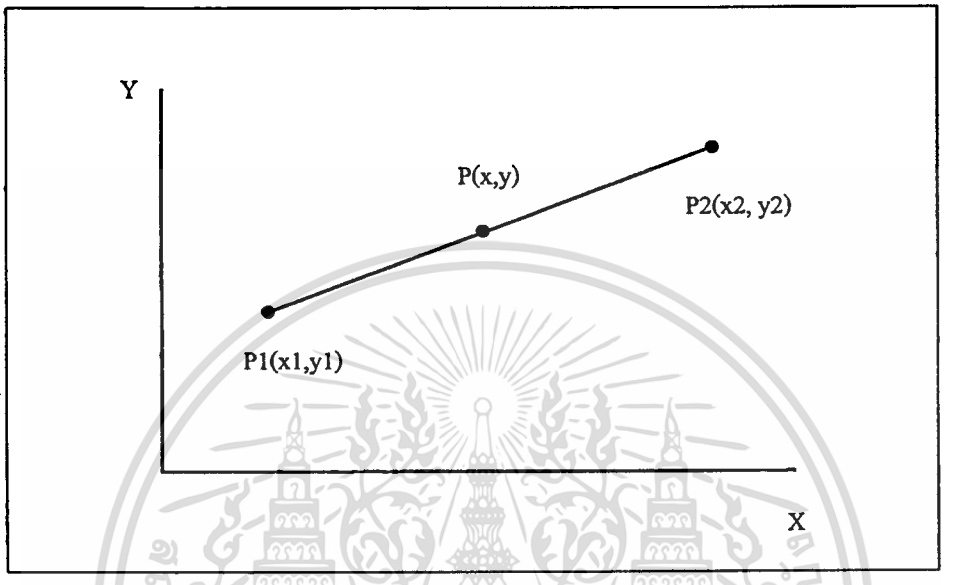


แสดง จุดในระบบกราฟิก 2 มิติ

เส้นตรง

เส้นตรงในระบบกราฟิก 2 มิตินั้น เกิดจาก ระยะทางระหว่างจุด 2 จุด ดังภาพที่ 8

ภาพที่ 8



แสดง เส้นตรงในระบบกราฟิก 2 มิติ

สมการของเส้นตรงสามารถที่จะอธิบายได้ดังนี้ ถ้าจุด P(x, y) เป็นจุดใดๆบนเส้นตรงที่เกิดจากจุด $P_1(x_1, y_1)$ และ $P_2(x_2, y_2)$ ใดๆแล้ว สมการของเส้นตรงคือ

$$y = mx + b \qquad (1)$$

โดยที่

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

$$b = y_1 - mx_1$$

สมการที่ (1) เรียกว่าสมการเส้นตรงแบบ Slope-intercept[5]

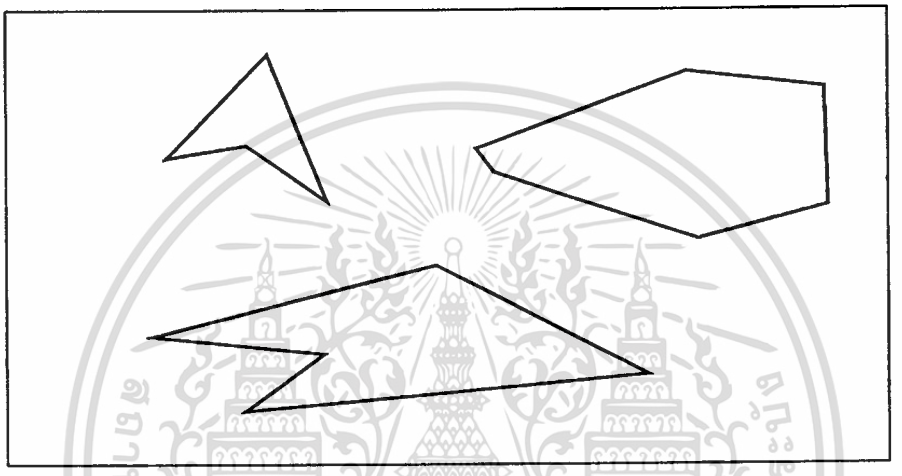
m = ความชัน (Slope)

b = จุดตัดแกน y ของสมการเส้นตรง ณ. ตำแหน่งที่ x มีค่าเป็นศูนย์ (0, b)

รูปหลายเหลี่ยม

รูปหลายเหลี่ยม [5] คือ รูปที่เกิดจากการรวมของจุดและเส้นตรงที่เชื่อมติดกันที่บริเวณปลายทั้งสองข้างดังภาพที่ 9 จะเห็นว่ารูปหลายเหลี่ยมที่เกิดขึ้นนั้นจะมีลักษณะเป็นภาพปิด (close figure) เรียกด้านที่ประกอบกันของรูปหลายเหลี่ยมว่าด้าน (side) หรือขอบ (edges) ของรูป ส่วนจุดปลายสุดของแต่ละด้านของรูปหลายเหลี่ยมจะเรียกว่าจุดรวม (vertices)

ภาพที่ 9



แสดง รูปหลายเหลี่ยม

การเปลี่ยนแปลงตำแหน่ง

ในระบบคอมพิวเตอร์กราฟิกนั้นในบางครั้งมีความจำเป็นที่จะต้องทำการเปลี่ยนแปลงลักษณะของภาพที่ได้เพื่อนำผลที่ได้ให้สามารถนำไปใช้ประโยชน์ในด้านอื่น หรือ เพื่อความสะดวกแก่ผู้ใช้งาน โดยไม่ต้องสร้างภาพขึ้นมาใหม่ เช่น การขยายภาพ การลดขนาดภาพ หรือ การเปลี่ยนทิศทางของภาพเดิม แต่จะนำสมการทางคณิตศาสตร์บางอย่างมาช่วยในการเปลี่ยนแปลงลักษณะของภาพจำลอง[6]

การเปลี่ยนแปลงขนาด

กำหนดให้ $P_1 = [\, x_1 \;\; y_1 \,]$ เป็นจุดใดๆที่มีขนาด 1x2 เมตริกซ์ (matrix) ถ้าคูณเมตริกซ์ P_1 ด้วยเมตริกซ์ T ที่มีขนาด 2x2 จะทำให้ได้เมตริกซ์ใหม่ที่มีขนาด 1x2 ด้วย ซึ่งสามารถใช้แทนตำแหน่งอื่นๆได้

$$[x_2 \;\; y_2] = P_2 = P_1T \qquad (2)$$

จากสมการที่ (2) จะเห็นว่าเมตริกซ์ T สามารถใช้เป็นเมตริกซ์ที่เป็นตัวเชื่อมความสัมพันธ์ของตำแหน่งเดิม P_1 และตำแหน่งใหม่ P_2 ได้ หรือเมตริกซ์ T สามารถใช้เป็นตัวกำหนดขนาดของตำแหน่ง P_2 เมื่อเทียบกับตำแหน่ง P_1 ได้ซึ่งเมตริกซ์ T สามารถเขียนได้ดังนี้

$$T = S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \qquad (3)$$

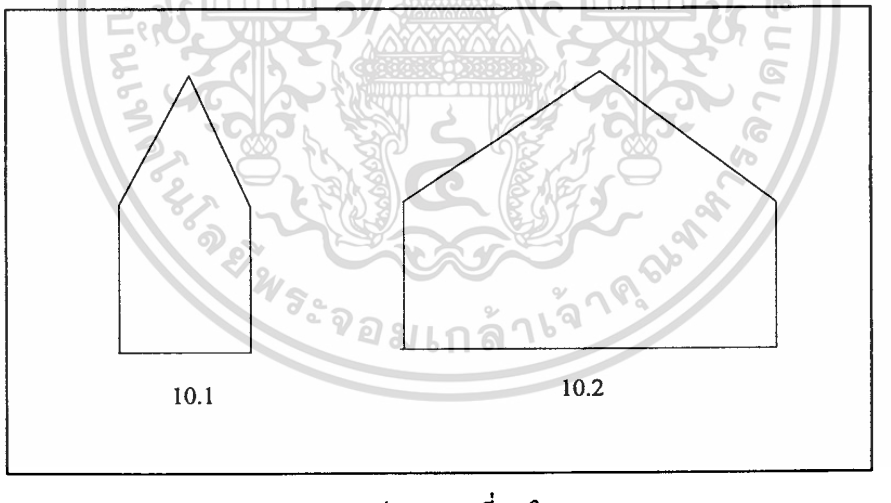
เมื่อ S = เมตริกซ์ของการขยายขนาด[5]

S_x = สัดส่วนการขยาย (Scale factor) ในแนวแกน

S_y = สัดส่วนการขยายในแนวแกน Y

ดังนั้นเมื่อนำมาใช้กับข้อมูลของรูปหลายเหลี่ยม จะทำให้สามารถที่จะลดหรือขยายสัดส่วนของภาพได้ตามต้องการ ดังแสดงให้เห็นในภาพที่ 10.1 และ 10.2

ภาพที่ 10



แสดง การขยายรูปหลายเหลี่ยมในแนวแกน

การเคลื่อนย้ายตำแหน่ง

การเคลื่อนย้ายภาพจากตำแหน่งเดิมไปสู่ตำแหน่งใหม่นั้นเรียกว่า การเคลื่อนย้ายตำแหน่ง (translation) ดังนั้นเป็นการง่ายถ้าต้องการเคลื่อนย้ายตำแหน่งของวัตถุไป (t_x, t_y) หน่วย จากตำแหน่งเดิม (x_1, y_1) ไปสู่ตำแหน่งใหม่ (x_2, y_2) ดังแสดงในภาพ 11

$$\left.\begin{aligned} x_2 &= x_1 + t_x \\ y_2 &= y_1 + t_y \end{aligned}\right\} \qquad (4)$$

แต่เนื่องจากในระบบคอมพิวเตอร์กราฟิกนั้น จะใช้สมการคณิตศาสตร์ที่เป็นระบบเมตริกซ์เป็นส่วนมาก ดังนั้นสมการที่ (4) จึงไม่เหมาะสมที่จะนำมาใช้ในระบบคอมพิวเตอร์กราฟิกมากนัก เพื่อที่จะแก้ปัญหานี้จึงได้มีการนำเอาวิธีการของพิกัดแบบ homogeneous[6] (homogeneous coordinate) มาใช้ซึ่งมีวิธีการดังนี้

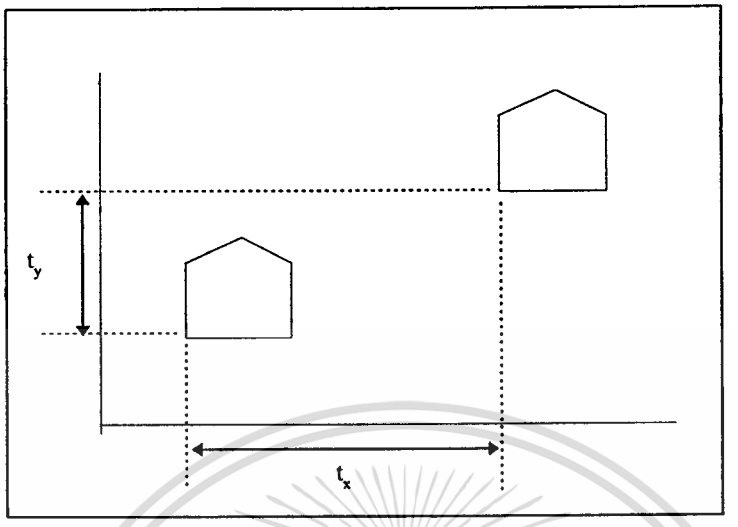
ในระบบ พิกัดแบบ homogeneous นั้นจะใช้เมตริกซ์ขนาด 3x3 แทนเมตริกซ์ขนาด 2x2 โดยเพิ่มสมาชิกที่เรียกว่า พิกัดตัวฝาก(dummy coordinate) คือ w ขึ้นมา จุดต่างๆจะถูกกำหนดโดยตัวเลข 3 ตัวแทนแบบเดิมที่ใช้ตัวเลข 2 ตัว สมาชิกตัวแรกของระบบ นี้จะเป็นผลคูณของ x และ w สมาชิกตัวที่ 2 จะเป็นผลคูณของ y และ w ส่วนสมาชิกตัวที่ 3 คือ w ดังนั้นจุด (x, y) จะถูกแทนที่เป็น (xw, yw, w) โดยกำหนดให้ w มีค่าเท่ากับ 1
ดังนั้นในระบบนี้ เมตริกซ์ของการเปลี่ยนแปลงตำแหน่ง[5] ด้วยขนาด t_x, t_y คือ

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \qquad (5)$$

และเมตริกซ์ของการเปลี่ยนแปลงขนาด[5] คือ

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (6)$$

ภาพที่ 11

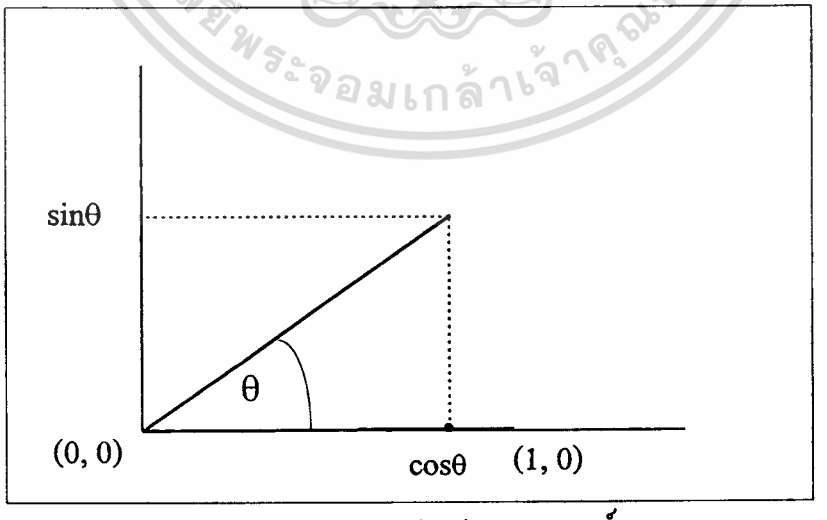


แสดง การเคลื่อนย้ายตำแหน่งของวัตถุ

การเปลี่ยนแปลงตำแหน่งโดยการหมุน

ในการหาเมตริกซ์ของการเปลี่ยนแปลงตำแหน่งโดยการหมุน (Rotation) นั้นสามารถกระทำได้โดย พิจารณาจุดที่ตำแหน่ง (1, 0) ถ้าหมุนตำแหน่งของจุดนี้ในทิศทวนเข็มนาฬิกาด้วยมุม θ ดังภาพ 12 จะพบว่าที่ตำแหน่งใหม่นี้จะมีค่า (cosθ, sinθ)

ภาพที่ 12



แสดง การหมุนของจุด (1, 0) รอบจุดศูนย์กลาง

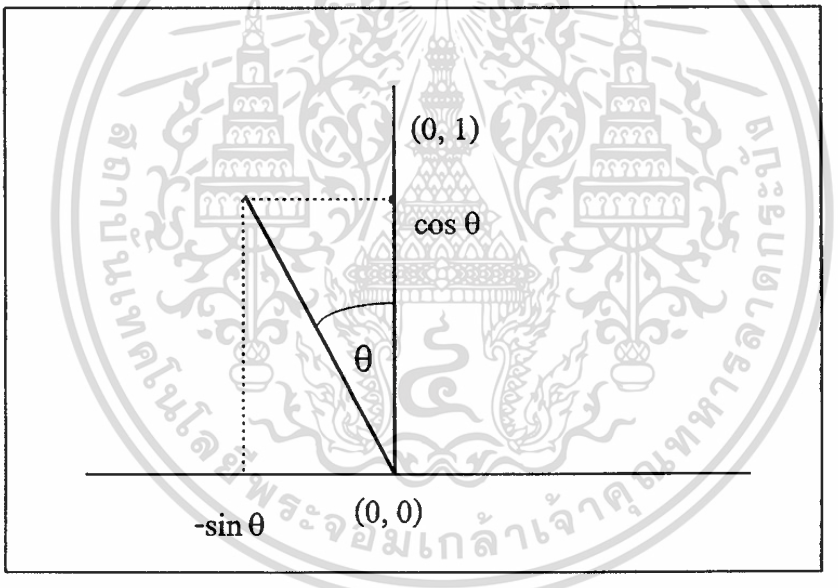
ดังนั้น

$$\begin{bmatrix}\cos\theta & \sin\theta\end{bmatrix} = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}a & b\\ c & d\end{bmatrix} = \begin{bmatrix}a & b\end{bmatrix} \qquad (7)$$

ในกรณีเดียวกันถ้าหมุนจุด (0, 1) ในลักษณะเดียวกันตำแหน่งใหม่ที่ได้คือ (-sinθ, cosθ) ดังภาพ 13 ดังนั้น

$$\begin{bmatrix}-\sin\theta & \cos\theta\end{bmatrix} = \begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}a & b\\ c & d\end{bmatrix} = \begin{bmatrix}c & d\end{bmatrix} \qquad (8)$$

ภาพที่ 13



แสดง การหมุนจุด (0, 1) รอบจุดศูนย์กลาง

จากสมการที่ (7) และ (8) สามารถที่จะกำหนดเมตริกซ์ของการเปลี่ยนแปลงตำแหน่งโดยการหมุนในทิศทางทวนเข็มนาฬิกาเป็นมุม θ รอบจุดศูนย์กลาง[7] ได้คือ

$$R = \begin{bmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{bmatrix} \qquad (9)$$

และในระบบพิกัดแบบ homogeneous คือ

$$R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (10)$$

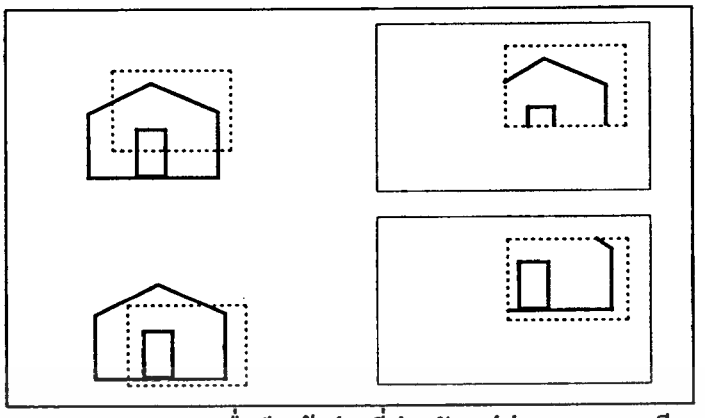
หน้าต่าง , ช่องมองภาพและการตัดขอบภาพ

ภาพต่างๆที่ได้จากการจำลองภาพในระบบคอมพิวเตอร์กราฟิกนั้นบางครั้งจะเป็นภาพที่มีขนาดใหญ่และมีรายละเอียดในภาพมาก ซึ่งทำให้การมองภาพจำลองบนจอภาพของคอมพิวเตอร์ไม่สามารถที่จะบอกรายละเอียดของส่วนต่างๆของภาพได้ทั้งหมดภายในครั้งเดียว ดังนั้นจึงต้องมีการนำเอาวิธีการบางอย่างเข้ามาช่วยในการในการมองของภาพจำลองที่ได้ เพื่อให้สามารถแสดงรายละเอียดบางส่วนของภาพจำลองได้ วิธีการที่นำมาช่วยในการมองภาพจำลองนั้นเรียกว่า หน้าต่าง (Window)[5]

หน้าต่างนั้นสามารถทำให้การแสดงภาพในส่วนที่ต้องการได้ไม่ว่าจะเป็นการแสดงภาพจำลองทั้งภาพ หรือ เฉพาะบางส่วนของภาพที่ต้องการแสดงรายละเอียดเป็นพิเศษ ขนาดของหน้าต่างสามารถที่จะกำหนดขนาดได้ตามต้องการ ไม่ว่าจะเป็นหน้าต่างที่สามารถแสดงภาพทั้งหมดของภาพจำลองหรือการแสดงเพียงบางส่วนของภาพจำลองก็ได้ การนำส่วนของภาพจำลองที่เลือกไว้มาแสดงผลบนจอภาพนั้นยังสามารถที่จะกำหนดได้ว่าสามารถที่จะแสดงที่ส่วนใดของจอภาพและมีขนาดเท่าใดของจอภาพ ช่องการแสดงผลบนจอภาพนั้น เรียกว่า ช่องมองภาพ (viewport)[6]

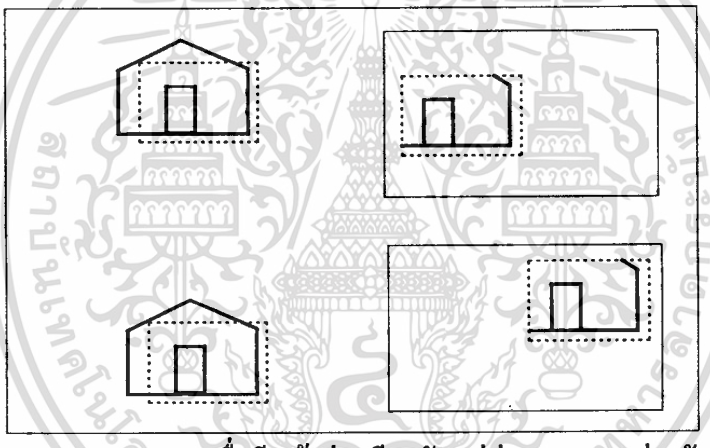
ส่วนของภาพจำลองที่เลือกโดยขนาดของหน้าต่างนั้นจะถูกนำมาแสดงในช่องมองภาพ และถึงแม้ว่าขนาดของภาพที่ถูกกำหนดโดยหน้าต่างและช่องมองภาพจะมีขนาดแตกต่างกันหรือตำแหน่งต่างกัน เมื่อทำการแสดงผลบนจอภาพ ส่วนของภาพจำลองที่ถูกกำหนดโดยหน้าต่างจะถูกนำมาคำนวณใหม่เพื่อเปลี่ยนแปลงขนาดและตำแหน่งให้เหมาะสมกับตำแหน่งและขนาดของช่องมองภาพโดยสมการทางคณิตศาสตร์ เช่น การขยายภาพหรือ การหมุนภาพ จากผลของการนำเรื่องหน้าต่างและช่องมองภาพมาใช้งานทำให้สามารถที่จะนำบางส่วนของภาพมาย่อหรือขยายสัดส่วนได้ตามต้องการ การแสดงผลของช่องมองภาพ และหน้าต่างแสดงไว้ในภาพที่ 14, ภาพที่ 15 และภาพที่ 16

ภาพที่ 14



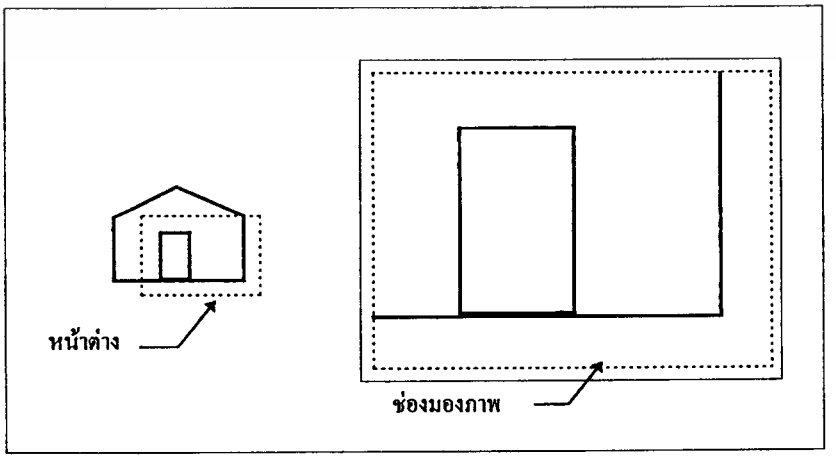
แสดง การแสดงผลเมื่อมีหน้าต่างที่ต่างกันแต่ช่องมองภาพเดียว

ภาพที่ 15



แสดง การแสดงผลเมื่อมีหน้าต่างเดียวกันแต่ช่องมองภาพต่างกัน

ภาพที่ 16

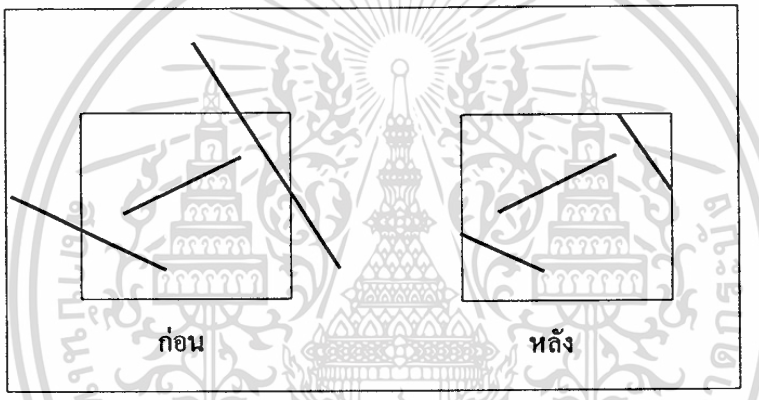


แสดง การขยายส่วนของภาพจำลองโดยช่องมองภาพ

การตัดส่วนของภาพ

การตัดส่วนของภาพ[6] (Clipping) เป็นการกำจัดส่วนของภาพจำลองที่ไม่ต้องการหรือส่วนของภาพจำลองที่อยู่ภายนอกหน้าต่างที่กำหนด ดังนั้นในการตัดส่วนของภาพจำลองนั้น จะต้องพิจารณาว่าส่วนใดส่วนหนึ่งของภาพจำลองมีความสัมพันธ์อย่างไรกับขนาดและตำแหน่งของหน้าต่าง เช่น ภาพจำลองอยู่ภายในหน้าต่างทั้งหมด ภาพจำลองอยู่ภายนอกหน้าต่างทั้งหมด หรือบางส่วนของภาพจำลองอยู่ภายนอกหน้าต่าง ภาพที่ 17 แสดงการตัดส่วนของภาพจำลอง ในการตัดส่วนของภาพมีวิธีการทำได้หลายวิธีด้วยกัน เช่น วิธีของ Cohen - Sutherland และวิธีของ Sutherland - Hodgman

ภาพที่ 17

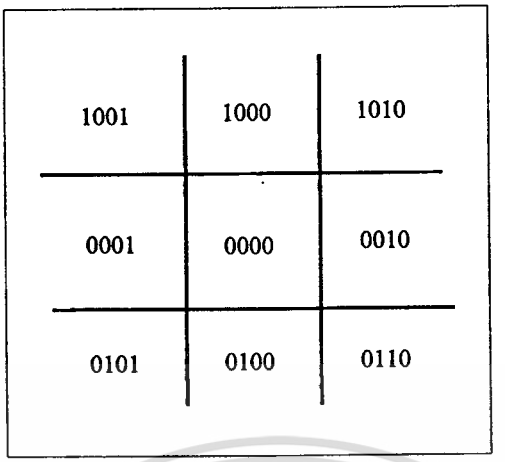


แสดง การตัดส่วนของภาพ

อัลกอริทึมของ Cohen - sutherland

วิธีการตัดส่วนของเส้นตรงที่นิยมใช้กันมากคือ วิธีของ Cohen - Sutherland[5] ซึ่งวิธีการตัดส่วนของเส้นตรงวิธีนี้สามารถทำได้รวดเร็วมาก เมื่อเส้นตรงที่ต้องการตัดอยู่เฉพาะด้านใดด้านหนึ่งของบริเวณที่ต้องการตัด วิธีการตัดส่วนของเส้นตรงโดยวิธีนี้อาศัย วิธีการของตัวดำเนินการแบบบิตหรือ bit operator หรือ outcode มาใช้ โดยกำหนดจุดจบ (endpoint) ของเส้นตรงมีค่า 4 ค่าด้วยกัน โดยแทนที่ด้วยเลขฐาน 2 บิตที่มีลำดับสูงสุดจะถูกกำหนดให้มีค่าเป็น 1 เมื่อจุดนั้นอยู่เหนือหน้าต่างบิตลำดับถัดมาจะถูกกำหนดให้มีค่าเป็น 1 เมื่อจุดนั้นอยู่ต่ำกว่าหน้าต่าง บิตลำดับที่ 3 และ 4 จะถูกกำหนดให้มีค่าเป็น 1 เมื่อจุดนั้นอยู่ด้านขวาและด้านซ้ายของหน้าต่างตามลำดับดังภาพที่ 18

ภาพที่ 18

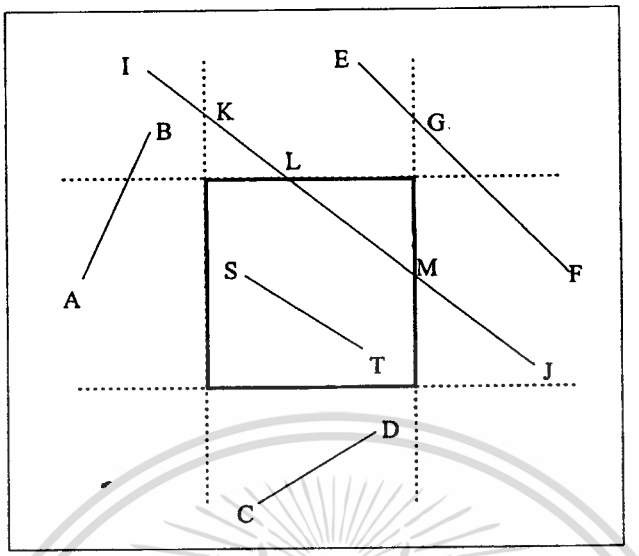


แสดงค่าของตำแหน่งของจุดต่าง ๆ รอบหน้าต่าง

จากภาพที่ 18 เมื่อเส้นตรงอยู่ภายในหน้าต่างทั้งเส้น จุดปลายทั้งสองของเส้นตรงจะมีค่า outcode เป็น "0000" เช่น เส้นตรง ST ในภาพที่ 19 หรือในกรณีที่เส้นตรงเส้นนั้นอยู่ภายนอกหน้าต่างด้านใดด้านหนึ่ง ดังนั้นที่จุดปลายทั้งสองของเส้นตรงจะมีค่า outcode เป็น 1 ในตำแหน่งที่กำหนดของหน้าต่างด้านนั้น การตรวจสอบว่าเส้นตรงเส้นใดอยู่ภายนอกหน้าต่างสามารถกระทำได้โดยใช้วิธีการทางตรรก (logic) เข้ามาช่วยวิธีการนั้นคือ การนำเอาผลของการใช้ตัว ดำเนินการ "AND" มาใช้กับตำแหน่งของจุดปลายของเส้นตรงนั้น ถ้าผลของการ "AND" มีค่าไม่เท่ากับศูนย์แสดงว่า เส้นตรงเส้นนั้นอยู่ภายนอกหน้าต่าง เช่น เส้นตรง AB และ CD ในภาพที่ 19

โดยวิธีการนี้ส่วนที่มีความยุ่งยากมากที่สุดในการหา คือ ในกรณีที่เส้นตรงนั้นมีบางส่วนอยู่ภายในหน้าต่าง เช่น เส้นตรง EF และ IJ ในภาพที่ 19 สำหรับในกรณีนี้จุดตัดของเส้นตรงและขอบของหน้าต่างจะถูกนำมาใช้ช่วยในการแยกเส้นตรงเส้นเดิมออกให้เป็นเส้นตรงเส้นใหม่ ส่วนของเส้นตรงเส้นใหม่จะถูกนำมาทดสอบโดยวิธีการเดิมเพื่อตรวจดูว่าเส้นตรงเส้นใหม่อยู่ภายในหน้าต่างหรือไม่ เช่น เส้นตรง EF จะถูกแยกออกเป็นเส้นตรง EG และ GF และพบว่าเส้นตรงทั้งสองอยู่ภายนอกหน้าต่าง แต่เส้นตรง IJ จะถูกแบ่งออกเป็น IK , KL , LM และ MJ ซึ่งจะทำให้ได้ส่วนของเส้นตรง LM อยู่ภายในหน้าต่างที่ต้องการ

ภาพที่ 19



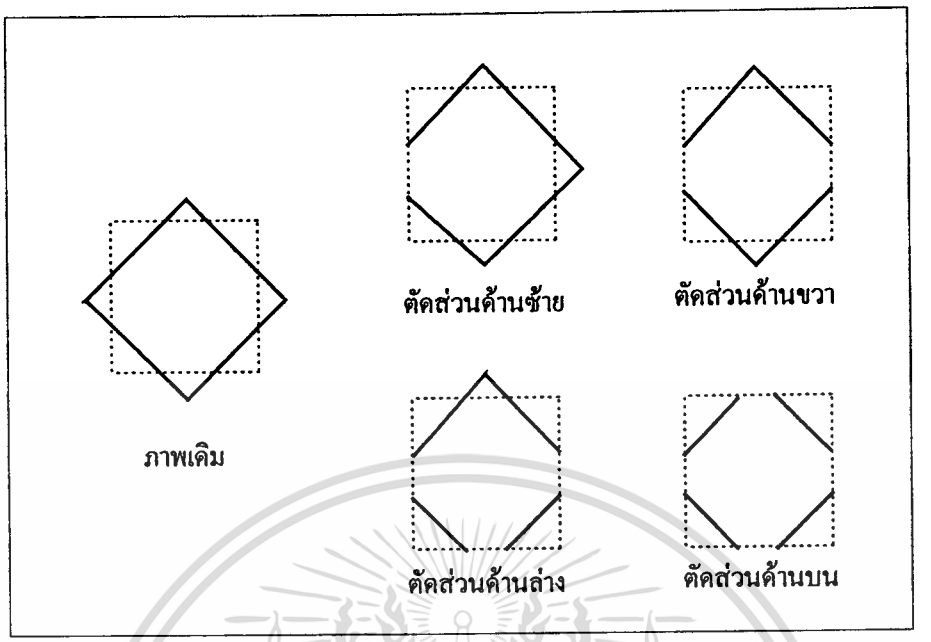
แสดง ลักษณะเส้นตรงต่าง ๆ รอบหน้าต่าง

วิธีของ Sutherland - Hodgman

วิธีการของ Cohen - Sutherland[5] นั้นสามารถนำมาใช้ได้ดีกับการตัดส่วนของเส้นตรง แต่ในบางกรณีภาพต่าง ๆ ที่ได้ไม่ใช่เป็นภาพของเส้นตรงแต่เป็นภาพของรูปหลายเหลี่ยม วิธีการของ Sutherland - Hodgman เป็นวิธีการตัดส่วนของภาพจากเส้นรอบรูปของหน้าต่างทั้ง 4 ด้าน โดยไม่ขึ้นต่อกัน หลักการของวิธีการนี้คือ สามารถที่จะทำให้ตัดภาพจากด้านใดด้านหนึ่งของหน้าต่างโดยไม่ขึ้นกับด้านอื่นๆ

วิธีการตัดส่วนของภาพที่บริเวณขอบเขตของหน้าต่างทำได้โดยพิจารณาที่จุดปลายของเส้นตรงทุกจุดว่าจุดนั้นอยู่ภายในหน้าต่างหรืออยู่ที่ขอบเขตของหน้าต่างหรือไม่ ถ้าอยู่ภายในหน้าต่างที่ต้องการ จุดนั้นจะถูกรักษาไว้หรือคำนวณใหม่ ในกรณีที่จุดอยู่ที่บริเวณจุดตัดของเส้นตรงและขอบเขตของหน้าต่าง ดังนั้นเมื่อได้จุดทั้งหมดที่อยู่ภายในหน้าต่างแล้วจะทำให้ได้ส่วนของภาพที่ในบริเวณของหน้าต่างที่ต้องการ ดังภาพที่ 20

ภาพที่ 20



แสดง การตัดส่วนของภาพที่บริเวณขอบของหน้าต่าง

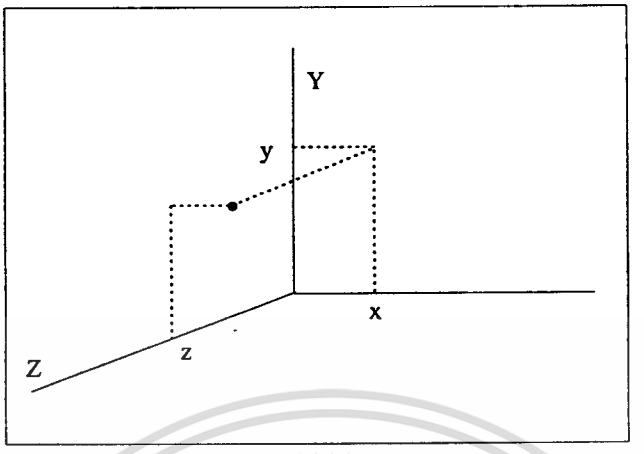
กราฟิก 3 มิติ

ภาพกราฟิกบางชนิดจำเป็นต้องใช้ระบบ 2 มิติเพื่อการแสดงผลเช่น ภาพของ กราฟต่างๆ, แผนที่ หรือแม้กระทั่งภาพที่ถูกสร้างจากจิตรกรต่าง ๆ ก็เป็นภาพกราฟิกในระบบ 2 มิติ แต่ในงานบางอย่างก็จำเป็นที่จะต้องใช้ภาพในระบบ 3 มิติ[8] ในการแสดงผลเพื่อให้เกิดความง่ายในการที่จะทำให้เข้าใจในภาพนั้น เช่น ภาพของโครงสร้างต่าง ๆ ซึ่ง ถ้าเป็นภาพจำลองที่สร้างขึ้นมาในระบบ 3 มิติจะสามารถทำให้ผู้ใช้สามารถมองภาพรวมในจุดมอง (viewpoint) ที่ต่าง ๆ กัน

จุดและระนาบ

สิ่งที่ง่ายที่สุดในการที่จะนำมาใช้ในการอธิบายระบบ 3 มิติคือ จุด เนื่องจากในระบบ 2 มิติได้อธิบายจุดโดยที่บอกตำแหน่งของจุดโดยอาศัย ค่า 2 ค่า แต่ในระบบ 3 มิตินี้จะต้องเพิ่มแกนอีก 1 แกนเพื่อที่จะสามารถนำมาใช้บอกตำแหน่งของจุดในระบบ 3 มิติ ซึ่งจะทำให้ตัวเลขที่ใช้ในการบอกตำแหน่งของจุดมี 3 ค่าด้วยกัน คือ ค่าแรกจะแทนค่าความสูง (hight) ของจุด ส่วนค่าที่ 2 และ 3 จะแทนค่าความกว้าง (width) และความลึก (depth) ตามลำดับ โดยที่ทั้ง 3 แกนที่ใช้ในการอธิบายตำแหน่งในระบบ 3 มิติ จะมีทิศทางที่ตั้งฉากซึ่งกันและกัน ในระบบ 2 มิตินั้นใช้ค่าในแนวแกน และ แกน Y แทนค่าตำแหน่งในด้านความกว้างและความยาวส่วนในระบบ 3 มิตินั้นจะใช้แนวแกน Z แทนค่าของตำแหน่งในด้านความลึก ส่วนในแนวแกน และแกน Y นั้นเหมือนกับในระบบ 2 มิติทุกประการ ดังนั้นการกำหนดตำแหน่งของจุดในระบบ 3 มิติ จะสามารถกำหนดได้โดยตัวแปร 3 ตัว คือ P(x, y, z) ดังแสดงในภาพที่ 21

ภาพที่ 21



แสดง ตำแหน่งของจุดในระบบ 3 มิติ

และสมการของเส้นตรงในระบบ 3 มิติ[5] คือ

$$\left.\begin{aligned} \frac{y - y_1}{x - x_1} &= \frac{y_2 - y_1}{x_2 - x_1} \\ \frac{z - z_1}{x - x_1} &= \frac{z_2 - z_1}{x_2 - x_1} \end{aligned}\right\} \qquad (11)$$

ส่วนสมการของระนาบในระบบ 3 มิติ คือ

$$Ax + By + Cz + D = 0 \qquad (12)$$

เมื่อ A , B , C และ D คือค่าคงที่ หรือ

$$x + B_1y + C_1z + D_1 = 0 \qquad (13)$$

โดยที่ $B_1 = \frac{B}{A}, C_1 = \frac{C}{A}$ และ $D_1 = \frac{D}{A}$

และระยะทางระหว่างจุด (x, y, z) ใด ๆ กับระนาบ ถูกกำหนดโดย

$$L = \left|A_2x + B_2y + C_2z + D_2\right| \qquad (14)$$

โดยที่ $A_2 = \frac{A}{d}$, $B_2 = \frac{B}{d}$, $C_2 = \frac{C}{d}$ และ $D_2 = \frac{D}{d}$

เมื่อ $d = \left(A^2 + B^2 + C^2\right)^{\frac{1}{2}}$

การเปลี่ยนตำแหน่งในระบบ 3 มิติ

ในระบบพิกัดแบบ homogeneous เมตริกซ์ขนาด 4x4 ของเมตริกซ์การเปลี่ยนแปลงขนาด[7] คือ

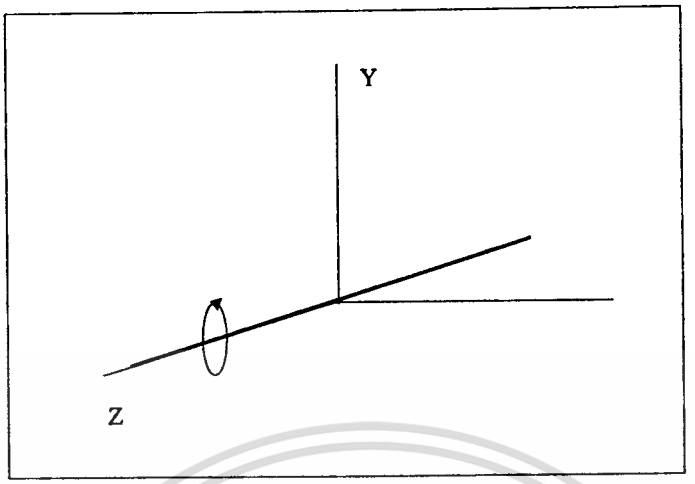
$$S = \begin{vmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad (15)$$

เมื่อ S_x, S_y และ S_z คือ สัดส่วนการขยายในแนวแกน , Y และ Z ตามลำดับ

สำหรับการเคลื่อนย้ายตำแหน่ง[5]นั้น เมตริกซ์ของการเปลี่ยนแปลงตำแหน่งในแกน x ไป t_x หน่วย และเคลื่อนย้ายตำแหน่งไป t_y และ t_z ในแกน y และ z ตามลำดับ คือ

$$T = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{vmatrix} \qquad (16)$$

ภาพที่ 22



แสดง การหมุนรอบแกน Z

ส่วนการหมุนภาพรอบแกนต่างๆ[7] สามารถแสดง เมตริกซ์ของการเปลี่ยนแปลงของการหมุน ได้ดังนี้

สำหรับการหมุนรอบแกน Z ด้วยมุม θ ในทิศทวนเข็มนาฬิกาดังแสดงในภาพที่ 22 คือ

$$R_z = \begin{vmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad (17)$$

และ เมตริกซ์ของการเปลี่ยนแปลงของการหมุนรอบแกน และแกน Y คือ R_x และ R_y ตามลำดับ

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad (18)$$

$$R_y = \begin{vmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad (19)$$

การหมุนวัตถุรอบแกนใด ๆ

การหมุนวัตถุรอบแกนใดๆ[7]สามารถกระทำได้โดยอาศัยเมตริกซ์ของการเปลี่ยนแปลงตำแหน่งต่างๆเช่น เมตริกซ์ของการเคลื่อนย้ายตำแหน่งและเมตริกซ์ของการหมุนรอบแกนใดๆ โดยทั่วไปแล้วเส้นตรงต่างๆ ในระบบ 3 มิติ สามารถถือได้ว่าเป็นแกนของระบบได้ ดังนั้น การหมุนวัตถุรอบแกนใดๆ เป็นมุม θ นั้น สามารถหาเมตริกซ์ของการเปลี่ยนแปลงได้โดยในขั้นแรกนั้นต้องเคลื่อนย้ายแกนใดๆให้จุดกำเนิดของระบบมาอยู่บนแกนที่ต้องการ จากนั้นจะต้องหมุนวัตถุรอบแกน และแกน Y เพื่อให้แกนที่ต้องการอยู่บนแกน Z จากนั้นจึงทำการหมุนวัตถุรอบแกน Z ตามมุมที่ต้องการ ในขั้นตอนสุดท้ายจะต้องทำการเลื่อนแกนที่ต้องการกลับไปสู่ตำแหน่งเดิมในตอนแรกโดยอาศัยส่วนกลับ(inverse) ของเมตริกซ์ของการเปลี่ยนแปลงที่ใช้ในครั้งแรก คือการหมุนรอบแกน Y และแกน ตามลำดับจากนั้นจึงทำการเลื่อนแกนของวัตถุกลับสู่ตำแหน่งเดิม ภาพที่ 23 แสดงขั้นตอนการหมุนวัตถุรอบแกนใด ๆ และเมตริกซ์ของการหมุนวัตถุรอบแกนใด ๆ เป็นมุม θ คือ

$$R_\theta = TR_xR_yR_zR_y^{-1}R_x^{-1}T^{-1} \qquad (20)$$

เมื่อ T = เมตริกซ์ของการเปลี่ยนแปลงตำแหน่ง

R_x = เมตริกซ์ของการเปลี่ยนแปลงการหมุนรอบแกน

R_y = เมตริกซ์ของการเปลี่ยนแปลงการหมุนรอบแกน Y

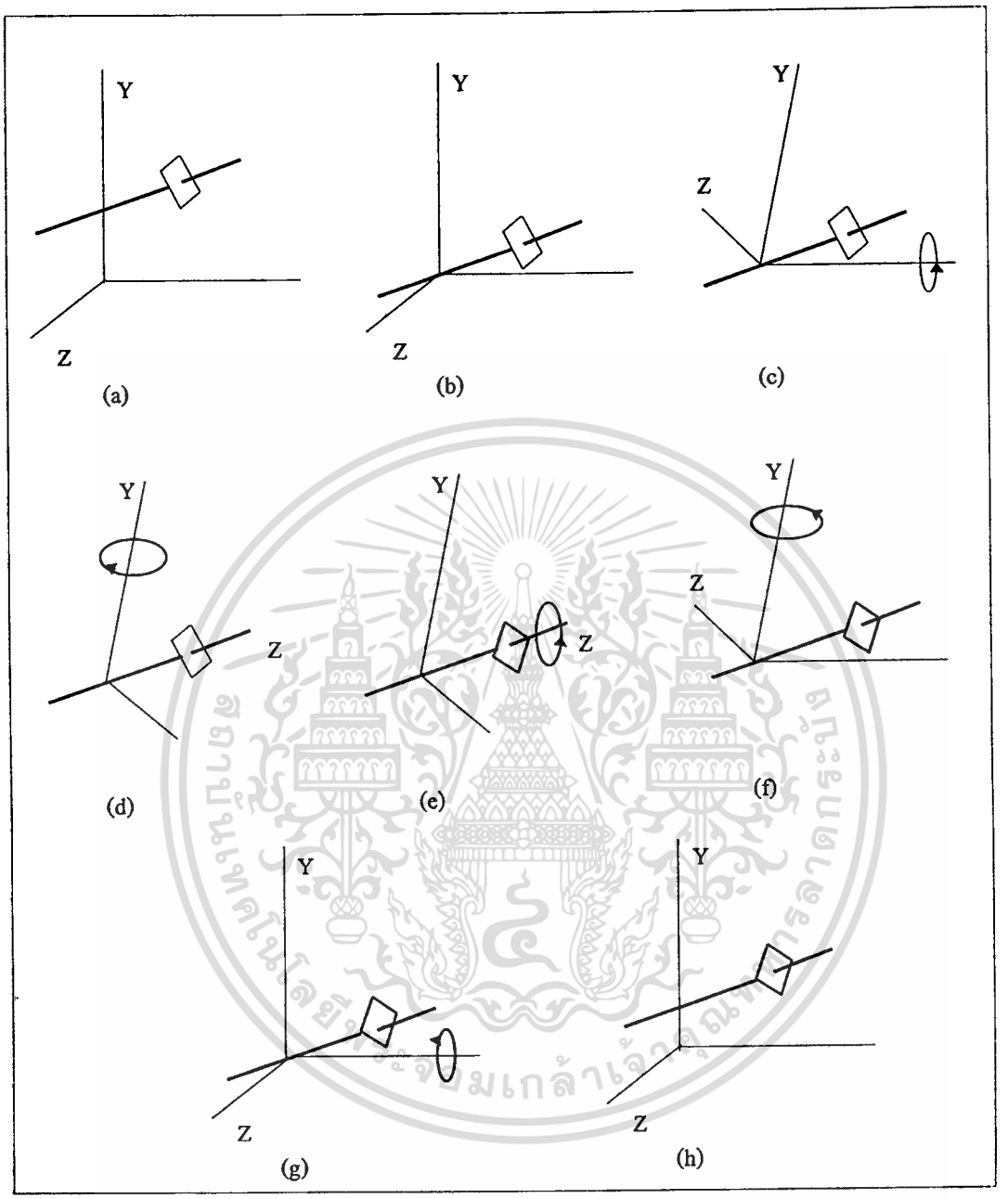
R_z = เมตริกซ์ของการเปลี่ยนแปลงการหมุนรอบแกน Z

T^{-1} = ส่วนกลับของเมตริกซ์ของการเปลี่ยนแปลงตำแหน่ง

R_x^{-1} = ส่วนกลับของเมตริกซ์ของการเปลี่ยนแปลงการหมุนรอบแกน

R_y^{-1} = ส่วนกลับของเมตริกซ์ของการเปลี่ยนแปลงการหมุนรอบแกน Y

ภาพที่ 23



แสดง การหมุนวัตถุรอบแกนใดๆ

การฉายภาพแบบขนานและการฉายภาพแบบเพอสเปคทีฟ

วัตถุต่างๆในระบบ 3 มิตินั้นประกอบด้วยคุณสมบัติของความกว้าง, ความยาวและความลึก ซึ่งต่างจากคุณสมบัติของวัตถุในระบบ 2 มิติซึ่งมีเพียงความกว้างและความยาว อุปกรณ์การแสดงผลของภาพต่างๆนั้นจึงถือว่าเป็นอุปกรณ์ในระบบ 2 มิติเพราะสามารถที่จะกำหนดค่าได้เพียงความกว้างและความยาวเท่านั้น ดังนั้นการที่จะนำภาพในระบบ 3 มิติมาแสดงผลบนอุปกรณ์แสดงผลหรือจอภาพ จึงต้องใช้วิธีการบางอย่างเข้ามาช่วยในการแสดงผลเพื่อให้สามารถแสดงผลของความลึกของวัตถุวิธีการนี้เรียกว่า การฉายภาพ(Projection)

การฉายภาพของวัตถุในระบบ 3 มิติลงบนจอภาพสามารถที่จะแยกออกได้เป็น 2 วิธีด้วยกัน คือ

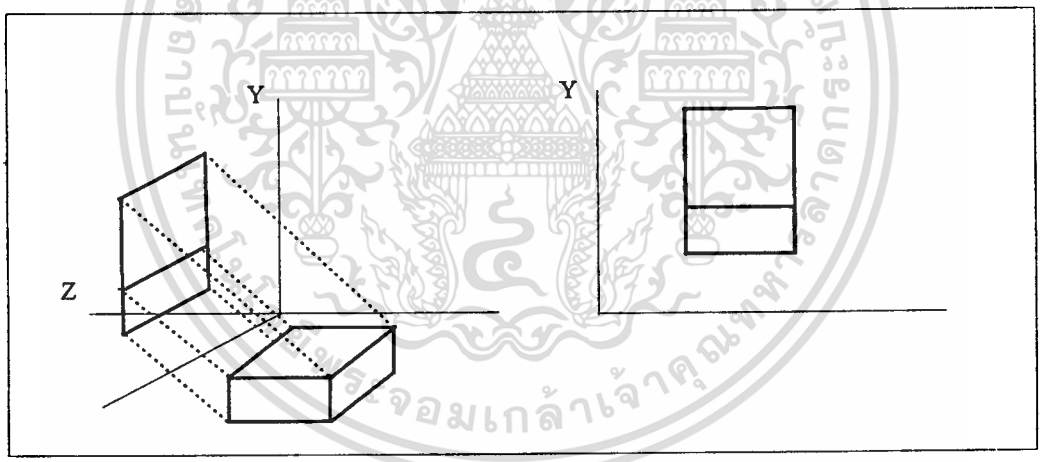
1. การฉายภาพแบบขนาน (Parallel Projection)

2. การฉายภาพแบบเพอสเปคตีป (Perspective Projection)

การฉายภาพแบบขนาน

วิธีการฉายภาพแบบขนาน[5] สามารถกระทำได้โดยการลากเส้นสมมุติ จากจุดยอดของวัตถุทุกๆจุดมาตัดกับระนาบของการมองหรือระนาบของจอภาพ จุดตัดของเส้นขนานจากจุดยอดของวัตถุและระนาบของจอภาพคือ จุดของการฉายภาพของจุดยอดของวัตถุที่ระนาบของจอภาพ ดังนั้นถ้าลากเส้นเชื่อมจุดของการฉายภาพเหล่านี้ จะได้ภาพของการฉายภาพแบบขนานของวัตถุ ดังแสดงในภาพที่ 24

ภาพที่ 24



แสดง การฉายภาพแบบขนาน

จากวิธีการฉายภาพแบบขนานนี้จะเห็นว่าความลึกของวัตถุในแนวแกน Z จะไม่มีผลกับภาพที่ได้ ในกรณีที่ระนาบของการมองอยู่ในระนาบ Y เนื่องจากเส้นของการฉายภาพจะขนานกับแกน Z ดังนั้น ถ้าเคลื่อนที่ภาพในแนวของเส้นของการฉายภาพ ค่าที่มีการเปลี่ยนแปลง คือ ค่าของวัตถุในแนวแกน Z เท่านั้นแต่ค่าของวัตถุในแนวแกน และแกน Y ยังคงมีค่าเท่าเดิม ซึ่งทำให้จุดตัดของการฉายภาพที่ระนาบของการมองมีค่า และ Y เท่ากับค่าของ และ Y ของจุดยอดของวัตถุ ดังนั้นภาพที่ถูกฉายที่ระนาบของการมองจะถูกสร้างจากค่าของ และ Y เท่านั้น โดยที่ค่าในแนวแกน Z จะไม่มีผลใด ๆ กับภาพ

ในการฉายภาพแบบขนานนั้นจะต้องกำหนดทิศทางของการฉายภาพ เช่น [x_p, y_p, z_p] และการฉายภาพจะเกิดที่ระนาบ Y ถ้าจุด (x_1, y_1, z_1) เป็นจุดใด ๆ บนวัตถุ สามารถจะทราบตำแหน่งที่จะถูกฉาย (x_2, y_2) ได้โดย

สมการเส้นตรงในรูปสมการพาราเมตริกซ์ที่มีทิศทางของการฉายภาพและผ่านจุด (x, y, z) ใดๆ คือ

$$\left.\begin{aligned} x &= x_1 + x_p u \\ y &= y_1 + y_p u \\ z &= z_1 + z_p u \end{aligned}\right\} \qquad (21)$$

ที่จุดตัดของทิศทางของการฉายภาพ และระนาบ Y คือ $z = 0$ ดังนั้น

$$u = -\frac{z_1}{z_p} \qquad (22)$$

ดังนั้น

$$\left.\begin{aligned} x_2 &= x_1 - z_1 \frac{x_p}{z_p} \\ y_2 &= y_1 - z_1 \frac{y_p}{z_p} \end{aligned}\right\} \qquad (23)$$

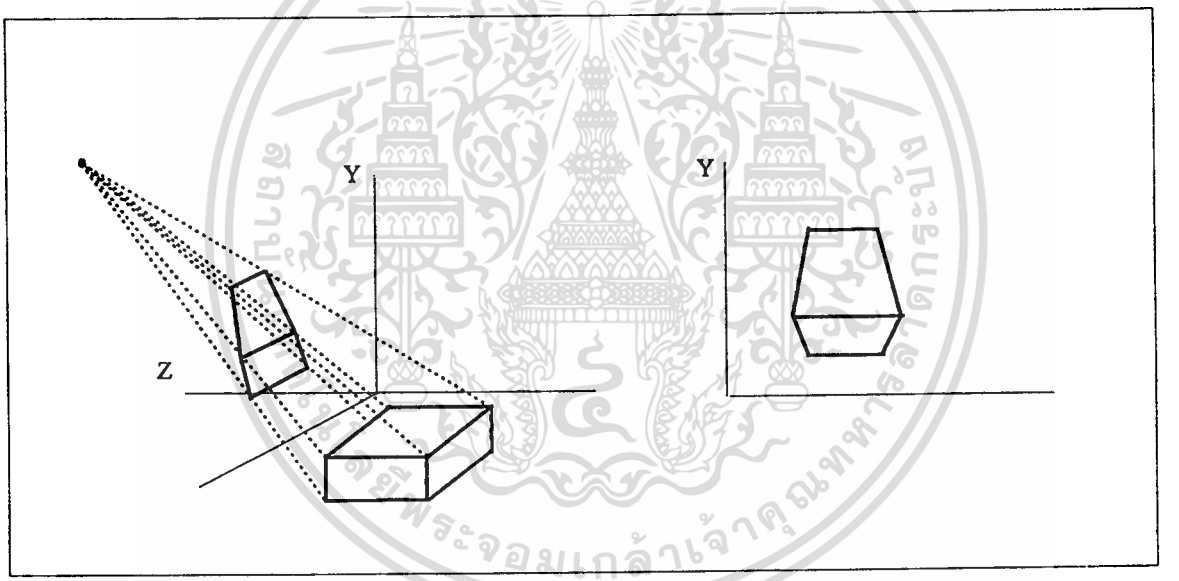
สามารถเขียนในรูปของเมตริกซ์ในระบบ พิกัดแบบ homogeneous ได้คือ

$$[x_2 \quad y_2 \quad z_2 \quad 1] = [x_1 \quad y_1 \quad z_1 \quad 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{x_p}{z_p} & -\frac{y_p}{z_p} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (24)$$

การฉายภาพแบบเพอสเปคตีป

การฉายภาพวัตถุแบบเพอสเปคตีป[7] นี้ภาพของวัตถุที่ได้จะมีขนาดขึ้นอยู่กับตำแหน่งของผู้สังเกต ถ้าผู้สังเกตอยู่ไกลภาพที่ได้จะมีขนาดเล็กและภาพของวัตถุจะมีขนาดใหญ่ขึ้นเมื่อผู้สังเกตอยู่ใกล้กับวัตถุ ดังนั้นการฉายภาพแบบนี้จะสามารถทำให้ภาพที่ได้มีขนาดต่างๆ กันขึ้นอยู่กับระยะห่างของผู้สังเกตและวัตถุ ในการฉายภาพแบบเพอสเปคตีปนี้เส้นของการฉายภาพ (lines of projection) จะไม่ใช่เส้นที่ขนานกัน แต่จะรวมกันที่จุดๆหนึ่ง เรียกว่า จุดศูนย์กลางของการฉายภาพ (center of projection) จุดตัดของเส้นของการฉายภาพและระนาบของการมองจะเป็นสิ่งที่กำหนดลักษณะของวัตถุ ลักษณะของภาพที่เกิดขึ้นบนระนาบของการมองจะเหมือนกับภาพที่ผู้สังเกตมองวัตถุที่ตำแหน่งจุดศูนย์กลางของการฉายภาพหรือทิศทางของเส้นของการฉายภาพจะมีลักษณะเหมือนกับทิศทางของแสงที่มาจากวัตถุมาสู่ดวงตา ภาพที่ 25 แสดงวิธีการฉายภาพแบบเพอสเปคตีป

ภาพที่ 25



แสดง การฉายภาพแบบเพอสเปคตีป

ถ้ากำหนดจุดศูนย์กลางของการฉายภาพอยู่ที่ตำหน่ง (x_c, y_c, z_c) และจุดบนวัตถุอยู่ที่ตำแหน่ง (x_1, y_1, z_1) ดังนั้นทิศทางของการฉายภาพ คือ เส้นตรง

$$\left.\begin{aligned} x &= x_c + (x_1 - x_c)u \\ y &= y_c + (y_1 - y_c)u \\ z &= z_c + (z_1 - z_c)u \end{aligned}\right\} \qquad (25)$$

ตำแหน่งของจุด (x_2, y_2) ที่ถูกฉายบนระนาบของการมอง Y เมื่อ $z = 0$ คือ

$$\left.\begin{aligned} x_2 &= \frac{x_c z_1 - x_1 z_c}{z_1 - z_c} \\ y_2 &= \frac{y_c z_1 - y_1 z_c}{z_1 - z_c} \end{aligned}\right\} \qquad (26)$$

สามารถเขียนการฉายภาพแบบเพอสเปคตีปในรูปของเมตริกซ์ของการเปลี่ยนแปลงในระบบของ homogenous coordinate คือ

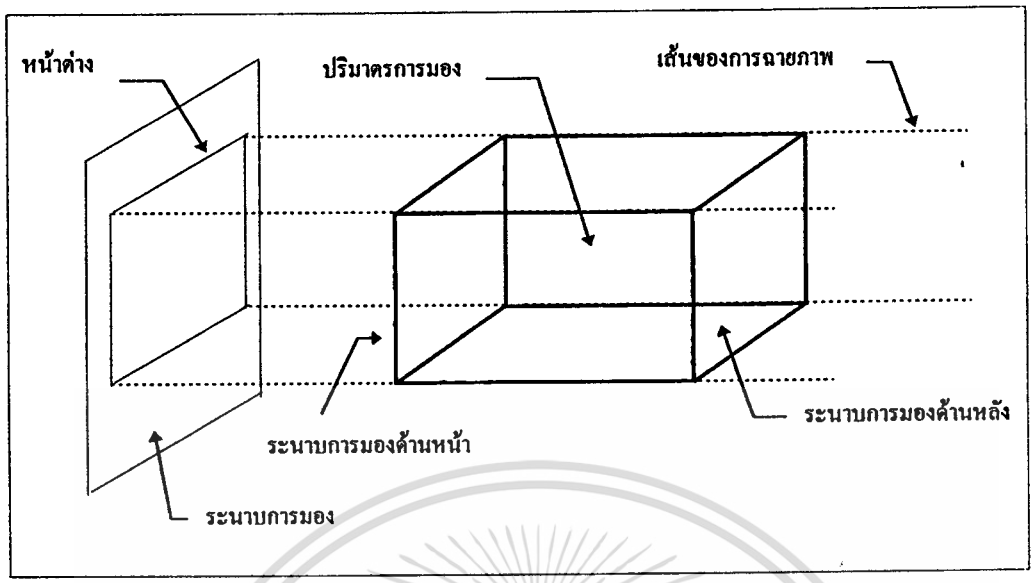
$$P = \begin{vmatrix} -z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & -1 & 1 \\ 0 & 0 & 0 & -z_c \end{vmatrix} \qquad (27)$$

การตัดส่วนของภาพในระบบ 3 มิติ

ในระบบ 2 มิติหน้าต่างถูกใช้เป็นขอบเขตของการมองภาพ เช่นเดียวกันในระบบ 3 มิติหลักการของหน้าต่างยังถูกนำมาใช้กำหนดปริมาตรของการมอง (view volume) วัตถุที่อยู่ภายนอกบริเวณปริมาตรของการมองจะถูกตัดออก[9] ดังนั้นส่วนต่างๆของวัตถุที่อยู่ภายในปริมาตรของการมองเท่านั้นที่จะถูกมองเห็น เนื่องจากเป็นระบบ 3 มิติปริมาตรของการมองจึงถูกกำหนดโดยระนาบของการมอง 6 ระนาบด้วยกัน คือ ด้านหน้า, ด้านหลังและด้านข้าง 4 ด้าน หลักการของการตัดส่วนของภาพในระบบ 3 มิติ ยังคงเหมือนกับการตัดส่วนของภาพในระบบ 2 มิติคือ จะต้องตรวจสอบว่าตำแหน่งของวัตถุอยู่ในปริมาตรของการมองหรือไม่ แต่ในระบบ 3 มิติจะต้องตรวจสอบว่าจุดต่างๆอยู่ภายในระนาบหรือไม่ แทนการตรวจสอบว่าวัตถุอยู่บนเส้นตรงเหมือนในระบบ 2 มิติ

ปริมาตรของการมองจะถูกแยกออกเป็น 2 แบบด้วยกันตามชนิดของการฉายภาพสำหรับการฉายภาพแบบขนาน ระนาบที่อยู่ในทิศเดียวกับทิศทางของการฉายภาพ จะกำหนดท่อรูปเหลี่ยมที่มีขนาดเท่ากับขนาดของหน้าต่าง ดังภาพที่ 26 และระนาบของการตัดภาพของด้านหน้าและด้านหลังจะถูกกำหนดขึ้นมาซึ่งจะทำให้ท่อรูปเหลี่ยมเปลี่ยนให้มีลักษณะเหมือนกล่องรูปลูกบาศก์ วัตถุต่างๆที่อยู่ภายในบริเวณกล่องรูปลูกบาศก์เท่านั้นที่จะสามารถมองเห็นได้

ภาพที่ 26

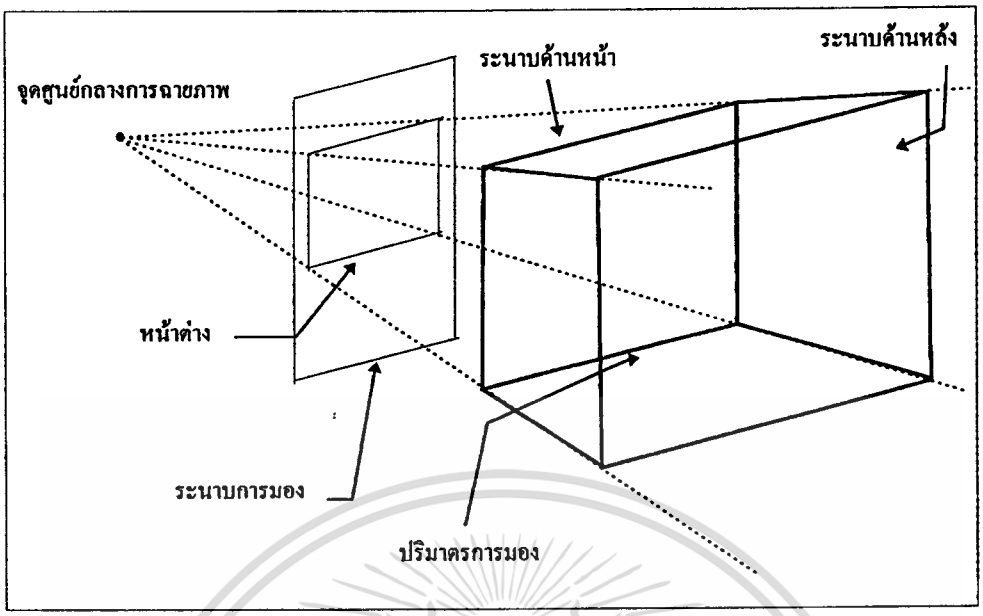


แสดง ปริมาตรการมองสำหรับการฉายภาพแบบขนาน

สำหรับการฉายภาพแบบเพอสเปคทีฟนั้น ขอบเขตของหน้าต่างที่ระนาบการมองและจุดศูนย์กลางของการฉายภาพ จะทำให้เกิดลักษณะที่เป็นพิรามิด (Pyramid) ขึ้น ดังภาพที่ 27 รูปพิรามิดจะถูกแบ่งโดยระนาบของการตัดด้านหน้าและระนาบของการตัดด้านหลัง จะทำให้เกิดปริมาตรของการมอง[10]

จากสมการที่ (12) ซึ่งเป็นสมการของระนาบใดๆ ทำให้สามารถนำไปประยุกต์ในการทดสอบว่าจุดต่าง ๆ อยู่บนระนาบหรือไม่หรืออยู่ด้านใดของระนาบที่กำหนด ซึ่งจะทำให้สามารถนำไปคำนวณหาภาพที่อยู่ในบริเวณของปริมาตรของการมองได้

ภาพที่ 27



แสดง ปริมาตรการมองสำหรับการฉายภาพแบบเพอสเปคตีป

การซ่อนเส้นและพื้นผิว

ในการมองภาพของวัตถุที่มีความเหมือนจริง รายละเอียดของวัตถุจะไม่สามารถมองเห็นได้ทั้งหมดภายในครั้งเดียว เนื่องจากการบดบังซึ่งกันและกัน ส่วนประกอบส่วนใดของวัตถุที่จะสามารถมองเห็นได้และส่วนใดของวัตถุจะถูกบดบังไว้ จะขึ้นอยู่กับตำแหน่งที่มองภาพเช่น จากการมองจากด้านหน้า ส่วนของวัตถุที่สามารถมองเห็นได้คือ บริเวณด้านหน้าและส่วนด้านหลังของวัตถุจะเป็นส่วนที่ถูกบดบังไว้ ในทางตรงกันข้ามถ้ามองวัตถุจากด้านหลังส่วนของวัตถุที่ถูกบดบังไว้คือด้านหน้าของวัตถุ หรือในกรณีที่มองวัตถุจากภายนอกจะทำให้ไม่สามารถที่จะมองเห็นรายละเอียดภายในของวัตถุได้เลย เนื่องจากถูกบดบังด้วยพื้นผิวของวัตถุภายนอก

จากการสร้างแบบจำลองของวัตถุในระบบ 3 มิตินั้น จะต้องสร้างและกำหนดรายละเอียดทุกจุดของภาพทั้งรายละเอียดภายในหรือภายนอก ดังนั้นในการแสดงภาพวัตถุแต่ละครั้งจะทำให้รายละเอียดที่กำหนดไว้แสดงออกมาด้วยซึ่งจะทำให้เป็นการยากที่จะกำหนดว่ารายละเอียดส่วนใดเป็นของบริเวณใดของวัตถุ วิธีการที่ช่วยในการแสดงผลของวัตถุให้สามารถแยกรายละเอียดของพื้นผิวของวัตถุได้เรียกว่า การซ่อนเส้นและพื้นผิว (Hidden lines and Surface)[6] การซ่อนเส้นและพื้นผิวสามารถกระทำได้หลายวิธีด้วยกัน ซึ่งแต่ละวิธีนั้นมีขั้นตอนการทำงานที่ต่างกัน

การใช้ Z BUFFERS

การซ่อนเส้นและพื้นผิวโดยวิธีการใช้ Z Buffers[11] จะอาศัยคุณสมบัติของอุปกรณ์ต่างๆ มาช่วยในการทำงานคือ จอภาพ เนื่องจากจอภาพที่เรียกว่า Raster display นั้นจะอาศัย frame buffer เป็นตัวช่วยในการแสดงผลคือ ค่าของจุดต่างๆที่จะถูกนำมาแสดงผลจะถูกนำมาเก็บไว้ในหน่วยความจำที่ใช้เฉพาะกับจอภาพก่อนแล้วจึงแสดงผลทางจอภาพ Z Buffers หมายถึง ชุดของหน่วยความจำที่ถูกกำหนดขึ้นมาเพื่อใช้ในการเก็บข้อมูลของทุกๆจุดบนจอภาพคล้ายกับ frame buffer หน่วยความจำนี้จะเก็บข้อมูลของตำแหน่งในแกน Z ของพื้นผิวที่จะถูกแสดงผล ภาพหรือพื้นผิวต่างๆ จะถูกเก็บเข้าสู่หน่วยความจำของ frame buffer ตามลำดับ จากนั้นตำแหน่งต่างๆ ของภาพหรือพื้นผิวจะถูกฉายลงบนส่วนการแสดงผล ดังนั้นจึงสามารถที่จะเปรียบเทียบค่าของตำแหน่งในแกน Z ของพื้นผิวกับค่าอ้างอิงที่เก็บใน Z Buffer จะทำให้ทราบว่าพื้นผิวที่ถูกฉายนั้นอยู่ที่ตำแหน่งใดเมื่อเทียบกับพื้นผิวเดิม โดยที่ถ้าพื้นผิวใหม่ที่มีค่า Z มากกว่าค่าที่เก็บใน Z Buffer แสดงว่าพื้นผิวนั้นอยู่ที่บริเวณด้านหน้าของพื้นผิวเดิมและค่าความเข้มของจุดนั้นจะถูกนำไปเก็บใน Z Buffer ในทางตรงกันข้ามถ้าค่า Z ของพื้นผิวใหม่มีค่าน้อยกว่าค่าที่เก็บใน Z Buffer จะหมายถึงว่าพื้นผิวใหม่อยู่ที่บริเวณด้านหลังของพื้นผิวที่เก็บค่าใน Z Buffer ซึ่งพื้นผิวใหม่จะเป็นพื้นผิวที่ถูกขังไว้

จากวิธีการดังที่กล่าวมาของ Z Buffer จะเห็นว่าวิธีการนี้ใช้เวลามาก เนื่องจากการใช้จุดทุกจุดในการคำนวณ และต้องใช้หน่วยความจำเป็นจำนวนมากเพื่อใช้เก็บข้อมูล แต่วิธีนี้เป็นวิธีที่ค่อนข้างง่าย ดังนั้นจึงสามารถนำไปพัฒนาในส่วนของ Hardware เพื่อให้มีความเร็วเพิ่มขึ้นในการประมวลผลและในปัจจุบันนี้ค่าใช้จ่ายในการผลิต Hardware ยังมีราคาไม่สูงมากด้วยอีกประการหนึ่ง

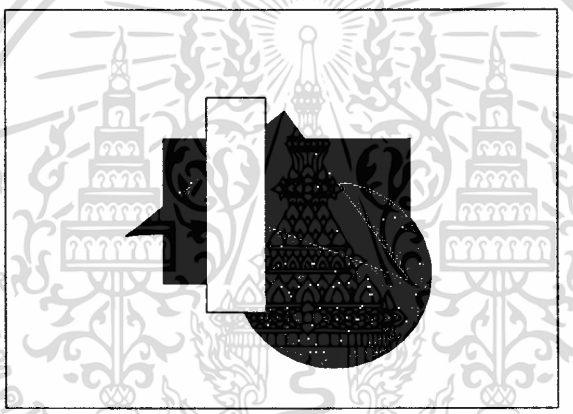
อัลกอริทึม SCAN LINE

วิธีการซ่อนเส้นและพื้นผิวในวิธีของ อัลกอริทึม Scan line [11] นั้นจะใช้วิธีการเปรียบเทียบความลึกของพื้นผิวทั้งหมดในครั้งเดียวที่เส้น Scan line หนึ่ง เมื่อได้พื้นผิวที่อยู่ด้านหน้าสุดแล้วจึงจะทำการหาพื้นผิวที่อยู่หน้าสุดที่เส้น Scan line อื่นๆ ส่วนหลักการทั่วไปยังคงเหมือนกับหลักการของ Z-Buffer จากวิธีการนี้จะเห็นว่าหน่วยความจำบางส่วนที่ใช้ในการเก็บค่าความลึกของพื้นผิวจะน้อยลงเพราะไม่ต้องมีการเก็บข้อมูลทุกครั้ง แต่จะเก็บข้อมูลเฉพาะในส่วนของ Scanline เท่านั้น เมื่อมีการเปลี่ยนแปลง Scanline เป็นเส้นใหม่ ค่าของ Z-Buffer จะถูกกำหนดใหม่ให้เป็นค่าเริ่มต้นแล้วจึงทำการคำนวณหาค่าใหม่ จึงเป็นการใช้หน่วยความจำน้อยกว่า

อัลกอริทึมของ PAINTER

วิธีการซ่อนเส้นและพื้นผิวโดยวิธีอัลกอริทึมของ PAINTER [5] นี้อาศัยลำดับความลึกของพื้นผิวเป็นตัวกำหนดในการแสดงผล พื้นผิวที่มีความลึกที่สุดจะถูกแสดงก่อน ส่วนพื้นผิวที่อยู่ใกล้กับผู้สังเกตมากที่สุดจะถูกแสดงที่หลัง ดังแสดงในภาพที่ 28 จากหลักการที่ผ่านมาพบว่าการซ่อนพื้นผิวโดยวิธีนี้นั้นเหมือนกับการวาดภาพของนักเขียนภาพโดยที่จะเริ่มจากส่วนด้านหลัง (backgrund) ก่อน จากนั้นจึงวาดส่วนด้าหน้าของภาพ (foreground) ซึ่งภาพของส่วนหน้าที่วาดจะเกิดการทับกับพื้นด้านหลัง ซึ่งทำให้ไม่มีการแก้ไขพื้นด้านหลังเลย แต่วิธีการนี้มีข้อเสีย คือการกำหนดพื้นผิวต่างๆต้องเรียงลำดับความลึกด้วย หรือในกรณีที่ไม่มีการเรียงลำดับความลึกนั้นจะต้องทำการเปรียบเทียบค่าความลึกของพื้นผิวทุกครั้งที่มีการกำหนด

ภาพที่ 28



แสดง อัลกอริทึมของ Painter

สรุป

การที่จะแสดงผลของภาพจำลองที่สร้างขึ้นให้ได้ดีนั้นจำเป็นจะต้องนำสมการทางคณิตศาสตร์เข้ามาช่วยในการคำนวณต่างๆ ดังนั้นเครื่องคอมพิวเตอร์ที่มีคุณสมบัติที่ดีขึ้นในด้านการประมวลผลและการแสดงผลจึงเป็นที่ต้องการมากขึ้น วิธีการอีกส่วนที่ทำให้ภาพจำลองมีคุณสมบัติที่ดีขึ้นคือ การนำเรื่องสีและความเข้มของแสงเข้ามาช่วยในการจำลองภาพ ซึ่งจะได้กล่าวถึงรายละเอียดต่อไป

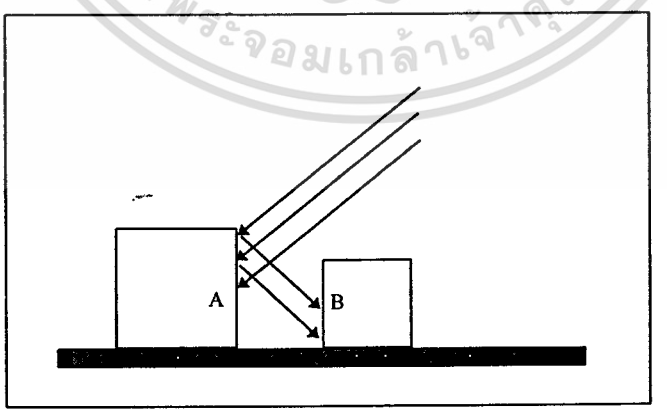
บทที่ 4
แสง สีและลำดับความเข้ม

การสะท้อนแสงของวัตถุ

วิธีการพื้นฐานในการสร้างภาพจำลองของวัตถุที่กล่าวไว้ในบทก่อนๆนั้นไม่สามารถที่จะสร้างภาพจำลองที่มีความเหมือนจริง (realistic images) ได้ แต่เป็นการสร้างภาพจำลองโดยการกำหนดลักษณะกว้าง ๆ เท่านั้น ในการจำลองภาพของวัตถุใดๆให้มีความเหมือนจริงนั้นสิ่งสำคัญที่สุดที่ต้องนำมาพิจารณา คือ แสงบนพื้นผิวของวัตถุ วิธีการนี้เรียกว่า rendering วัตถุต่างๆที่สามารถมองเห็นได้ในธรรมชาตินั้นโดยมากเกิดจากการสะท้อนของแสง (reflection of light) จากพื้นผิวของวัตถุมาสู่ดวงตาของผู้สังเกต ซึ่งจะทำให้ผู้สังเกตสามารถบอกความแตกต่างของวัตถุได้ เช่น สี, รูปร่าง หรือลวดลายต่างๆของพื้นผิวของวัตถุ (texture) ได้ แสงที่มากระทบกับวัตถุนั้นสามารถมาจากหลายทิศทางและด้วยความเข้ม (intensity) ที่แตกต่างกัน ซึ่งเป็นการยากที่จะจำลองภาพของวัตถุให้มีความเหมือนจริงโดยการจำลองแสงต่าง ๆ ที่มากระทบกับวัตถุเพื่อทำให้สามารถมองเห็นวัตถุนั้น

วัตถุต่างๆที่สามารถมองเห็นได้นั้นเป็นผลมาจากแสงที่มาจากต้นกำเนิดแสงที่มีทิศทางต่างกัน เช่น แสงจากดวงอาทิตย์ หรือ แสงที่สะท้อนมาจากพื้นผิวของวัตถุอื่นๆ จากภาพที่ 29 จะเห็นได้ว่าพื้นผิว A จะกระทบกับแสงจากต้นกำเนิดแสงโดยตรงในขณะที่พื้นผิว B จะถูกกระทบโดยแสงที่สะท้อนมาจากพื้นผิว A

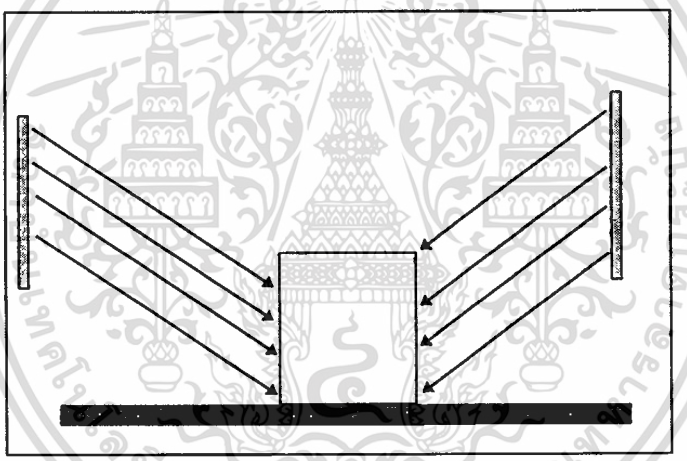
ภาพที่ 29



แสดงการตกกระทบของแสงจากต้นกำเนิดแสง

แสงที่ตกกระทบกับวัตถุ[12] สามารถแยกออกได้เป็น แสงที่มาจากต้นกำเนิดแสงโดยตรง (emitted light) และแสงที่เกิดจากการสะท้อนจากพื้นผิวต่างๆ เรียกว่า แสงโดยรอบ (ambient light ,background light) แสงโดยรอบสามารถที่จะเกิดได้จากต้นกำเนิดแสงหลายชนิดด้วยกันไม่ว่าจะเป็นต้นกำเนิดแสงจุดเดียวหรือว่าต้นกำเนิดแสงจำนวนมากก็ได้ แสงที่ส่องสว่างจากพื้นผิวขนาดเล็ก ๆ เรียกว่า แสงจากต้นกำเนิดแสงแบบจุด (point light source) แสงที่ส่องสว่างจากต้นกำเนิดแสงแบบจุดนี้จะมีคุณสมบัติพิเศษ คือ ทิศทางของแสงจากต้นกำเนิดจะมีทิศทางเดียวและมีค่าความเข้มของแสงเท่ากัน เมื่อสังเกตจากตำแหน่งใดตำแหน่งหนึ่งในระบบ ส่วนแสงที่เกิดมาจากพื้นผิวใด ๆ จะเรียกต้นกำเนิดแสงนั้นว่าต้นกำเนิดแสงแบบกระจาย (distributed light source) ลำแสงต่าง ๆ ที่มาจากต้นกำเนิดแสงแบบกระจายจะกระทบจุดที่กำหนดจากหลายทิศทางและยังมีค่าความเข้มต่างกันด้วย ดังภาพที่ 30

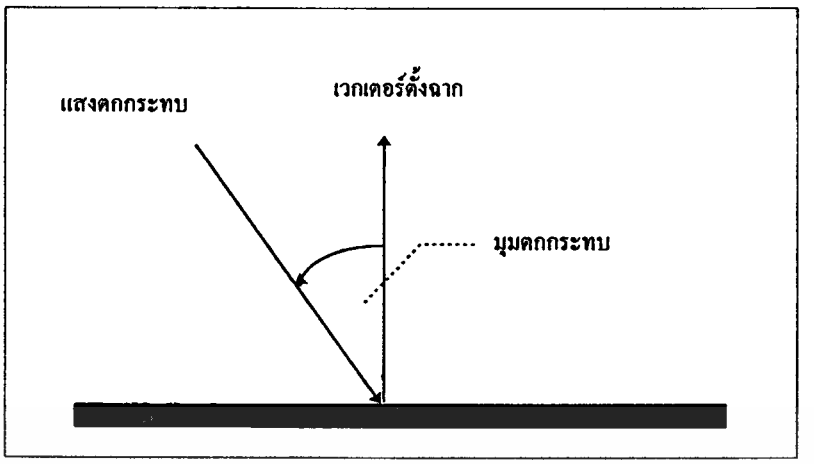
ภาพที่ 30



แสดง การตกกระทบของแสงจากต้นกำเนิดแสงแบบกระจาย

ถึงแม้ว่าดวงอาทิตย์จะเป็นต้นกำเนิดแสงที่มีขนาดใหญ่ แต่เนื่องจากระยะทางระหว่างโลกกับดวงอาทิตย์มีค่ามาก จึงสามารถทำให้อนุมาณได้ว่าดวงอาทิตย์สามารถแสดงลักษณะเป็นต้นกำเนิดแสงแบบจุด ดังนั้นสามารถที่จะกำหนดได้ว่า ลำแสงจากต้นกำเนิดแสงที่มีระยะห่างจากวัตถุมากๆจะมีลักษณะเป็นลำแสงที่ขนานกัน และจะตกกระทบกับทุกตำแหน่งบนพื้นผิวด้วยมุมที่เท่ากัน เรียกมุมนี้ว่า มุมตกกระทบของแสง (angle of incidence) ดังแสดงในภาพที่ 31 เนื่องจากจุดทุกจุดบนพื้นผิวจะมีระยะทางที่เกือบจะเท่ากันจากต้นกำเนิดของแสงที่อยู่ที่ระยะไกล จึงสามารถทำให้กำหนดได้ว่าความเข้มของแสงที่มาตกกระทบจะมีค่าเท่ากัน

ภาพที่ 31



แสดง มุมตกกระทบของแสง

ปริมาณของแสงที่สะท้อนมาจากพื้นผิวของวัตถุ[12]นั้นจะขึ้นอยู่กับลักษณะพื้นผิวและสีของวัตถุ รวมถึงลักษณะของแสงที่ตกกระทบกับพื้นผิวนั้นด้วย การสะท้อนแสงแบบกระจาย (Diffuse Reflection) เกิดขึ้นเมื่อมีแสงมาตกกระทบกับพื้นผิวแล้วพื้นผิวนั้นจะดูดกลืน (absorb) พลังงานของแสงบางส่วนไว้ และปลดปล่อย (emit) แสงออกมาในทุกทิศทาง ซึ่งลักษณะของพื้นผิว เช่น สี และลวดลายต่าง ๆ สามารถมองเห็นได้โดยการสะท้อนแสงแบบกระจาย ส่วนการสะท้อนแสงแบบกรวยหรือการสะท้อนแสงแบบกระจก (Specular Reflection) เป็นการสะท้อนของแสงโดยตรงจากพื้นผิวของวัตถุ ดังภาพที่ 32 พื้นผิวที่มีลักษณะมันวาว (shiny surface) จะสะท้อนแสงที่มาตกกระทบเกือบทั้งหมด ซึ่งจะทำให้เกิดจุดสว่าง (hot spot, highlight) ขึ้น สำหรับพื้นผิวของวัตถุที่มีลักษณะหยาบ (rough) การสะท้อนของแสงแบบกรวยจะเกิดขึ้นน้อยมากหรือไม่เกิดขึ้นเลย

การสะท้อนแสงจากพื้นผิวของวัตถุนั้นจะเป็นผลรวมของการสะท้อนหลายชนิดด้วยกันซึ่งเป็นผลมาจากแสงที่อยู่โดยรอบและแสงที่มาจากต้นกำเนิดแสงโดยตรง

$$I = R_a + R_d + R_s \qquad (28)$$

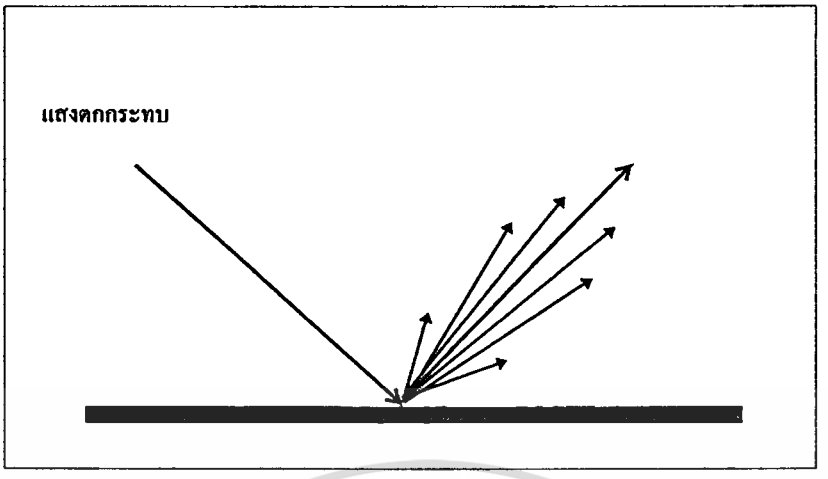
I = ผลรวมของแสงสะท้อนจากผิวของวัตถุ

R_a = การสะท้อนของแสงโดยรอบ

R_d = การสะท้อนแสงแบบกระจาย

R_s = การสะท้อนแสงแบบกรวยหรือการสะท้อนแสงแบบกระจก

ภาพที่ 32



แสดง การสะท้อนแสงแบบกรวย

แสงที่กระจายอยู่โดยรอบจะตกกระทบกับวัตถุในเวลาเดียวกัน ด้วยค่าความเข้มที่เท่ากันในทิศทางต่างๆกัน การกระจายแสงจะเป็นลักษณะที่เหมือนกัน (uniformly) ในทิศทางต่าง ๆ ถ้าความเข้มของแสงที่กระจายอยู่โดยรอบคือ I_a ค่าความเข้มของแสงที่สะท้อนจากแสงสีที่กระจายอยู่โดยรอบ[13] คือ

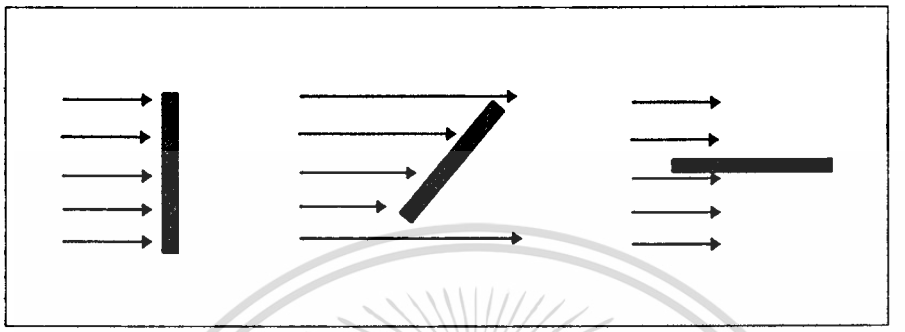
$$R_a = K_d I_a \qquad (29)$$

เมื่อ K_d คือค่าสัมประสิทธิ์การสะท้อนแสงของพื้นผิวของวัตถุ (Coefficient of reflectivity , reflectivity) หรืออัตราส่วนของแสงที่สะท้อนจากพื้นผิวของวัตถุและจำนวนของแสงที่ตกกระทบทั้งหมด

วัตถุที่มีพื้นผิวสีขาวจะสามารถสะท้อนแสงได้เกือบทั้งหมดจากจำนวนของแสงที่มาตกกระทบ ดังนั้นค่าสัมประสิทธิ์การสะท้อนของแสงจะมีค่าใกล้เคียง 1 ส่วนวัตถุที่มีพื้นผิวสีดำจะดูดกลืนแสงที่มาตกกระทบเกือบทั้งหมด ดังนั้นค่าสัมประสิทธิ์การสะท้อนจะมีค่าใกล้เคียง 0 วัตถุทุกชนิดจะสะท้อนบางสีของแสงที่มาตกกระทบได้มากกว่าสีอื่นๆ เช่น วัตถุที่มีค่าสัมประสิทธิ์การสะท้อนของแสงสีแดงสูงกว่าสัมประสิทธิ์การสะท้อนแสงของสีอื่นๆเมื่อมีแสงสีขาว (แสงสีขาวเป็นแสงที่มีปริมาณความเข้มของแสงทุกสีเท่ากัน) มาตกกระทบจะทำให้ส่วนของแสงสีแดงสะท้อนกลับได้มากกว่าสีอื่นๆ จะทำให้เห็นวัตถุนั้นมีสีแดง

เมื่อแสงที่มาจากต้นกำเนิดแสงตกกระทบกับวัตถุโดยตรงในทิศทางที่แน่นอนจะสะท้อนในทุกทิศทาง ดังนั้น เมื่อมุมของแสงที่ตกกระทบมีค่ามากขึ้น พื้นผิวของวัตถุจะรับแสงได้น้อยลง ดังนั้นแสงที่สะท้อนจากพื้นผิวของวัตถุจะมีค่าน้อยลงด้วย ดังแสดงในภาพที่ 33

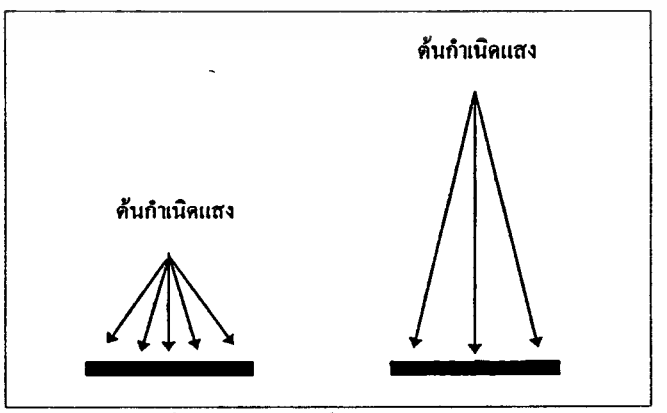
ภาพที่ 33



แสดง แสงที่ตกกระทบกับพื้นผิวที่มุมต่าง ๆ

ความสัมพันธ์ดังกล่าวนี้ถูกอธิบายโดย กฎของ Lambert[13] ที่กล่าวว่าความเข้มของแสงที่สะท้อนแบบกระจายจะเป็นสัดส่วนโดยตรงกับค่า cosine ของมุมของแสงที่ตกกระทบ เมื่อต้นกำเนิดแสงอยู่บริเวณใกล้เคียงกับพื้นผิวของวัตถุ แสงจะตกกระทบที่ผิวของวัตถุด้วยมุมที่แตกต่างกันเนื่องจากแสงที่ตกกระทบไม่ได้มีทิศทางที่ขนานกัน ดังแสดงในภาพที่ 34 มุมของแสงที่ตกกระทบกับพื้นผิวจะมีหลายค่าด้วยกันซึ่งทำให้ยากต่อการคำนวณค่าความเข้มของแสงที่สะท้อนจากวัตถุ เพื่อความสะดวกในการคำนวณจึงกำหนดให้ต้นกำเนิดแสงอยู่ห่างจากวัตถุมากพอที่จะทำให้ต้นกำเนิดแสงนั้นเป็นเหมือนต้นกำเนิดแสงแบบจุด ซึ่งจะทำให้มุมของแสงที่ตกกระทบที่จุดทุกจุดบนพื้นผิวมีค่าเท่ากัน

ภาพที่ 34

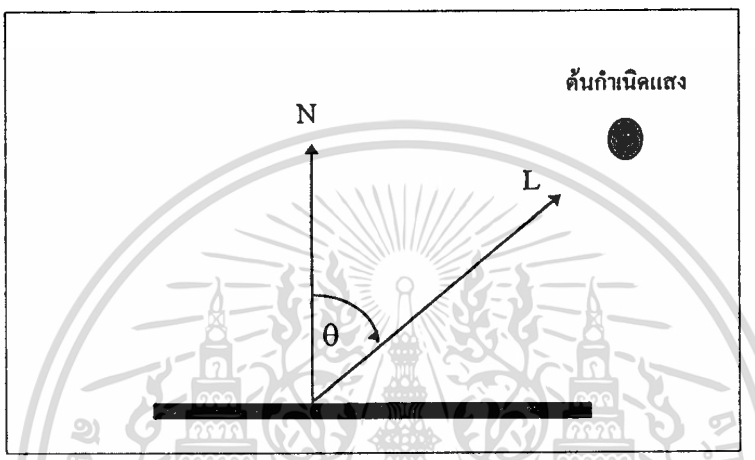


แสดง ทิศทางของแสงที่ตกกระทบ เมื่อต้นกำเนิดแสงอยู่ใกล้เคียงกับวัตถุ

สำหรับต้นกำเนิดแสงแบบจุดใด ๆ นั้นค่า cosine ของมุมตกกระทบ สามารถคำนวณหาได้จาก dot product ของเวกเตอร์หนึ่งหน่วย (unit vector) ที่ตั้งฉากกับพื้นผิวของวัตถุ N และเวกเตอร์หนึ่งหน่วยที่ชี้ไปในทิศทางของต้นกำเนิดแสง L ดังภาพที่ 35

$$\cos\theta = N.L \qquad (30)$$

ภาพที่ 35



แสดง มุมของแสงตกกระทบ

ถ้าต้นกำเนิดแสงมีพลังงาน E_p พลังงานของแสงที่ไปถึงพื้นผิวของวัตถุจะขึ้นกับระยะทางระหว่างต้นกำเนิดแสงและวัตถุด้วย โดยที่ความเข้มของแสงจากต้นกำเนิดแสงที่กระทบวัตถุ (I_p) จะลดลงเป็นสัดส่วนกับกำลังสองของระยะทางระหว่างต้นกำเนิดแสงและวัตถุ (D) [14]

$$I_p = \frac{E_p}{D^2} \qquad (31)$$

และค่าความเข้มของแสงที่เกิดจากการสะท้อนของการกระจาย คือ

$$R_d = \frac{K_d E_p}{D + D_0}(N.L) \qquad (32)$$

เมื่อ D_0 คือ ค่าคงที่ และ K_d คือค่าสัมประสิทธิ์การสะท้อนแสงของพื้นผิวของวัตถุ

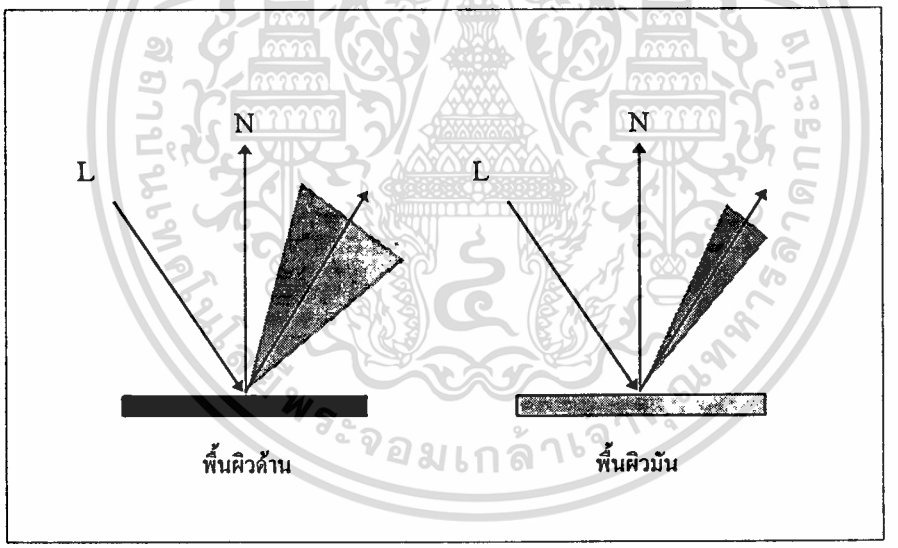
ผลรวมของความเข้มของแสงที่เกิดจากการสะท้อนแบบกระจาย โดยแสงที่อยู่โดยรอบและแสงจากต้นกำเนิดแสงโดยตรง คือ

$$R_{td} = K_d I_a + \frac{K_d E_p}{D + D_0} (N.L) \qquad (33)$$

ขณะที่สีของพื้นผิวของวัตถุถูกกำหนดโดยการสะท้อนแสงแบบกระจาย แต่ความสว่าง (shine) และรูปร่าง (shape) ของวัตถุจะถูกกำหนดโดย จุดสว่างที่เกิดจากการสะท้อนแสงแบบกรวย[13] เนื่องจากมุมของการสะท้อนของแสง (angle of reflection) คือ มุมระหว่างทิศทางของแสงที่สะท้อนจากผิวของวัตถุ และเวกเตอร์ที่ตั้งฉากกับผิวของวัตถุ จะเท่ากับมุมของแสงที่ตกกระทบกับพื้นผิวของวัตถุ ดังภาพที่ 36 ดังนั้นการสะท้อนแสงของพื้นผิววัตถุจะคล้ายกับการสะท้อนแสงของกระจก

ตัวแปรสำคัญที่เป็นตัวกำหนดลักษณะการสะท้อนแสงแบบกรวย คือ ลักษณะการกระจายตัวของพื้นผิวของวัตถุ, ทิศทางของพื้นผิวของวัตถุ, การดูดซับความเข้มของแสงของพื้นผิวของวัตถุ และปริมาณของแสงที่ถูกบังโดยพื้นผิวของวัตถุ

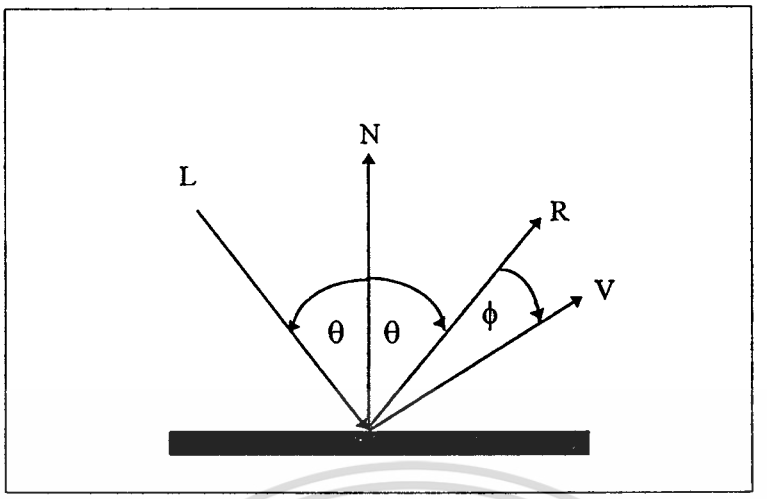
ภาพที่ 36



แสดง การสะท้อนของแสงที่พื้นผิวต่าง ๆ

ในการจำลองแสงแบบ Phong ค่าความเข้มของแสงสะท้อนที่มุม ϕ จากมุมของแสงสะท้อนจะขึ้นกับค่าของ $\cos^n\phi$ ดังแสดงในภาพที่ 37 เมื่อ n คือคุณสมบัติของพื้นผิวของวัตถุที่เกี่ยวกับความมันวาว เมื่อ ϕ คือ 0 ค่าของ $\cos^n\phi$ จะมีค่าเท่ากับ 1 ดังนั้น ค่าของการสะท้อนแสงแบบกรวยจะมีค่ามากที่สุดเมื่อ ϕ มีค่าเท่ากับมุมของแสงสะท้อน และค่า n ของวัตถุที่มีพื้นผิวด้านจะมีค่าเท่ากับ 1 ส่วนวัตถุที่มีพื้นผิวมันวาว จะมีค่า n มากกว่า 1

ภาพ 37



แสดง มุมตกกระทบของแสงและมุมมอง

และจากค่าสัมประสิทธิ์การสะท้อนของพื้นผิวของวัตถุที่จะมีค่าเปลี่ยนแปลงตามมุมของแสงตกกระทบ เช่น สำหรับพื้นผิวที่เป็นน้ำหรือแก้ว เมื่อมุมของแสงตกกระทบมีค่ามาก จะทำให้สัมประสิทธิ์การสะท้อนมีค่ามากและ เมื่อมุมตกกระทบของแสงมีค่าน้อย ค่าสัมประสิทธิ์การสะท้อนจะมีค่าน้อยตามไปด้วย การเปลี่ยนแปลงของค่านี้ขึ้นกับค่าความยาวคลื่นของแสง (wavelength, λ) ของแสงตกกระทบซึ่งเขียนได้ในรูป $w(\theta,\lambda)$ เรียกว่า ค่าสัมประสิทธิ์การสะท้อนแสงแบบทรงกรวย (specular reflection coefficaient, specular reflectance)จากความสัมพันธ์ทั้งหมดสามารถคำนวณหาค่าของการสะท้อนแสงแบบกรวย[14] R_s คือ

$$R_s = \frac{E_p}{D + D_0} w(\theta, \lambda) \cos^n\phi \qquad (34)$$

สำหรับพื้นผิวทั่ว ๆ ไปการเปลี่ยนแปลงของ $w(\theta,\lambda)$ มีค่าน้อยซึ่งสามารถแทนได้โดยค่าคงที่ k_s และ $\cos\phi = V.R$ เมื่อ V คือทิศทางของผู้สังเกต

$$R_s = \frac{E_p}{D + D_0} k_s (V.R)^n \qquad (35)$$

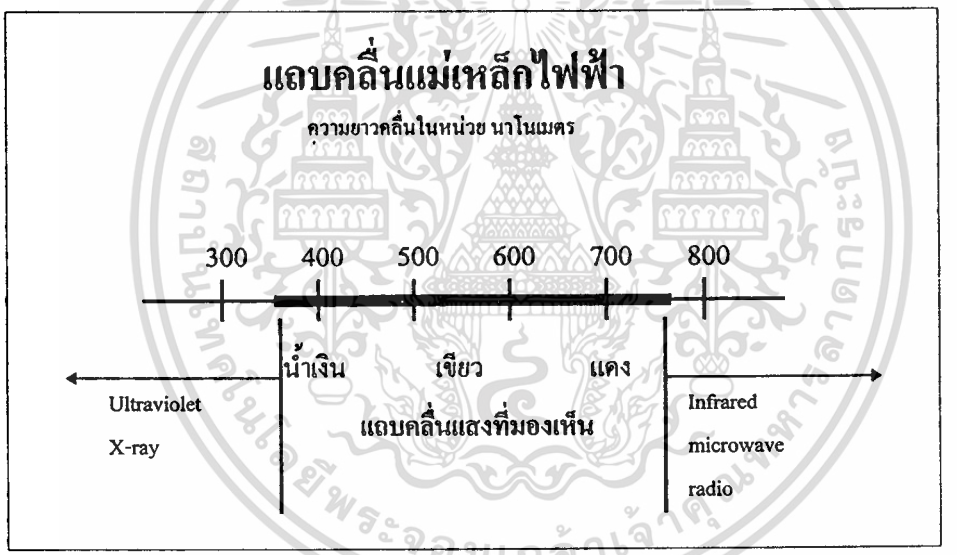
ดังนั้นสมการการสะท้อนแสงที่ผิวของวัตถุคือ

$$I = K_d I_a + \frac{E_p}{D + D_0} (K_d(N.L) + k_s (V.R)^n) \qquad (36)$$

แสงและสี

สีพื้นฐานของจอภาพของคอมพิวเตอร์คือ สีแดง (Red) สีเขียว (Green) และสีน้ำเงิน(Blue) ดังนั้น จึงจำเป็นต้องใช้วิธีการบางอย่างมาช่วยในการแสดงผลของสีอื่น ๆ ที่สร้างขึ้นมาจากสีพื้นฐานทั้ง 3 สี [15] แสงที่สามารถมองเห็นได้ (Visible light) นั้นเป็นส่วนประกอบส่วนหนึ่งของคลื่นแม่เหล็กไฟฟ้า (electromagnetic wave) รวมทั้ง คลื่นวิทยุ (Radio wave) หรือ รังสีเอกซ์ (X-ray) ความแตกต่างของคลื่นเหล่านี้คือค่าความยาวคลื่น ซึ่งคลื่นวิทยุจะมีช่วงความยาวคลื่นที่ยาวกว่าความยาวคลื่นของรังสีเอกซ์ ส่วนแสงที่มองเห็นได้นั้นจะอยู่ในช่วงความยาวคลื่นช่วงสั้นๆ ที่ต่อเนื่องกัน[13] ภาพที่ 38 แสดงช่วงความยาวคลื่นของคลื่นแม่เหล็กไฟฟ้าที่มองเห็นได้เรียกว่า แถบของคลื่นแสงที่มองเห็น (visible spectrum) ซึ่งมีความยาวคลื่นอยู่ระหว่าง 400 ถึง 700 นาโนเมตร (nanometer ; nm ; 10^{-9} meter)

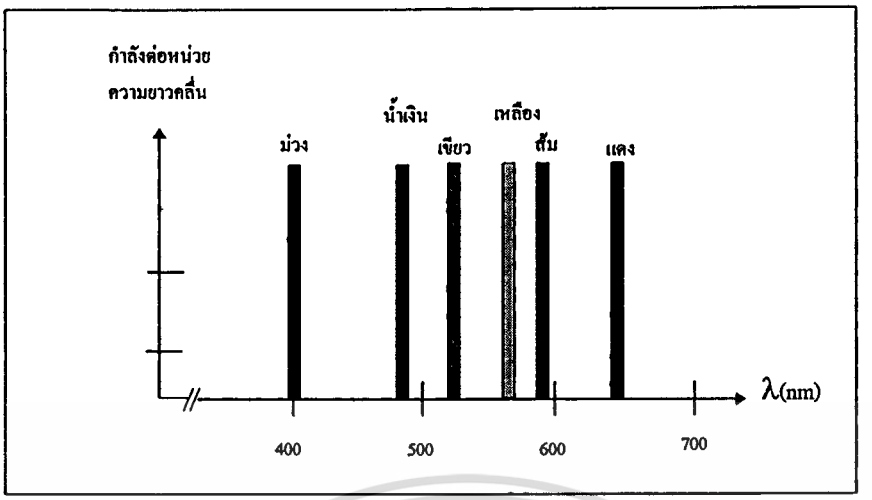
ภาพที่ 38



แสดง ความยาวคลื่นของแม่เหล็กไฟฟ้าที่สามารถมองเห็นได้

ภายในดวงตาของมนุษย์นั้นจะมีเนื้อเยื่อที่บริเวณผนังรอบ ๆ ดวงตาที่มีความไวต่อแสงและประกอบด้วย เซลตัวรับ (receptor cell) 2 ชนิดด้วยกันคือ เซลแบบกรวย (Cone cell) และเซลแบบแท่ง (rod cell) เซลแบบกรวยจะเป็นเซลที่จะตอบสนองต่อสี โดยที่มันจะตอบสนองต่อสีแดง, สีเขียวและสีน้ำเงิน สีต่างๆที่มองเห็นนั้นจะเป็นผลรวมของการตอบสนองต่อสีแดง, สีเขียวและสีน้ำเงิน เซลแบบแท่งจะตอบสนองต่อแสงที่มีความเข้มต่ำๆได้ดีกว่า ภาพที่ 39 แสดงสีต่างๆที่มองเห็นได้

ภาพที่ 39



แสดง สีต่าง ๆ ที่มองเห็นได้

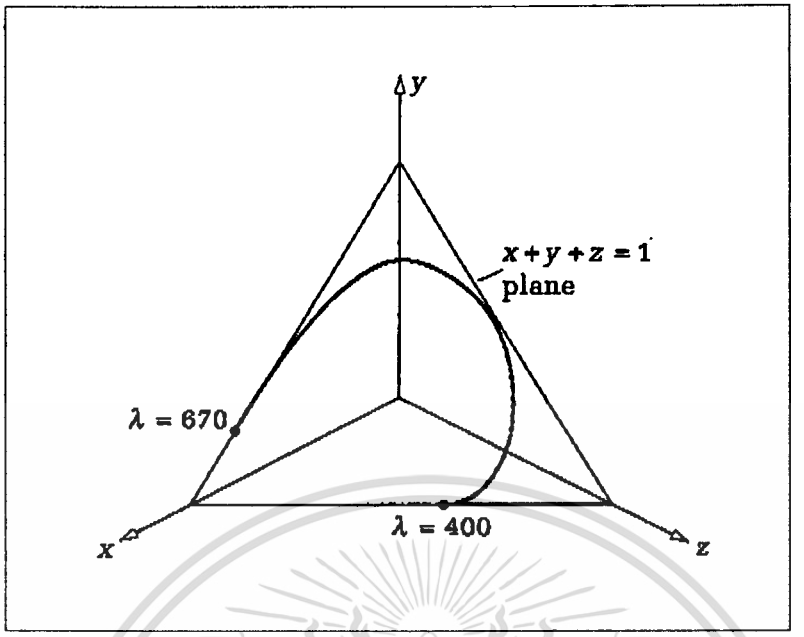
ไดอะแกรม CIE Chromaticity

การกำหนดสีของแสงที่ต้องการโดยการผสมกันของค่าคงที่ 3 ค่าเป็นสิ่งที่มีความสำคัญมาก ในระบบคอมพิวเตอร์กราฟิก ในปี ค.ศ.1931 Commission Internationale de L'Eclairage (CIE) [16] ได้กำหนดตัวแปรมาตรฐาน 3 ตัวด้วยกัน เรียกว่า X, Y และ Z เพื่อแทนค่าของสีแดง สีเขียวและสีน้ำเงิน [17] ในการผสมกันของสีเพื่อให้ได้สีของแสงที่ต้องการ ถ้า C คือสีที่ต้องการ

$$C = xX + yY + zZ \qquad (37)$$

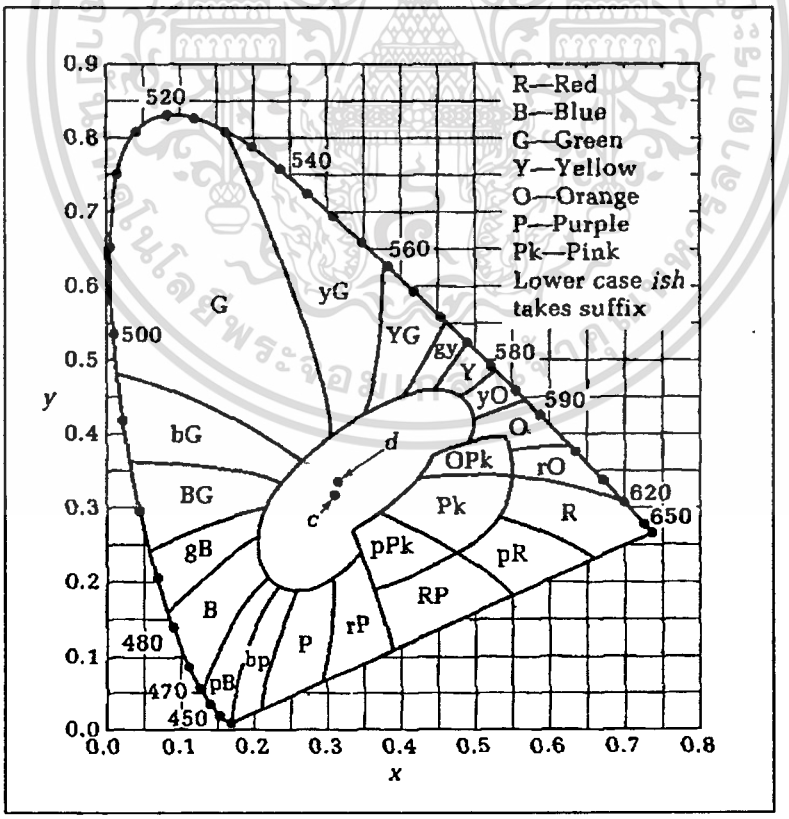
และ การกำหนดค่าของ x, y และ z สามารถหาได้จากที่ระนาบ x+y+z = 1 ดังแสดงในภาพที่ 40 ดังนั้น เส้นโค้งของแถบสีบริสุทธิ์ (pure spectrum color curve) คือ $S(\lambda) = (X(\lambda), Y(\lambda), Z(\lambda))$ จะอยู่ภายในบริเวณค่าที่เป็นบวกของรูปกราฟ เนื่องจากเส้นโค้งของแถบสีถูกกำหนดบนระบบ 3 มิติบนระนาบที่ x+y+z = 1 ดังนั้นจึงสามารถที่จะแทนค่าไปสู่ระบบ 2 มิติได้ และ Standard CIE Chromaticity Diagram คือเส้นโค้งของ $S'(\lambda) = (X(\lambda), Y(\lambda))$ ดังแสดงในภาพที่ 41

ภาพที่ 40



แสดง การสร้างมาตรฐาน CIE

ภาพที่ 41



แสดง ไดอะแกรม CIE Chromaticity

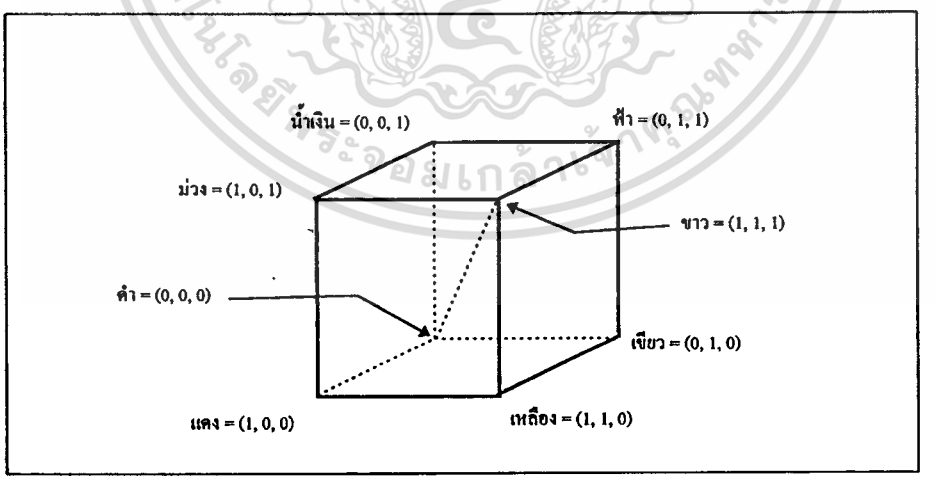
แบบจำลองของสี

แบบจำลองของสี (color model) คือการกำหนดลักษณะของสีที่ใช้ในระบบคอมพิวเตอร์กราฟิกเพื่อความสะดวกในการใช้งาน แบบจำลองของสีที่กำหนดนั้นสามารถแยกออกได้เป็นแบบจำลองของสีที่ใช้สำหรับอุปกรณ์ต่างๆในระบบคอมพิวเตอร์ และแบบจำลองของสีที่ใช้ในระบบชีวิตประจำวันที่ประกอบด้วย ชนิดของสี (hue), ความอิ่มตัวของสี (Saturation) และความสว่างของสี (Brightness)

แบบจำลองของสีแบบ RGB

เป็นแบบจำลองของสีที่ใช้ในระบบจอภาพของเครื่องคอมพิวเตอร์[14] โดยประกอบจากสีแดง, สีเขียวและสีน้ำเงิน แบบจำลองของสีชนิดนี้ถูกกำหนดขึ้นในระบบพิกัดแบบ Cartesian ค่าของตัวแปรทั้ง 3 ตัว คือ R, G และ B นั้น สามารถที่จะนำมารวมกันได้โดยตรงเพื่อทำให้ได้สีที่ต้องการ โดยตัวแปร R, G และ B จะใช้แทนค่าของสีแดง, สีเขียว และสีน้ำเงินตามลำดับ[18] แบบจำลองของสีแบบ RGB นี้ถูกกำหนดโดยรูปลูกบาศก์ขนาด 1 หน่วยดังภาพที่ 42 โดยแนวเส้นทะแยงมุมของรูปลูกบาศก์นั้น ค่าของตัวแปรทั้งสามจะมีค่าเท่ากันและสีที่เกิดจากค่าของตัวแปรทั้งสามตามเส้นทะแยงมุมของรูปลูกบาศก์ คือ ระดับของสีเทา (gray level) และสีดำจะอยู่ที่บริเวณจุดกำเนิด ที่ตัวแปร (R, G, B) มีค่าเป็น (0, 0, 0) ส่วนสีขาวจะอยู่ที่ ตำแหน่ง (1, 1, 1)

ภาพที่ 42



แสดง แบบจำลองสีแบบ RGB

แบบจำลองของสีแบบ CMY

แบบจำลองของสีแบบ CMY[14] เป็นแบบจำลองที่ใช้เป็นมาตราฐานในอุปกรณ์การพิมพ์ต่างๆ เช่น ink jet plotter ตัวแปรทั้ง 3 ของแบบจำลองของสีชนิดนี้ คือ C (Cyan, สีฟ้า), M (Magenta ,สีม่วง) และ Y (Yellow,สีเหลือง) และตัวแปรทั้ง 3 นี้ คือ C, M และ Y เป็นส่วนประกอบของตัวแปรของสีในแบบจำลองแบบ RGB ด้วย ตัวแปรทั้ง 3 นี้ถูกกำหนดขึ้นมาคล้ายกับตัวแปรในระบบ RGB แต่ต่างกันที่สีขาวจะอยู่ที่บริเวณจุดกำเนิด (0, 0, 0) แทนสีดำในแบบจำลองแบบของสีแบบ RGB การกำหนดสีของแบบจำลอง CMY นี้ กระทำได้โดยการแยกหรือลบสีที่ไม่ต้องการออกจากสีขาว แทนที่จะเป็นการเพิ่มสีที่ต้องการในสีดำตามแบบจำลองของสีแบบ RGB เช่น สีฟ้า ได้จากสีขาวลบด้วยสีแดงหรือ เกิดจากการรวมกันของสีน้ำเงินและเขียว ส่วนสีม่วงจะดูดกลืนสีเขียว ดังนั้นจะเกิดจากสีแดงผสมกับสีน้ำเงิน สีเหลืองจะดูดกลืนสีน้ำเงิน จึงเป็นผลของสีแดงผสมกับสีน้ำเงิน ดังนั้นพื้นผิวใดๆที่มีสีที่เกิดจากการผสมกันของสีฟ้าและสีเหลือง จะดูดกลืนสีแดงและน้ำเงิน ดังนั้น เมื่อมีแสงสีขาวมาตกกระทบ ส่วนของสีเขียวเท่านั้นจะสะท้อนออกมา ส่วนพื้นผิวที่มีสีที่เกิดจากการผสมกันของสีฟ้า สีเหลือง และสีม่วง จะดูดกลืน สีแดง สีเขียวและสีน้ำเงิน เมื่อมีแสงสีขาวมาตกกระทบกับพื้นผิวจะทำให้เห็นเป็นสีดำ ดังนั้นสามารถกำหนดสมการของการเปลี่ยนแปลงจากแบบจำลองของสีแบบ RGB และแบบจำลองของสีแบบ CMY ได้คือ

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (38)$$

แบบจำลองของสีแบบ YIQ

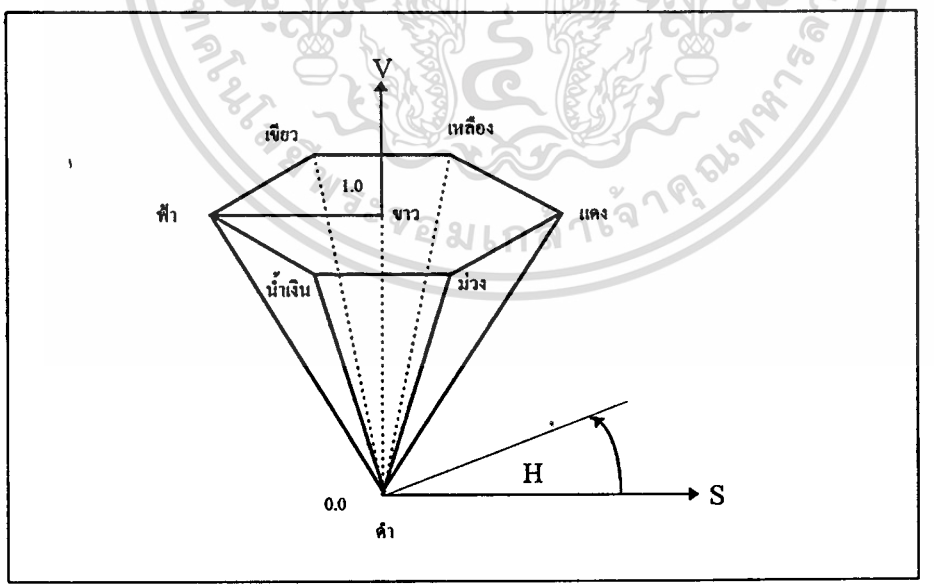
แบบจำลองของสีชนิดนี้ถูกใช้ในการแพร่ภาพทางโทรทัศน์ ในระบบของ National Television System Committee (NTSC) แบบจำลองของสีชนิดนี้อาศัยค่าพื้นฐานจากแบบจำลองของสีแบบ RGB โดยจะทำการเรียงข้อมูลของตัวแปร RGB เพื่อคุณภาพของการสื่อสาร ตัวแปร Y ในแบบจำลองสีแบบ YIQ[14] คือ ความสว่าง (Luminance) ของสีแบบเดียวกับที่ใช้ในระบบ CIE และค่าของตัวแปร Y เท่านั้นที่ใช้ในการส่งสัญญาณโทรศัพท์จะถูกแสดงในระบบของโทรทัศน์ระบบขาวดำ ส่วนค่าของสีต่างๆจะถูกกำหนดโดยตัวแปร I และ Q ซึ่งสามารถกำหนดสมการของการเปลี่ยนแปลงไปสู่แบบจำลองของสีแบบ RGB ได้ [19] คือ

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & 0.311 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (39)$$

แบบจำลองของสีแบบ HSV

แบบจำลองของสีที่กล่าวมาแล้วข้างต้นนั้นเป็นแบบจำลองของสีที่กำหนดขึ้นมาใช้เป็นมาตราฐานสำหรับอุปกรณ์ต่างๆ แต่โดยทั่วไปแล้วเป็นการยากที่จะใช้แบบจำลองของสีเหล่านี้ในการกำหนดสีที่ต้องการสำหรับผู้ใช้โดยตรง[20] ดังนั้นแบบจำลองของสีแบบ HSV[14] จึงถูกกำหนดขึ้นมาเพื่ออำนวยความสะดวกแก่ผู้ใช้งาน ซึ่งตัวแปรทั้ง 3 ตัวได้แก่ H (Hue) ใช้ในการกำหนดสีต่างๆ S (Saturation) คือค่าความอิ่มตัวของสี และ V (Value) คือ ค่าความสว่างของสีที่ต้องการ ในบางครั้งเรียกแบบจำลองของสีแบบ HSV ว่า แบบจำลองของสีแบบ HSB (HSB Color model, B = Brightness) แบบจำลองของสีชนิดนี้ถูกกำหนดในพิกัดทรงกระบอก (Cylindrical coordinate) โดยรูปแบบที่ใช้มาจากรูปกรวยหกเหลี่ยม (Hexagonal cone) หรือรูปพิรามิดที่มี 6 ด้าน (6 - sided pyramid) ดังแสดงในภาพที่ 43 ส่วนบนสุดของรูปกรวยหกเหลี่ยมจะมีค่า V=1 ที่บริเวณนี้ของภาพจะประกอบด้วยสีที่มีค่าความสว่างมากที่ สุดค่า H หรือค่าของสีต่าง ๆ จะถูกกำหนดโดยมุมรอบ ๆ แกนตั้งโดยที่สีแดงอยู่ที่มุม 0 องศา สีเขียวอยู่ที่มุม 120 องศา และสีน้ำเงินอยู่ที่มุม 240 องศา

ภาพที่ 43



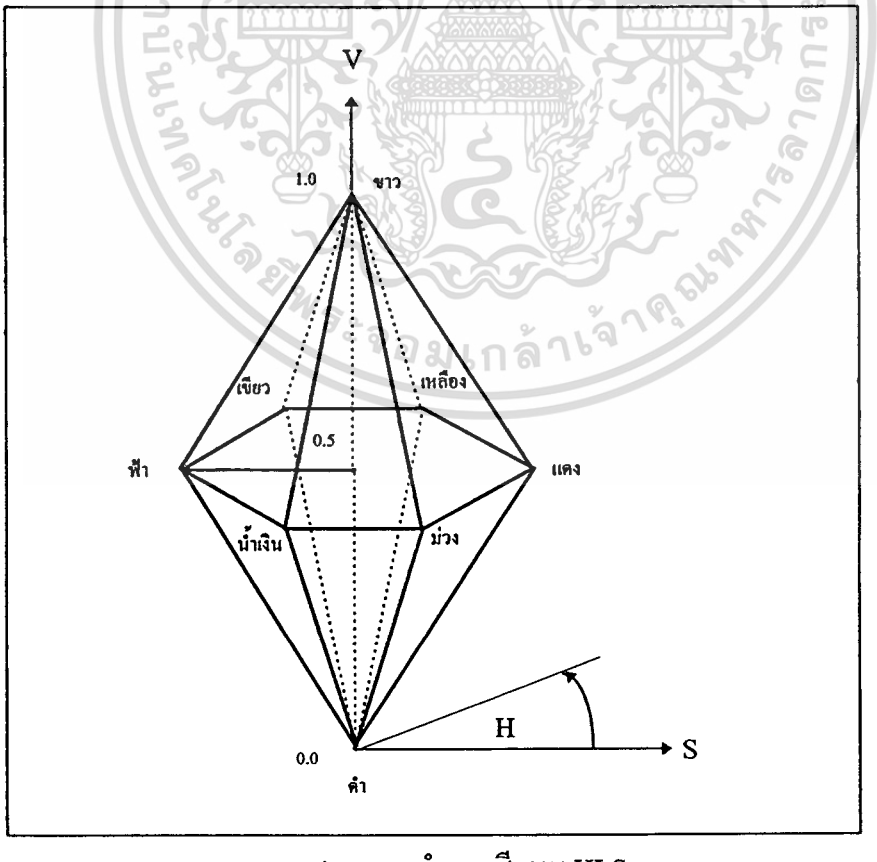
แสดง แบบจำลองสีแบบ HSV

ค่าของความสว่าง หรือ V มีค่าระหว่าง 0 ที่ตำแหน่งกลางถึง 1 ที่บริเวณสามเหลี่ยมที่ประกอบเป็นรูปกรวยหกเหลี่ยม กรวยหกเหลี่ยมที่กำหนดขึ้นมีขนาดความสูง 1 หน่วย (V) โดยมีจุดยอดของกรวยอยู่ที่บริเวณจุดกำเนิด ที่จุดยอดของรูปกรวยจะแทนที่โดยสีดำและมีค่า V = 0 ที่จุดนี้ค่าของ H และ S จะไม่มีผลของต่อค่าของสีเลย และที่ S = 0, V = 1 จะแทนที่โดยสีขาว ที่ S = 0 คือค่าในแนวแกนV ที่มีค่าระหว่าง 0 และ 1 นั้น จะแทนที่โดยระดับของสีเทา โดยที่ค่าของ H จะไม่มีผลต่อสีเช่นกัน ดังนั้นเมื่อ S มีค่าไม่เท่ากับ 0 แล้วค่าของ H จะเป็นค่าที่กำหนดสีต่าง ๆ เช่น สีแดงบริสุทธิ์ (pure red) จะมีค่า H = 0, S = 1 และ V = 1 ดังนั้นที่ตำแหน่ง S = 1 , V = 1 ค่าของสีที่ได้จะเป็นสีที่มีความบริสุทธิ์ (pure color) การเพิ่มสีขาวในสีต่างๆทำได้โดยการเปลี่ยนแปลงค่าของ S และระดับ (Shade) ของสีต่างๆ ทำได้โดยการเปลี่ยนแปลงค่าของ V

แบบจำลองของสีแบบ HLS

แบบจำลองของสีแบบ HLS[14] (Hue, Lightness, Saturation) กำหนดขึ้นโดยอาศัยรูปกรวยหกเหลี่ยม 2 รูป ที่ซ้อนกันในระบบพิกัดทรงกระบอก ดังภาพที่ 44

ภาพที่ 44



แสดง แบบจำลองสีแบบ HLS

H คือค่าของมุมรองแกนตั้งของรูปหกเหลี่ยมที่ซ้อนกันโดยมีสีแดงที่ค่ามุม 0 องศา เหมือนกับค่าของ H ในแบบจำลองของสีแบบ HSV และค่าของ S จะอยู่ที่บริเวณแกนนอน มีค่าอยู่ระหว่าง 0 ที่จุดกำเนิด และ 1 ที่ผิวของรูปหกเหลี่ยม L จะมีค่าเท่ากับ 0 ที่บริเวณจุดปลายของรูปหกเหลี่ยมรูปที่ต่ำกว่าซึ่งมีค่าเป็นสีดำ และจะมีค่าเท่ากับ 1 ที่บริเวณจุดปลายของรูปหกเหลี่ยมรูปบนซึ่งมีค่าเป็นสีขาว ระดับของสีเทาจะอยู่ที่บริเวณที่ S มีค่าเท่ากับ 0 และค่าสีที่มีความอิ่มตัวมากที่สุดจะอยู่ที่บริเวณ $S = 1, L = 0.5$

ทฤษฎีการกำหนดความเข้มของแสง

วัตถุใด ๆ ก็ตามที่มีแสงจากแหล่งกำเนิดใดๆมาตกกระทบ จะทำให้เกิดระดับความเข้มของแสงที่สะท้อนมีค่าต่าง ๆ กัน ความเข้มของแสงที่สะท้อนนั้นจะขึ้นอยู่กับลักษณะของแสงที่มาจากต้นกำเนิดและคุณสมบัติบางประการของวัตถุเอง ทฤษฎีการกำหนดความเข้มของแสงที่พื้นผิวของวัตถุเมื่อมีแสงมาตกกระทบ สามารถแยกได้เป็น

การกำหนดความเข้มของแสงแบบคงที่ (Constant shading)

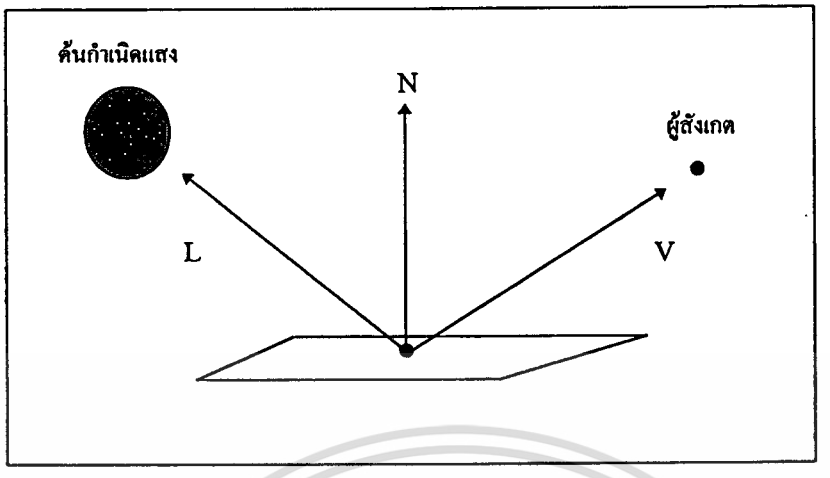
การกำหนดความเข้มของแสงแบบคงที่[12] เป็นวิธีการกำหนดความเข้มของแสงที่สะดวกและง่ายที่สุด เนื่องจากในวิธีนี้ จะคำนวณค่าความเข้มของแสงที่สะท้อนจากพื้นผิวเพียงครั้งเดียว เพราะถือว่า ค่าความเข้มของแสงที่สะท้อนจากพื้นผิวของวัตถุนั้นมีค่าเท่ากันเสมอ วิธีการนี้จะต้องอยู่ภายใต้ข้อกำหนดที่ว่า

1. ต้นกำเนิดแสงอยู่ที่ infinity ดังนั้น เวกเตอร์ N.L จะมีค่าคงที่ทุกตำแหน่งบนพื้นผิวของวัตถุ เมื่อ เวกเตอร์ N คือ เวกเตอร์ที่ตั้งฉากของพื้นผิว และ เวกเตอร์ L คือทิศทางของต้นกำเนิดแสง ดังแสดงในภาพที่ 45

2. ผู้สังเกตอยู่ที่อนันต์ (infinity) ดังนั้น เวกเตอร์ N.V จะมีค่าคงที่ทุกตำแหน่งบนพื้นผิวของวัตถุ เมื่อ เวกเตอร์ V คือ เวกเตอร์ของผู้สังเกต

3. พื้นผิวของวัตถุจะต้องเป็นวัตถุผิวเรียบ

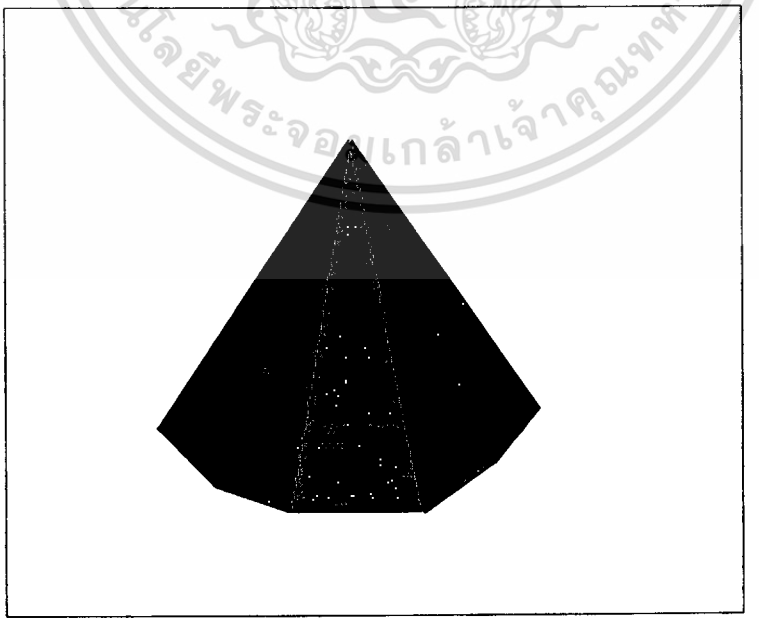
ภาพที่ 45



แสดง ทิศทางการสะท้อนของแสง

จากข้อกำหนดทั้ง 3 ทำให้ค่าความเข้มของแสงจากสมการที่ (36) มีค่าคงที่เสมอทุกตำแหน่งของพื้นผิวของวัตถุ ในกรณีที่วัตถุประกอบมาจากรูปหลายเหลี่ยม (polygon) หลายภาพนั้น วิธีการนี้จะทำให้เกิดความไม่ต่อเนื่องของความเข้มของแสงที่บริเวณรอยต่อของรูปหลายเหลี่ยมแต่ละรูป [21] ดังแสดงในภาพที่ 46 บางครั้งเรียกวิธีการกำหนดความเข้มของแสงแบบนี้ว่า การกำหนดความเข้มของแสงแบบเรียบ (flat shading)

ภาพที่ 46

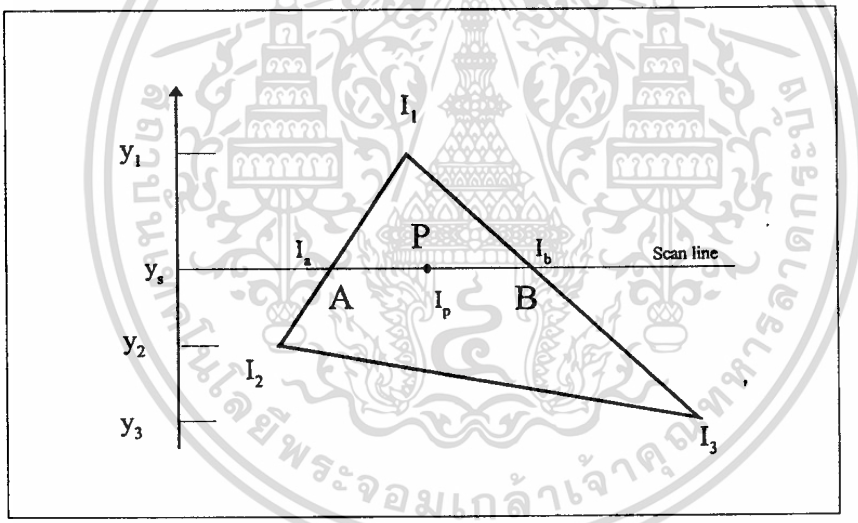


แสดง วิธี Constant shading

การกำหนดความเข้มของแสงแบบ Gouraud

วิธีการกำหนดความเข้มของแสงโดยวิธีของ Gouraud [16] หรือ intensity interpolation shading นี้จะทำให้ไม่มีผลของความไม่ต่อเนื่องกันของความเข้มของแสงที่บริเวณรอยต่อของรูปหลายเหลี่ยมที่ประกอบเป็นวัตถุ มีหลักการคือ การกำหนดความเข้มของแสงที่ตำแหน่งต่างๆบนพื้นผิวของรูปหลายเหลี่ยมจะอาศัยค่าเฉลี่ยของค่าความเข้มของแสงจากค่าความเข้มของแสงที่จุดปลาย (vertex) ต่างๆของรูปหลายเหลี่ยม ดังนั้นสิ่งที่สำคัญที่สุดคือ จะต้องทราบเวกเตอร์ที่ตั้งฉาก (normal vector) ที่จุดปลายต่างๆของรูปหลายเหลี่ยม เพื่อใช้ในการคำนวณค่าความเข้มของแสง เมื่อทราบค่าความเข้มที่จุดปลายต่างๆของรูปหลายเหลี่ยมแล้วจึงสามารถหาค่าความเข้มของแสงที่ตำแหน่งใด ๆ บนรูปหลายเหลี่ยมได้โดยอาศัยวิธีการของ Scan line เข้ามาช่วย ดังแสดงในภาพที่ 47

ภาพที่ 47



แสดง การหาค่าความเข้มของแสงโดยวิธี Gouraud

จากภาพที่ 47 ค่าความเข้มที่จุด P สามารถหาได้โดย

I_1 = ความเข้มของแสงที่จุดปลายที่ 1

I_2 = ความเข้มของแสงที่จุดปลายที่ 2

I_3 = ความเข้มของแสงที่จุดปลายที่ 3

I_a = ความเข้มของแสงที่จุดปลายที่ a

I_b = ความเข้มของแสงที่จุดปลายที่ b

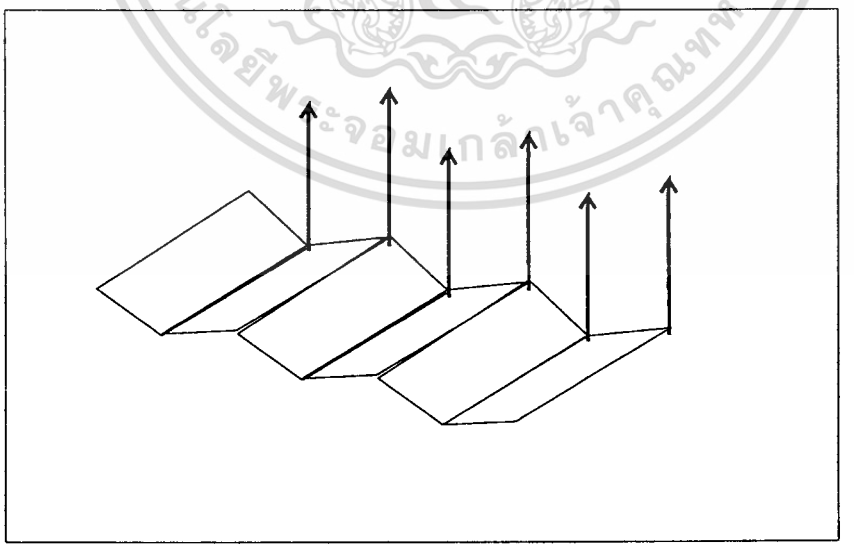
I_p = ความเข้มของแสงที่จุดปลายที่ p

$$\left.\begin{aligned} I_a &= I_1 - (I_1 - I_2)\,\frac{y_1 - y_s}{y_1 - y_2} \\ I_b &= I_1 - (I_1 - I_3)\,\frac{y_1 - y_s}{y_1 - y_3} \\ I_p &= I_b - (I_b - I_a)\,\frac{x_a - x_p}{x_b - x_a} \end{aligned}\right\} \qquad (40)$$

การกำหนดความเข้มของแสงแบบ Phong

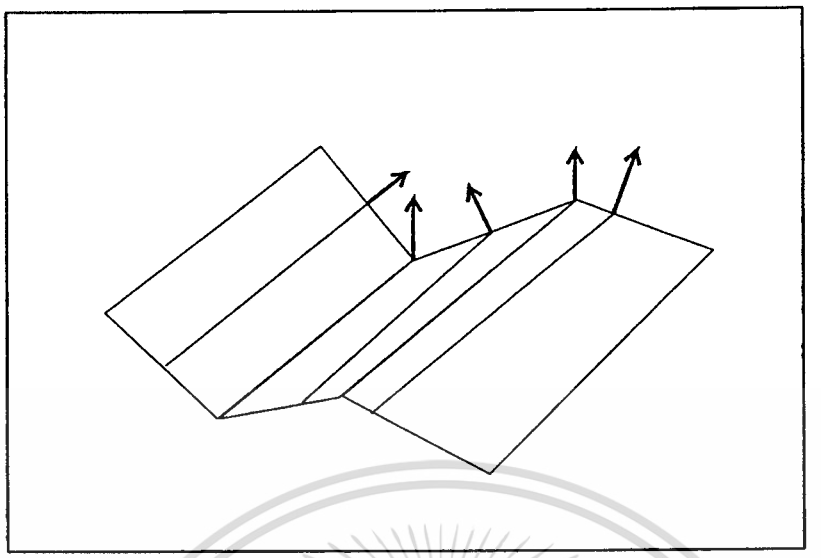
การกำหนดความเข้มของแสงแบบ Phong[12] หรือ normal-vector interpolation shading มีวิธีการคำนวณคล้ายกับวิธีการของ Gouraud Shading แต่แตกต่างกันที่ ในวิธีการของ Phong shading นั้นอาศัยเวกเตอร์ที่ตั้งฉากมาใช้ในการคำนวณค่าโดยประมาณของความเข้มของแสงบนวัตถุแทนการใช้ความเข้มของแสงตามแบบ Gouraud[14] จากภาพที่ 47 การหาค่าความเข้มของแสงที่จุด P สามารถกระทำได้โดยการหาค่าเวกเตอร์ที่ตั้งฉาก ที่บริเวณจุดยอดของรูปหลายเหลี่ยมก่อนและค่าของเวกเตอร์ที่ตั้งฉากที่จุด A และ B นั้นสามารถประมาณได้จากค่าของเวกเตอร์ที่ตั้งฉากที่จุด 1,2 และจุด 1,3 โดยวิธีเดียวกันค่าเวกเตอร์ที่ตั้งฉากที่จุด P ก็จะสามารถประมาณได้จากเวกเตอร์ที่ตั้งฉากที่จุด A และ B ซึ่ง เวกเตอร์ที่ตั้งฉากที่จุด P นี้จะทำให้สามารถนำไปคำนวณหาค่าความเข้มของแสงที่จุด P ได้ ภาพที่ 48 และ 49 แสดงพื้นผิวที่ใช้ในการกำหนดความเข้มของแสงแบบ Gouraud และ Phong ตามลำดับ

ภาพที่ 48



แสดง การกำหนดความเข้มของแสงแบบ Gouraud

ภาพที่ 49



แสดง การกำหนดความเข้มของแสงแบบ Phong

จากภาพที่ 48 เมื่อทำการกำหนดค่าความเข้มของแสงโดยวิธี Gouraud ของพื้นผิวดังกล่าว ค่าความเข้มที่ได้จะมีค่าเท่ากันเนื่องจากค่าเฉลี่ยของค่าเวกเตอร์ที่ตั้งฉากจะมีค่าเท่ากัน แต่ในกรณีของการกำหนดค่าความเข้มของแสงโดยวิธี Phong ดังภาพที่ 49 นั้นพื้นที่ที่กำหนดจะถูกแบ่งออกเป็นพื้นที่ย่อยๆ ทำให้ค่าของเวกเตอร์ตั้งฉากที่จุดยอดของพื้นที่มีค่าแตกต่างกันด้วย ดังนั้นค่าความเข้มของแสงของพื้นผิวที่ได้จะมีค่าแตกต่างกัน

สรุป

แสง, สี และการจัดลำดับความเข้มของแสงเป็นวิธีหนึ่งที่ทำให้ภาพจำลองที่ได้มีลักษณะเหมือนจริงมากขึ้น [22] เนื่องจากค่าความเข้มของแสงที่แตกต่างกันที่ปรากฏบนวัตถุจะทำให้สามารถที่จะมองเห็นลักษณะรูปร่างของวัตถุได้ดีขึ้น [23, 24] ในบทต่อไปจะได้กล่าวถึงการสร้างภาพจำลองของวัตถุที่ใช้วิธีการที่ได้กล่าวไปแล้วไม่ว่าจะเป็นเรื่องการกำหนดลักษณะของวัตถุในระบบต่างๆ การกำหนดค่าความเข้มของแสง และการกำหนดค่าของสี มาช่วยในการสร้างภาพจำลอง

บทที่ 5

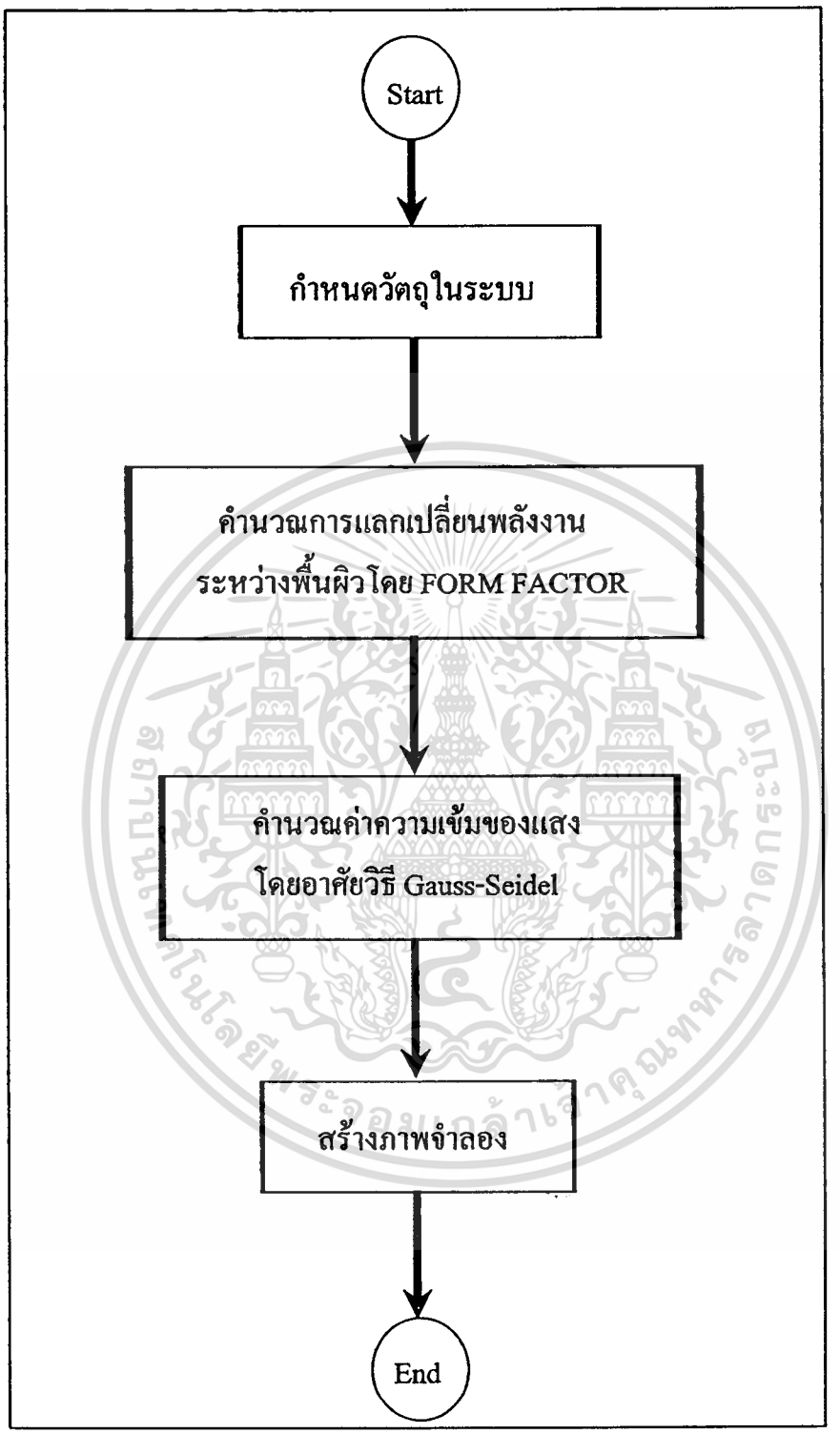
Radiosity

ในบทที่ผ่านมา ได้กล่าวถึง วิธีการพื้นฐานต่างๆ ของระบบคอมพิวเตอร์กราฟิก วิธีการเหล่านี้จะถูกนำมาใช้ในการสร้างรูปแบบของวัตถุที่ใช้ในการสร้างภาพจำลองที่เหมือนจริง หลักการที่ใช้ในการสร้างภาพจำลองที่เหมือนจริงนั้นสามารถแยกออกได้เป็น 2 วิธีการด้วยกัน ตามวิธีการสร้างภาพจำลอง คือวิธีการจำลองภาพแบบ Ray tracing ที่จะต้องอาศัยตำแหน่งของผู้สังเกตและทิศทางของแสงที่สะท้อนมาช่วยในการสร้างภาพจำลอง วิธีการจำลองภาพแบบ Ray tracing นั้นจะกล่าวถึงในภายหลัง วิธีการจำลองภาพที่เหมือนจริงวิธีที่ 2 คือ การจำลองภาพแบบ Radiosity[25] วิธีการจำลองภาพแบบนี้อาศัยหลักการแลกเปลี่ยนพลังงานของแสงระหว่างพื้นผิวของวัตถุ ดังนั้นการจำลองภาพโดยวิธี Radiosity นั้นค่าความเข้มของแสงที่คำนวณได้จะไม่ขึ้นกับตำแหน่งของผู้สังเกตเลย [26]

ทฤษฎีพื้นฐาน

การจำลองภาพเหมือนจริงโดยวิธี Radiosity นั้นได้พัฒนามาจากหลักการแลกเปลี่ยนความร้อนระหว่างพื้นผิวของวัตถุในระบบปิดใด ๆ จากหลักการคงที่ของพลังงาน (Conservation of energy) ค่าของพลังงานในระบบปิดใดๆ ปริมาณของพลังงานที่ถูกส่งออกจากวัตถุจะมีค่าเท่ากับพลังงานที่วัตถุได้รับ พลังงานในระบบปิดนั้นจะได้จากพื้นผิวใด ๆ ที่กระทำตัวเป็นตัวจ่ายหรือให้พลังงาน เช่น พื้นผิวที่มีความร้อนจะสามารถแผ่ความร้อนไปสู่พื้นผิวอื่น ๆ ได้ เช่นเดียวกับต้นกำเนิดแสงใดๆก็สามารถที่จะส่งพลังงานไปสู่พื้นผิวอื่นๆเช่นกัน และพื้นผิวในระบบปิดนั้นจะถูกกำหนดให้เป็นพื้นผิวแบบ Lambertian ที่สามารถสะท้อนหรือปลดปล่อยพลังงานได้อย่างสมบูรณ์ เช่น สามารถที่จะสะท้อนแสงที่มาตกกระทบได้ในทุกทิศทางโดยมีค่าความเข้มของแสงเท่ากัน หลักการของการจำลองภาพแบบ Radiosity นี้กระทำได้โดย การแบ่งพื้นผิวของวัตถุในระบบออกเป็นพื้นผิวเล็ก ๆ จากนั้นจึงคำนวณค่าของพลังงานของพื้นผิวเล็กๆนี้ [27] ที่ได้จากการแลกเปลี่ยนพลังงานกับพื้นผิวอื่นๆ พลังงานที่พื้นผิวเล็ก ๆ เหล่านี้จะสามารถนำไปประมวลผลเพื่อนำไปหาค่าความเข้มของแสงที่พื้นผิวของวัตถุได้ จากวิธีการเหล่านี้จะเห็นว่าการหาค่าพลังงาน หรือค่าความเข้มของแสงของวัตถุในระบบนั้นจะไม่ขึ้นกับทิศทางของผู้สังเกตเลย (View independent) ภาพที่ 50 แสดงวิธีการจำลองภาพแบบ Radiosity

ภาพที่ 50



แสดง หลักการจำลองภาพแบบ Radiosity

ถ้า B คือค่า Radiosity ของพื้นผิวเล็กๆ ซึ่งหมายถึง พลังงานทั้งหมดที่ถูกปลดปล่อยและจะมีค่าเท่ากับผลรวมของพลังงานที่สะท้อนและพลังงานที่ถูกส่งออกโดยพื้นผิวเล็กๆ หรือ

Radiosity x Area = Emitted energy + Reflected Energy

ถ้า P_i และ P_j เป็นพื้นที่เล็กๆ อัตราการแลกเปลี่ยนพลังงานระหว่างพื้นที่ทั้งสองจะขึ้นอยู่กับตำแหน่งและทิศทางของพื้นที่เล็กๆทั้งสองที่สัมพันธ์กัน เช่น ระยะห่างระหว่างพื้นที่ทั้งสองและตำแหน่งที่สัมพันธ์กัน พลังงานที่แลกเปลี่ยนจะมีค่ามากเมื่อพื้นที่ทั้งสองอยู่ใกล้กันและมีทิศทางที่ขนานกัน

$$B_i dA_i = E_i dA_i + \rho_i \int_j B_j dA_j F_{dA_j dA_i} \qquad (41)$$

เมื่อ E_i คือ พลังงานที่ถูกส่งออกจากพื้นผิว (Emitted Energy)

ρ_i คือ ค่าสัมประสิทธิการสะท้อน (Reflectivity) ของพื้นผิว หรืออัตราส่วนของแสงที่ตกกระทบบนพื้นผิวและแสงที่สะท้อนจากพื้นผิวไปสู่ระบบ

$F_{dA_j dA_i}$ คือ ค่า Form Factor ระหว่างพื้นผิวเล็ก ๆ dA_j และ dA_i หรืออัตราส่วนของพลังงานที่ถูกส่งออกมาจากพื้นผิว dA_j ไปสู่พื้นผิว dA_i และพลังงานทั้งหมด

สำหรับในระบบปิดใด ๆ ที่วัตถุถูกแบ่งออกเป็น n พื้นผิว

$$B_i A_i = E_i A_i + \rho_i \sum_{j=1}^{n} B_j F_{ji} A_j \qquad (42)$$

และจากภาพที่ 51 การแลกเปลี่ยนพลังงานระหว่างพื้นผิว A_i และ A_j จะขึ้นอยู่กับทิศทางและตำแหน่งที่สัมพันธ์กันของพื้นผิวเล็ก ๆ เท่านั้น ดังนั้น

$$F_{ij} A_i = F_{ji} A_i \qquad (43)$$

และ

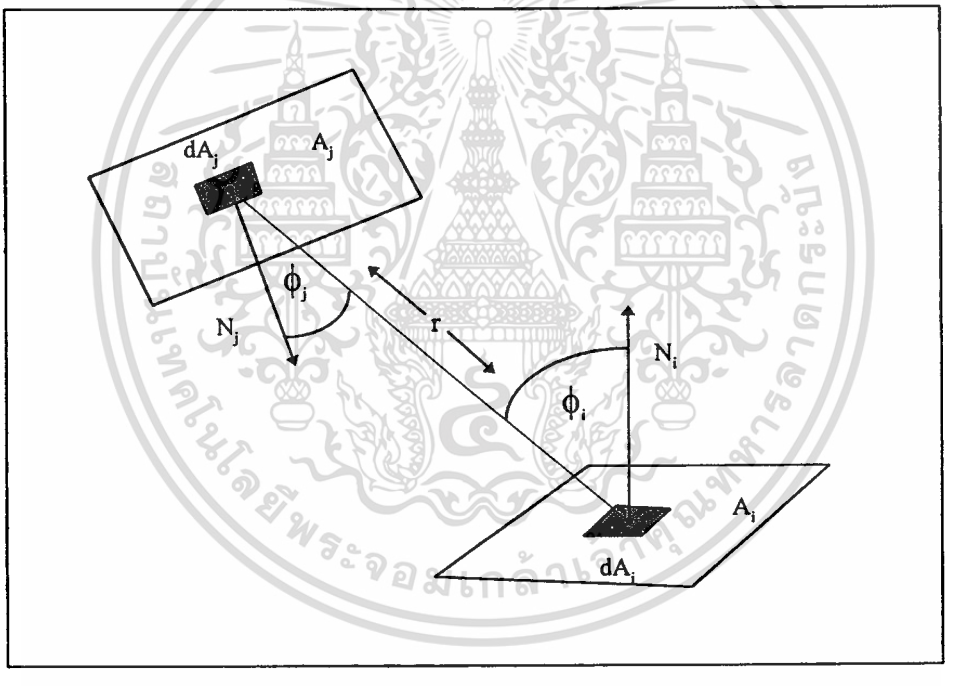
$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij} \qquad (44)$$

หรือ

$$\begin{bmatrix} 1-\rho_1F_{11} & -\rho_1F_{12} & \cdots & -\rho_1F_{1n} \\ -\rho_2F_{21} & 1-\rho_2F_{22} & \cdots & -\rho_2F_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ -\rho_nF_{n1} & -\rho_nF_{n2} & \cdots & 1-\rho_nF_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \qquad (45)$$

จากสมการที่ (45) นั้น สามารถกำหนดได้ว่า $F_{ii} = 0$ เนื่องจากพลังงานที่ถูกส่งออกจากพื้นผิวไม่สามารถที่จะตกกระทบกับพื้นผิวเดิมได้ ดังนั้น สามารถหาค่า B_i จากสมการที่ (45) ได้โดยใช้วิธีการของ Gauss-Seidel [28] ซึ่งค่า B_i สามารถที่จะนำไปใช้ในการสร้างภาพจำลองได้

ภาพที่ 51



แสดง ความสัมพันธ์ระหว่างพื้นผิวเล็กๆ

การหาค่า Form Factor

ค่า Form Factor ที่ใช้ในการจำลองภาพแบบ Radiosity นั้นหมายถึงอัตราส่วนของพลังงานที่ถูกส่งออกจากพื้นผิวหนึ่งไปสู่อีกพื้นผิวหนึ่งและค่าพลังงานทั้งหมด [29] ค่าของ Form factor จะขึ้นอยู่กับมิติหรือระบบแบบเรขาคณิตของระบบเท่านั้นไม่ขึ้นกับทิศทางการมอง หรือ ค่าความเข้มของต้นกำเนิดแสงในระบบ ดังนั้น ค่า Form Factor ระหว่างพื้นผิว A_i และ A_j ดังภาพที่ 51 คืออัตราส่วนของพลังงานที่ถูกส่งออกจากพื้นผิว A_i ที่ไปตกกระทบพื้นผิว A_j และพลังงานทั้งหมดที่ถูกส่งออกจากพื้นผิว A_i ในทุกทิศทาง

$$F_{A_iA_j} = F_{ij} = \frac{1}{A_i}\int_{A_i}\int_{A_j}\frac{\cos\phi_i \cos\phi_j}{\pi r^2}V_{ij}dA_j dA_i \qquad (46)$$

เมื่อ V_{ij} คือ ตัวแปรที่ใช้ในการระบุว่าพื้นผิว dA_i สามารถมองเห็นพื้นผิว dA_j หรือไม่ โดยที่ V_{ij} จะมีค่าเท่ากับ 1 เมื่อพื้นผิว dA_i มองเห็นพื้นผิว dA_j และ r คือระยะทางระหว่างพื้นผิว

การหาค่า Form factor โดยอาศัยรูปครึ่งสี่เหลี่ยมลูกบาศก์

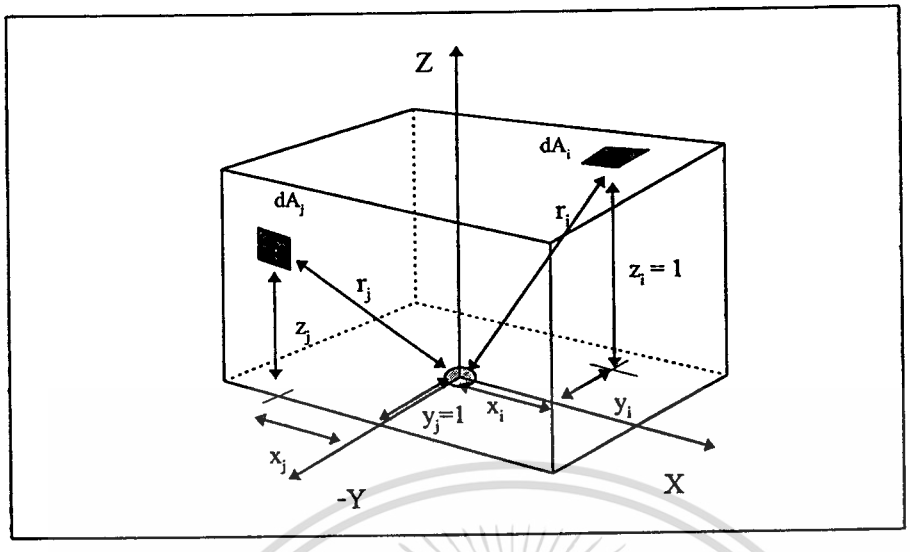
วิธีการหนึ่งที่นิยมใช้ในการหาค่า Form factor คือ การใช้รูปครึ่งสี่เหลี่ยมลูกบาศก์ (hemicube) กระทำได้โดยการสร้างรูปครึ่งสี่เหลี่ยมลูกบาศก์ขนาดความสูงหนึ่งหน่วย ที่จุดศูนย์กลางของพื้นผิว A_i โดยที่พื้นผิวของรูปครึ่งสี่เหลี่ยมลูกบาศก์จะถูกแบ่งออกเป็นพื้นผิวขนาดเล็กประกอบกันที่มีค่าตำแหน่งที่แน่นอนดังแสดงในภาพที่ 52 ที่ผิวเล็กๆเหล่านี้จะถูกคำนวณค่า Form factor ย่อยๆเรียกว่า Delta form factor (ΔF_q) ดังนั้น

$$F_{ij} = \sum \Delta F_q \qquad (47)$$

และ

$$\left.\begin{aligned} \Delta F dA_i &= \frac{1}{\pi\sqrt{x_i^2 + y_i^2 + 1}}\Delta A_i \\ \Delta F dA_j &= \frac{z_i}{\pi\sqrt{x_i^2 + y_i^2 + 1}}\Delta A_j \end{aligned}\right\} \qquad (48)$$

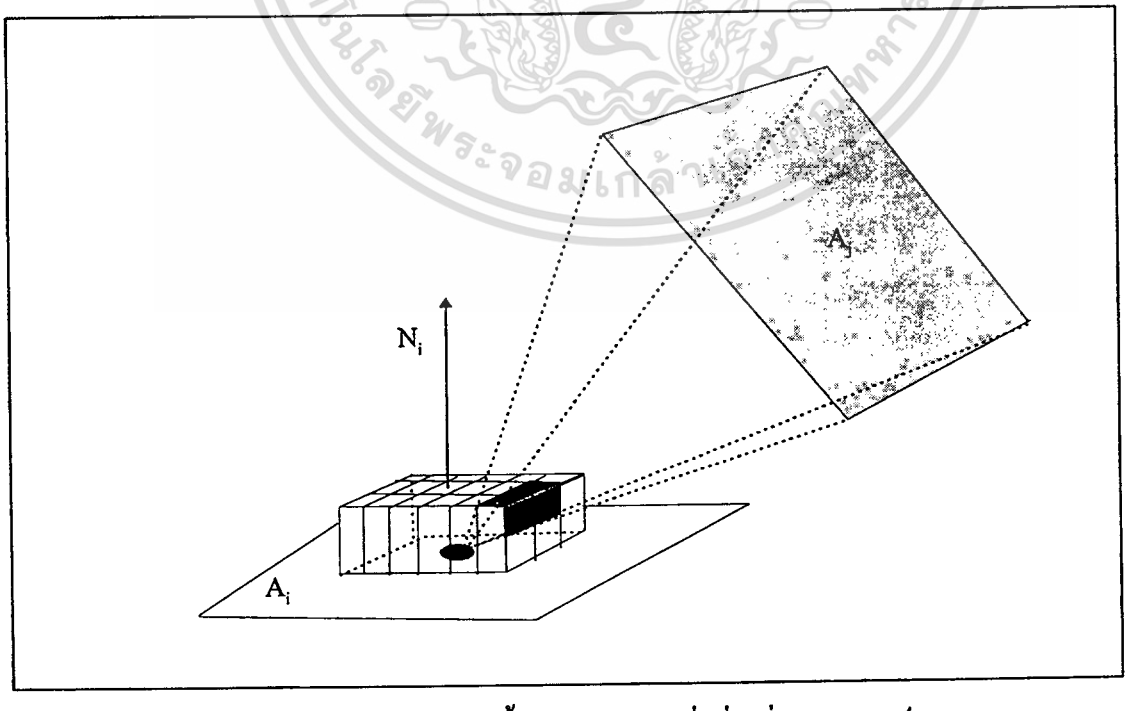
ภาพที่ 52



แสดง การกำหนดรูปครึ่งสี่เหลี่ยมลูกบาศก์

ที่พื้นผิวเล็กๆ ของรูปครึ่งสี่เหลี่ยมลูกบาศก์นั้นการหาค่า ΔF จะทำได้โดยกำหนดให้จุดศูนย์กลางของการฉายภาพอยู่ที่จุดกึ่งกลางของพื้นผิว A_i จากนั้นทำการฉายภาพของพื้นผิว A_j ลงบนพื้นผิวของรูปครึ่งสี่เหลี่ยมลูกบาศก์ ดังภาพที่ 53 พื้นผิวเล็กๆบนพื้นผิวรูปครึ่งสี่เหลี่ยมลูกบาศก์ที่ถูกฉายภาพสามารถหาค่า ΔF ได้ตามสมการที่ (58) เพื่อหาค่า F_{ij} ตามต้องการ

ภาพที่ 53

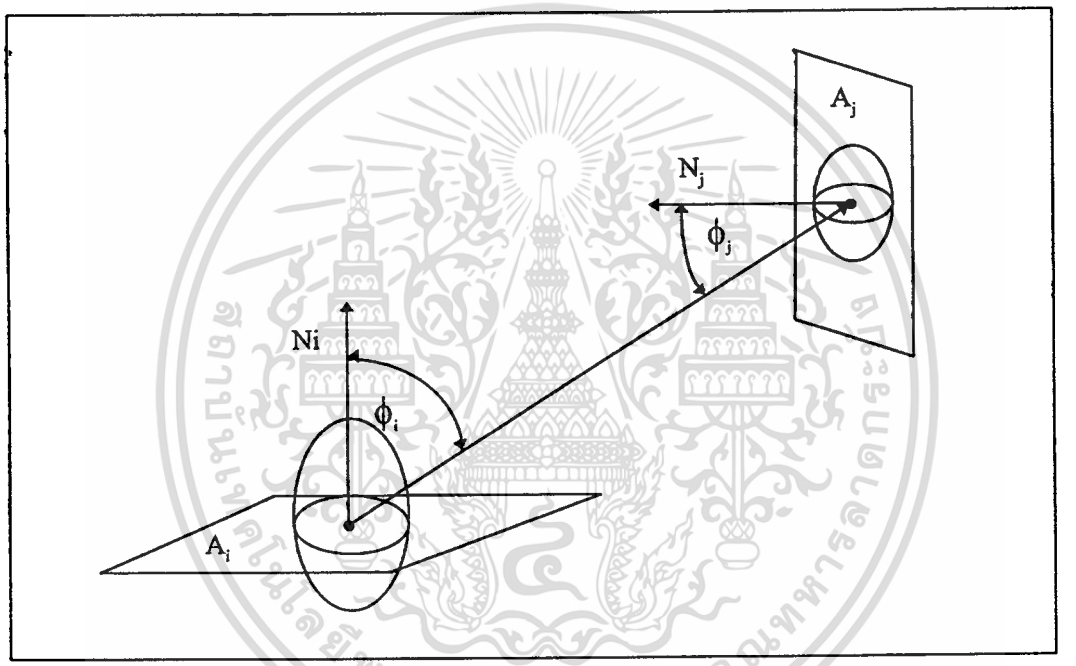


แสดง การฉายภาพพื้นผิวลงบนรูปครึ่งสี่เหลี่ยมลูกบาศก์

การหาค่า Form factor โดยอาศัยรูปทรงกลม

การหาค่า Form factor โดยการสร้างรูปครึ่งสี่เหลี่ยมลูกบาศก์นั้นจะให้ความถูกต้องมากขึ้น เมื่อจำนวนพื้นที่เล็กๆบนพื้นผิวของรูปครึ่งสี่เหลี่ยมลูกบาศก์มีจำนวนมาก แต่ก็จะทำให้ใช้เวลาในการคำนวณมากด้วย แต่การหาค่า Form factor โดยอาศัยรูปทรงกลมนั้นจะใช้เวลาที่น้อยกว่าเนื่องจากไม่มีการแบ่งพื้นผิวของทรงกลมออกเป็นพื้นผิวเล็กๆ สามารถกระทำได้โดยสร้างทรงกลมโดยมีจุดศูนย์กลางอยู่ที่จุดศูนย์กลางของพื้นผิว A_i และ A_j จากนั้นสร้างลำแสงสมมุติจากจุดศูนย์กลางของทรงกลม A_i และ A_j ดังภาพที่ 54

ภาพที่ 54



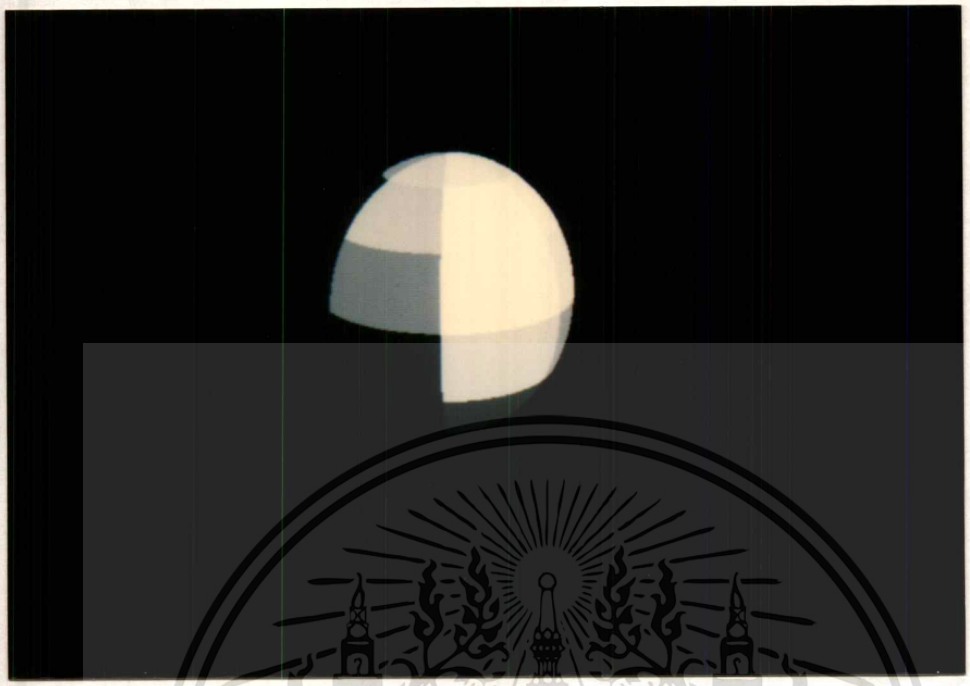
แสดง การสร้างทรงกลมบนพื้นผิว

จากนั้นทดสอบลำแสงสมมุติว่ามีการชนกับพื้นผิวอื่นๆก่อนหรือไม่ ถ้ามีการชนแสดงว่าพื้นผิว A_i ไม่สามารถมองเห็นพื้นผิว A_j ดังนั้นค่า F_{ij} จะมีค่าเท่ากับ 0 แต่ถ้าไม่มีการชนกันจะสามารถหาค่า F_{ij} ได้โดย

$$F_{ij} = \frac{\cos\phi_i}{\pi} V_{ij} \qquad (49)$$

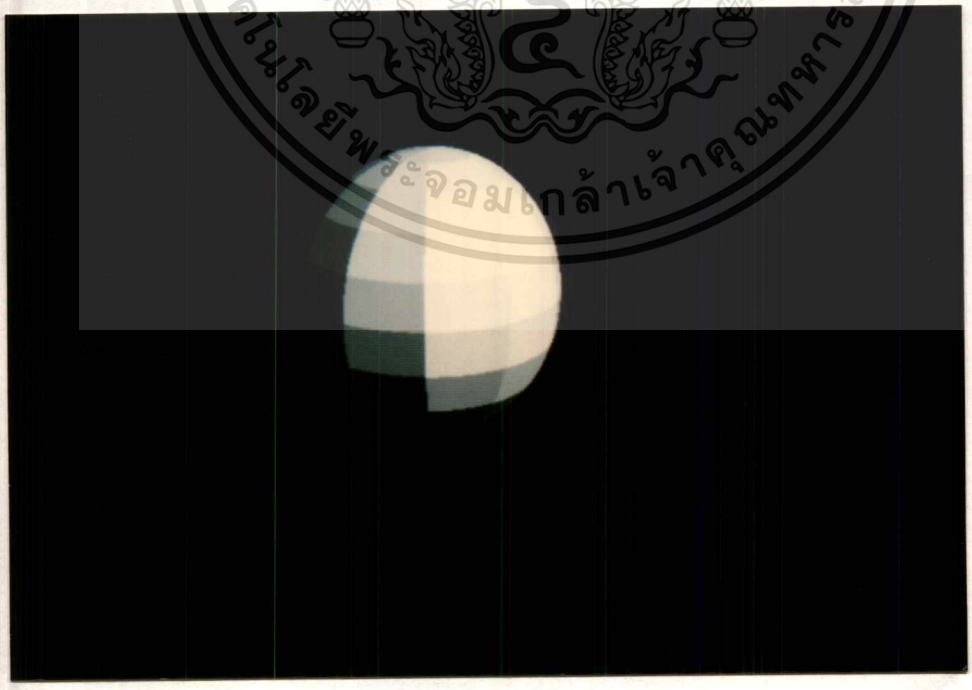
ภาพที่ 55 และ 56 แสดงภาพจำลองที่ได้จากวิธีจำลองภาพแบบ Radiosity เมื่อแบ่งวัตถุออกเป็น 36 ช่องและ 100 ช่องตามลำดับ

ภาพที่ 55



แสดง การจำลองภาพแบบ Radiosity เมื่อแบ่งวัตถุเป็น 36 ช่อง

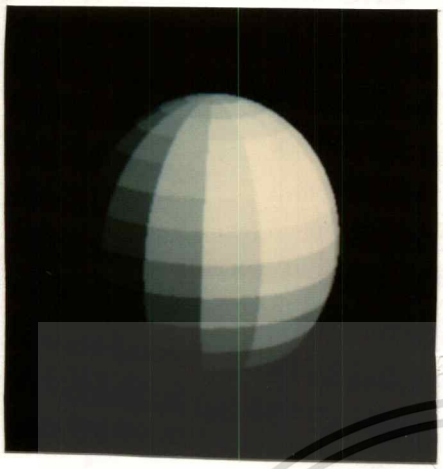
ภาพที่ 56



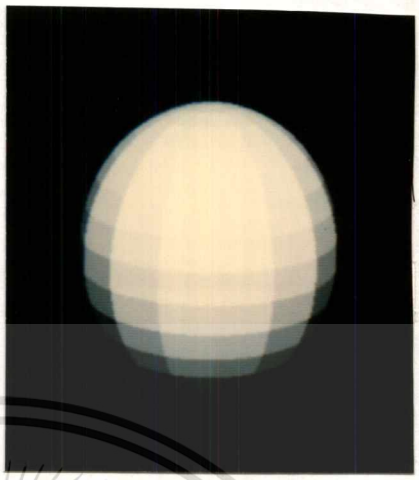
แสดง การจำลองภาพแบบ Radiosity เมื่อแบ่งวัตถุเป็น 100 ช่อง

ภาพที่ 57

57.1

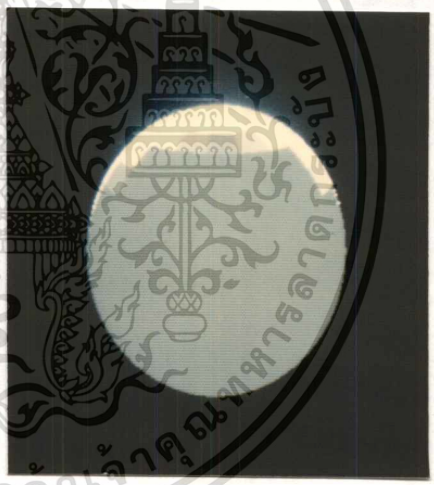


57.2



57.3

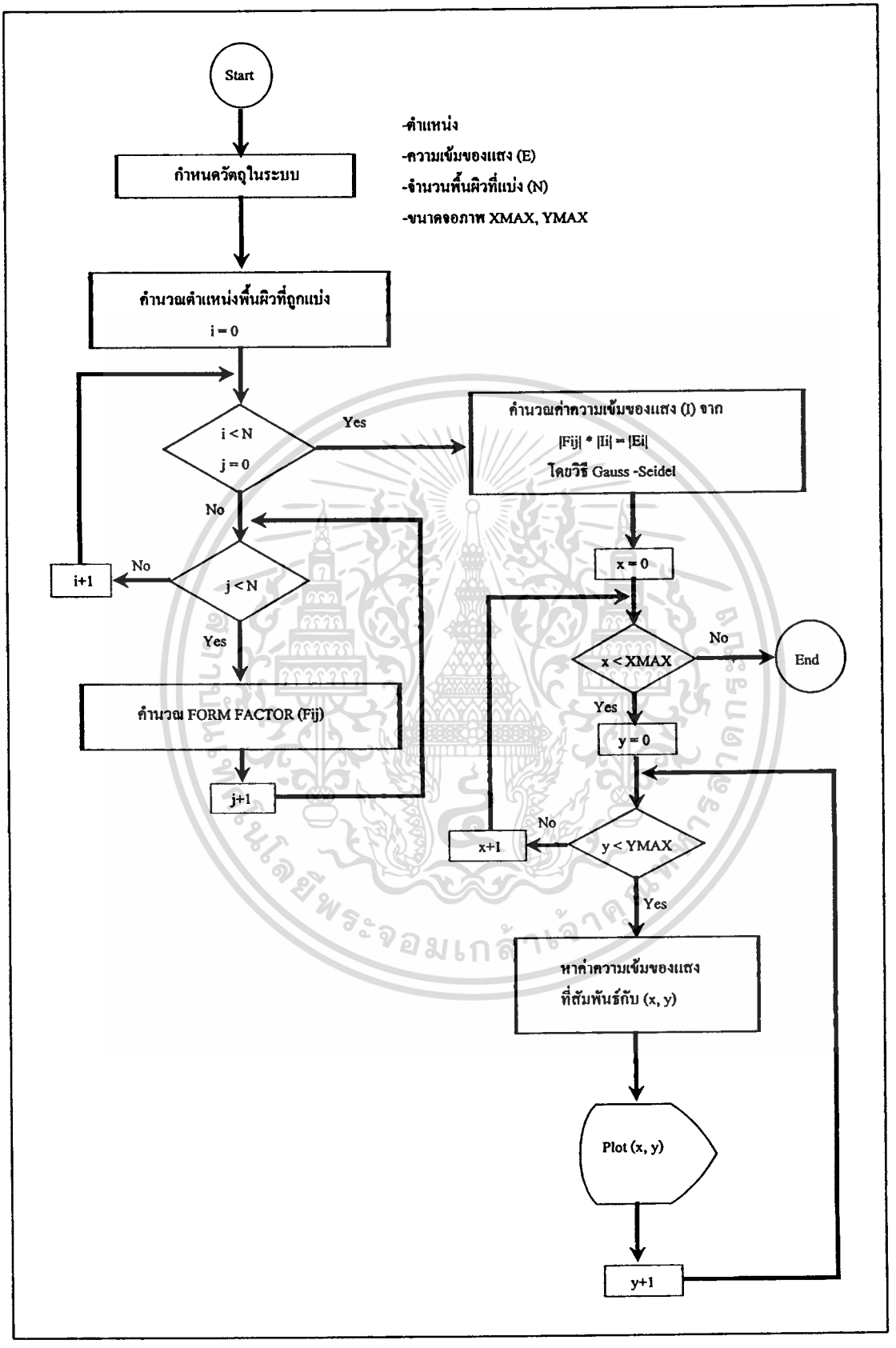
57.4



แสดง ภาพจำลองแบบ Radiosity ที่มุมมองต่างๆกัน

จากการจำลองภาพโดยวิธี Radiosity นั้น ค่าความเข้มของแสงที่คำนวณได้นั้นจะเป็นค่าความเข้มของแสงของวัตถุทั้งหมดซึ่งไม่ขึ้นกับตำแหน่งการมองของผู้สังเกต ดังนั้น ในการสร้างภาพจำลองของวัตถุที่มุมมองต่างๆจะสามารถคำนวณได้โดยไม่ต้องใช้เวลาในการคำนวณค่า Form factor ใหม่สำหรับภาพเดิม ภาพที่ 57 แสดงภาพจำลองแบบ Radiosity เมื่อแบ่งวัตถุออกเป็น 225 ช่องที่มุมมองต่างๆกัน ภาพที่ 58 แสดงวิธีการจำลองภาพแบบ Radiosity และ ตารางที่ 1 แสดงเวลาที่ใช้ในการสร้างภาพจำลอง

ภาพที่ 58



แสดง วิธีการจำลองภาพแบบ Radiosity

ตารางที่ 1

ภาพที่	จำนวนพื้นผิว	T1 (นาที)	T2 (นาที)	T3 (นาที)
55	36	0.18	0.02	2.97
56	100	0.83	0.06	3.00
57.1	225	3.48	0.18	3.07
57.2	255	-	-	3.07
57.3	255	-	-	3.12
57.4	255	-	-	3.10

แสดง เวลาที่ใช้ในการจำลองภาพแบบ Radiosity

หมายเหตุ : ภาพจำลองมีขนาด 1024 x 768 จุด

T1 = เวลาที่ใช้ในการสร้าง Form Factor

T2 = เวลาที่ใช้ในการหาค่าความเข้มของแสงโดยวิธี Gauss-Seidel

T3 = เวลาที่ใช้ในการสร้างภาพ

สรุป

จากการสร้างภาพจำลองโดยใช้วิธี Radiosity นี้ภาพที่ได้จะมีลักษณะของความเข้มของแสงไม่ต่อเนื่องกัน เนื่องจากการคำนวณค่าความเข้มของแสงนั้นเป็นไปตามลักษณะการแบ่งพื้นผิวของวัตถุที่ค่าความเข้มของแสงของพื้นผิวเล็กๆจะมีค่าเท่ากัน ซึ่งเป็นตัวกำหนดรายละเอียดของภาพจำลองและเวลาที่ใช้ในการคำนวณ[30] ดังนั้นถ้าแบ่งพื้นผิวของวัตถุออกเป็นจำนวนมาก จะทำให้ได้ภาพจำลองที่มีความละเอียดมากขึ้น แต่ก็จะใช้เวลามากขึ้นตามไปด้วยและยังมีผลต่อหน่วยความจำของเครื่อง ซึ่งทำให้เครื่องไมโครคอมพิวเตอร์ขนาดเล็กจะใช้งานไม่ได้ เนื่องจากการกำหนดวิธีการคำนวณค่าความเข้มของแสงนั้นจำเป็นต้องใช้หน่วยความจำเป็นจำนวนมาก วิธีการแก้ไขในเรื่องหน่วยความจำนั้นสามารถกระทำได้โดยการเก็บข้อมูลต่างๆที่ใช้ในการคำนวณลงในแฟ้มข้อมูล แต่วิธีการดังกล่าวนี้จะทำให้ใช้เวลาในการอ่านและเขียนข้อมูลมาก ซึ่งการการจำลองภาพแบบ Radiosity นี้สามารถที่จะพัฒนาต่อไปได้เพื่อแก้ไขความไม่ต่อเนื่องกันของค่าความเข้มของแสงและการใช้เวลาในการสร้างภาพจำลองให้น้อยลง การจำลองภาพโดยวิธีนี้จะให้ได้ผลดีในการสร้างภาพจำลองที่อยู่ในระบบปิดหรือภาพที่อยู่ภายในห้อง เนื่องในการคำนวณนั้นได้คำนึงถึงรายละเอียดของแสงต่างๆ ที่มาจากทุกทิศทาง ในบทต่อไปจะได้กล่าวถึงวิธีการจำลองภาพแบบ Ray tracing ซึ่งเป็นการจำลองภาพอีกวิธีหนึ่งที่มีหลักการต่างกัน

บทที่ 6

RAY TRACING

การจำลองภาพแบบ Ray tracing

Ray tracing เป็นวิธีทางคอมพิวเตอร์กราฟิกที่ใช้ในการจำลองภาพของวัตถุต่างๆในระบบ 3 มิติ เพื่อแสดงผลของภาพที่ได้ในระบบ 2 มิติ โดยใช้หลักการว่า ภาพใดๆที่เกิดขึ้นนั้นเป็นผลมาจากเมื่อแสงจากต้นกำเนิดแสงใดๆมาตกกระทบกับวัตถุแล้วสะท้อนมาสู่ตาของผู้สังเกต ลักษณะของภาพที่ได้จะขึ้นอยู่กับคุณสมบัติของแสงที่มาตกกระทบกับวัตถุ เช่น สีและความเข้มของแสง รวมถึงคุณสมบัติของวัตถุเอง เช่น ลักษณะความมันของพื้นผิวของวัตถุ[31] เป็นต้น

ในการจำลองภาพในระบบ 3 มิตินั้น การกำหนดค่าต่างๆจะต้องกำหนดให้อยู่ในรูปของตัวแปร 3 มิติ เช่น ลักษณะรูปร่าง ตำแหน่งของวัตถุ ตำแหน่งของต้นกำเนิดแสง ส่วนสำคัญอีกส่วนในการสร้างภาพจำลองของวัตถุที่จำเป็นต้องกำหนดคือ ตำแหน่งของผู้สังเกตหรือตำแหน่งของตา (Eye point) และระนาบของภาพ (Image plane) ดังภาพที่ 59 ตำแหน่งของผู้สังเกต จะเป็นตัวกำหนดว่าต้องการมองภาพที่ตำแหน่งไหนในระบบ ส่วนระนาบของภาพนั้นจะเป็นระนาบสมมุติเพื่อที่จะแสดงภาพจำลองที่ได้ ตำแหน่งของระนาบของภาพจะมีความสัมพันธ์กับตำแหน่งของผู้สังเกต ภาพจำลองที่เกิดขึ้นที่ระนาบของภาพจะเป็นภาพที่เกิดจาก การฉายภาพแบบ Perspective โดยจุดศูนย์กลางของการฉายภาพจะอยู่ที่ตำแหน่งของผู้สังเกต [32]

หลักการของการจำลองภาพนั้น คือ การหาค่าของสีและค่าความเข้มของแสงที่ถูกต้องสำหรับจุดใด ๆก็ตามที่เป็นจุดสังเกต วิธีการหนึ่งที่ใช้กัน คือ การหาค่าเฉลี่ยของสีจากต้นกำเนิดแสงที่มาตกกระทบที่จุดสังเกตนั้น จากวิธีการดังกล่าวนี้ การจำลองภาพ โดยวิธี Ray tracing สามารถแยกออกได้เป็น 2 วิธีด้วยกัน ตามลักษณะของการพิจารณาทิศทางของแสงที่มาตกกระทบจุดที่สังเกต[33] คือ

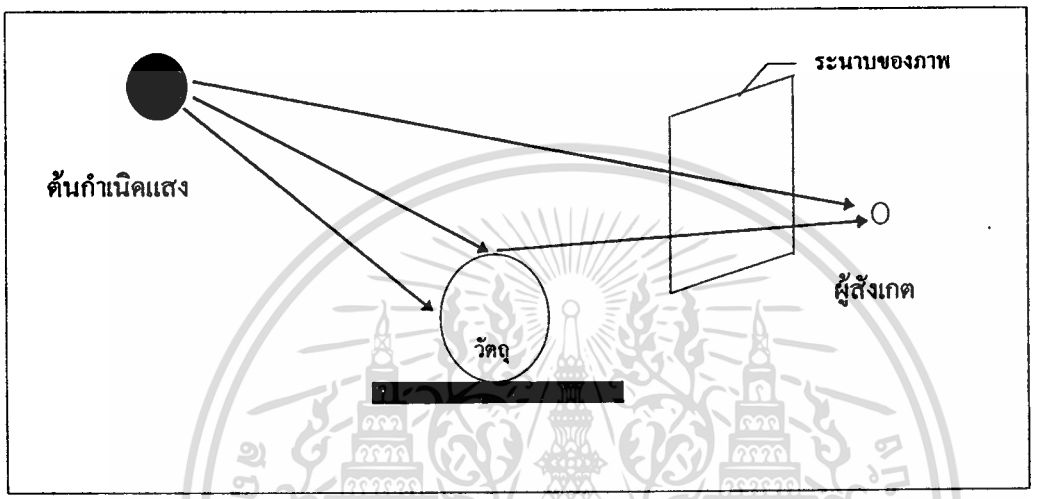
การจำลองภาพแบบ Ray tracing โดยวิธีติดตามแสงจากต้นกำเนิดแสง

วิธีการจำลองภาพแบบ Ray tracing โดยวิธีติดตามแสงจากต้นกำเนิดแสง (Forward Ray Tracing)[33] นี้ จะพิจารณาทิศทางของแสงจากต้นกำเนิดแสงในระบบก่อน ดังแสดงในภาพที่ 59

จากภาพที่ 59 ภาพของวัตถุต่าง ๆ ที่สามารถมองเห็นได้จะเป็นภาพที่เกิดจากการสะท้อนของแสงจากต้นกำเนิดแสงที่มาตกกระทบกับวัตถุแล้วสะท้อนมาสู่ตาของผู้สังเกตเท่านั้น ดังนั้นถ้ากำหนดว่าแสงจากต้นกำเนิดแสงใด ๆ เป็นอนุภาคหนึ่งที่สามารถเคลื่อนที่ได้ในที่ว่าง (space) [34] การจำลองภาพแบบ Ray tracing โดยวิธีติดตามแสงจากต้นกำเนิดแสง จะเริ่มพิจารณาจากอนุภาคต่างๆเหล่านี้ที่ถูกส่งมาจากต้นกำเนิดแสง และภาพที่เกิดบนจอภาพหรือระนาบของภาพจะเกิดจากการที่อนุภาคเหล่านี้ตกกระทบกับวัตถุแล้วสะท้อนมาสู่ผู้สังเกต การเริ่มต้นพิจารณาจากต้นกำเนิด

แสงนี้จะทำให้ใช้เวลามาก[35,36] เนื่องจากไม่สามารถที่จะจำกัดจำนวนการพิจารณาของอนุภาคต่างๆ ที่มาจากต้นกำเนิดแสงและตกกระทบกับวัตถุแล้วสะท้อนมาสู่ผู้สังเกตได้ อนุภาคต่างๆที่ถูกส่งออกมาจากต้นกำเนิดแสงนั้นจะมีเพียงส่วนน้อยเท่านั้นที่จะตกกระทบกับวัตถุและสะท้อนกลับมาสู่ผู้สังเกต อนุภาคที่ถูกส่งออกมาส่วนมากจะเป็นอนุภาคที่ไม่มีผลต่อผู้สังเกตเลย ดังนั้นการที่จะพิจารณาทุกๆอนุภาคที่ส่งออกมาจากต้นกำเนิดแสงจึงทำให้เสียเวลามาก [37,38]

ภาพที่ 59



แสดงการจำลองภาพแบบ Ray tracing โดยวิธีติดตามแสงจากต้นกำเนิดแสง

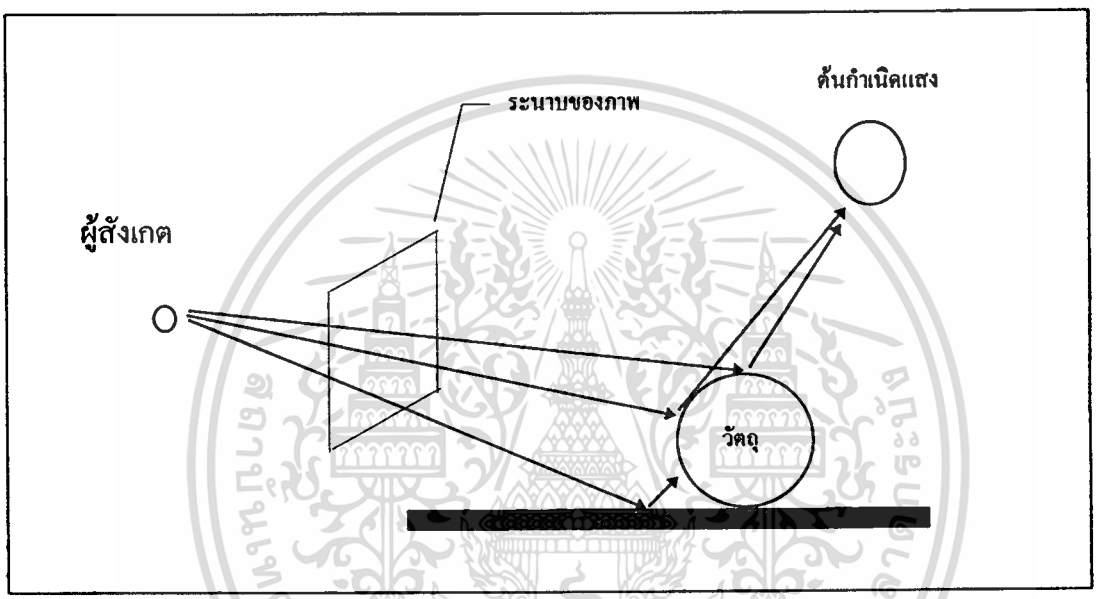
การจำลองภาพแบบ Ray tracing โดยวิธีติดตามทิศทางการมองของผู้สังเกต

เนื่องจากการจำลองภาพแบบ Ray tracing โดยวิธีติดตามทิศทางการมองของผู้สังเกตเป็นการจำลองภาพโดยอาศัยคุณสมบัติของการเกิดภาพจากการมองเห็นภาพจริงคือ แสงจากต้นกำเนิดแสงมาตกกระทบกับวัตถุแล้วสะท้อนมาสู่ผู้สังเกต[39] แต่ในการจำลองภาพแบบ Ray tracing โดยวิธีติดตามทิศทางการมองของผู้สังเกต (Backward Ray tracing) [33] เป็นการจำลองภาพโดยอาศัยวิธีการสมมุติ โดยการเริ่มต้นพิจารณาจากตำแหน่งของผู้สังเกตไปสู่ต้นกำเนิดแสง โดยอาศัยขนาดของระนาบของภาพเป็นตัวช่วยในการสร้างภาพ เนื่องจากที่ระนาบของภาพจะเป็นส่วนที่แสดงภาพจำลองที่ได้ ดังนั้นบริเวณภายนอกของระนาบของภาพจะเป็นส่วนที่ไม่มีส่วนเกี่ยวข้องกับภาพจำลองเลย จากหลักการนี้ทำให้สามารถพิจารณาเฉพาะส่วนของแสงที่สะท้อนจากวัตถุผ่านระนาบของภาพเท่านั้น[40]

การสร้างภาพจำลองแบบ Ray tracing โดยวิธีติดตามทิศทางการมองของผู้สังเกตนี้เริ่มจากการสร้างเวกเตอร์จากตำแหน่งของผู้สังเกตผ่านทุกๆจุดบนระนาบของภาพ ดังแสดงในภาพที่ 60 ถ้าเวกเตอร์จากตำแหน่งของผู้สังเกตผ่านตำแหน่งใดๆบนระนาบของภาพแล้วตกกระทบกับวัตถุแสดงว่าที่ตำแหน่งนี้บนระนาบของภาพจะสามารถมองเห็นภาพของวัตถุที่ตำแหน่งที่ตกกระทบได้

จากนั้นจึงหาค่าความเข้มของแสงและสีที่ตำแหน่งตกกระทบนั้น[41] ขั้นตอนการสร้างภาพจำลองแบบ Ray tracing โดยวิธีติดตามทิศทางการมองของผู้สังเกตนั้นแสดงในภาพที่ 61 และที่ตำแหน่งของจุดตกกระทบจำเป็นที่จะต้องคำนวณหาจุดตกกระทบกับวัตถุจุดแรก (first hit point) [42] โดยอาศัยเวลาที่ใช้ในการตกกระทบกับวัตถุต่างๆในระบบเป็นตัวตัดสิน วัตถุในระบบที่มีเวลาตกกระทบน้อยที่สุดจะเป็นวัตถุแรกที่เวกเตอร์นี้ไปตกกระทบ เนื่องจากผู้สังเกตจะเห็นวัตถุที่อยู่ใกล้จุดสังเกตมากที่สุดเสมอ

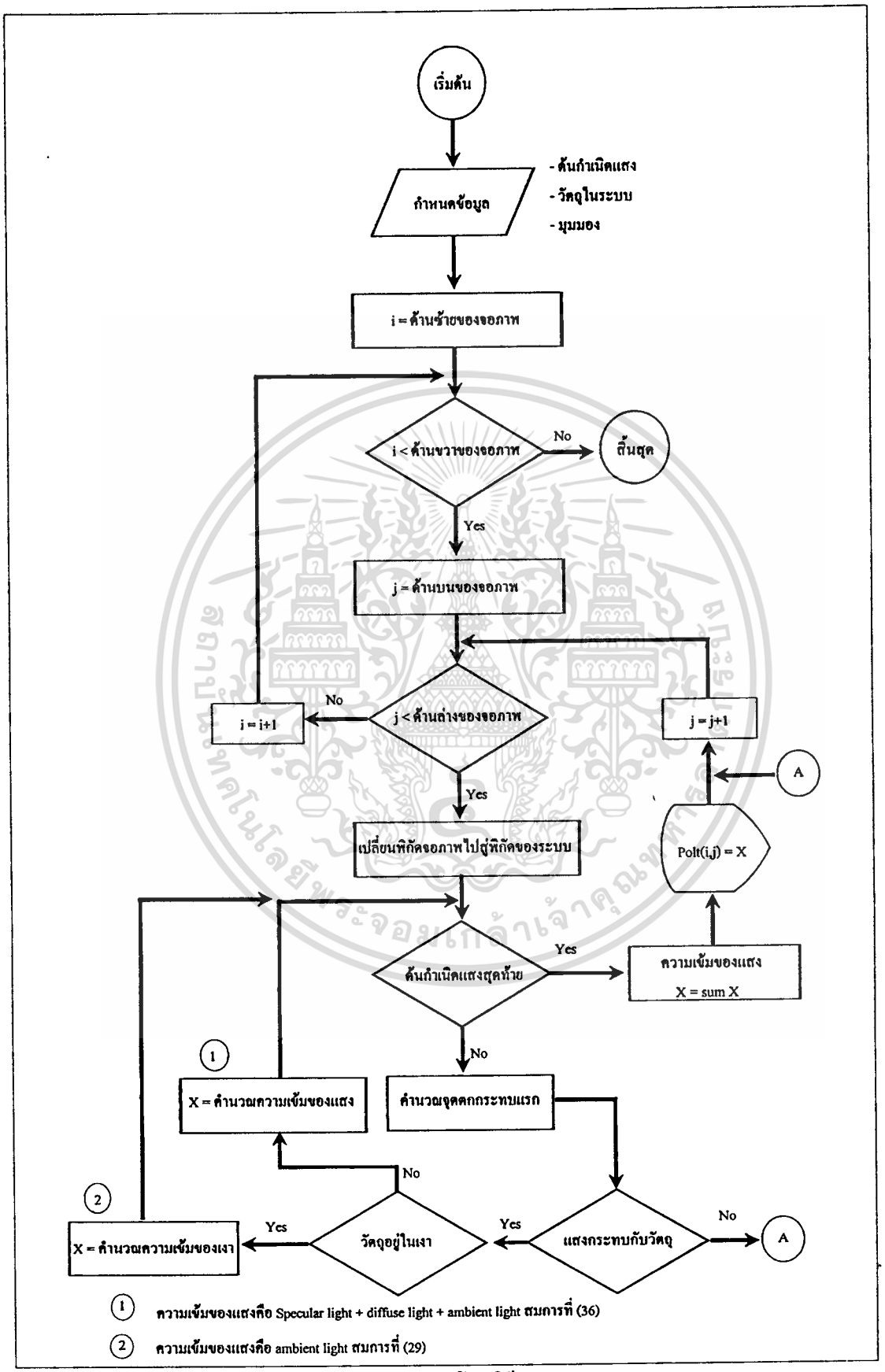
ภาพที่ 60



แสดง การจำลองภาพแบบ Ray tracing โดยวิธีติดตามทิศทางการมองของผู้สังเกต

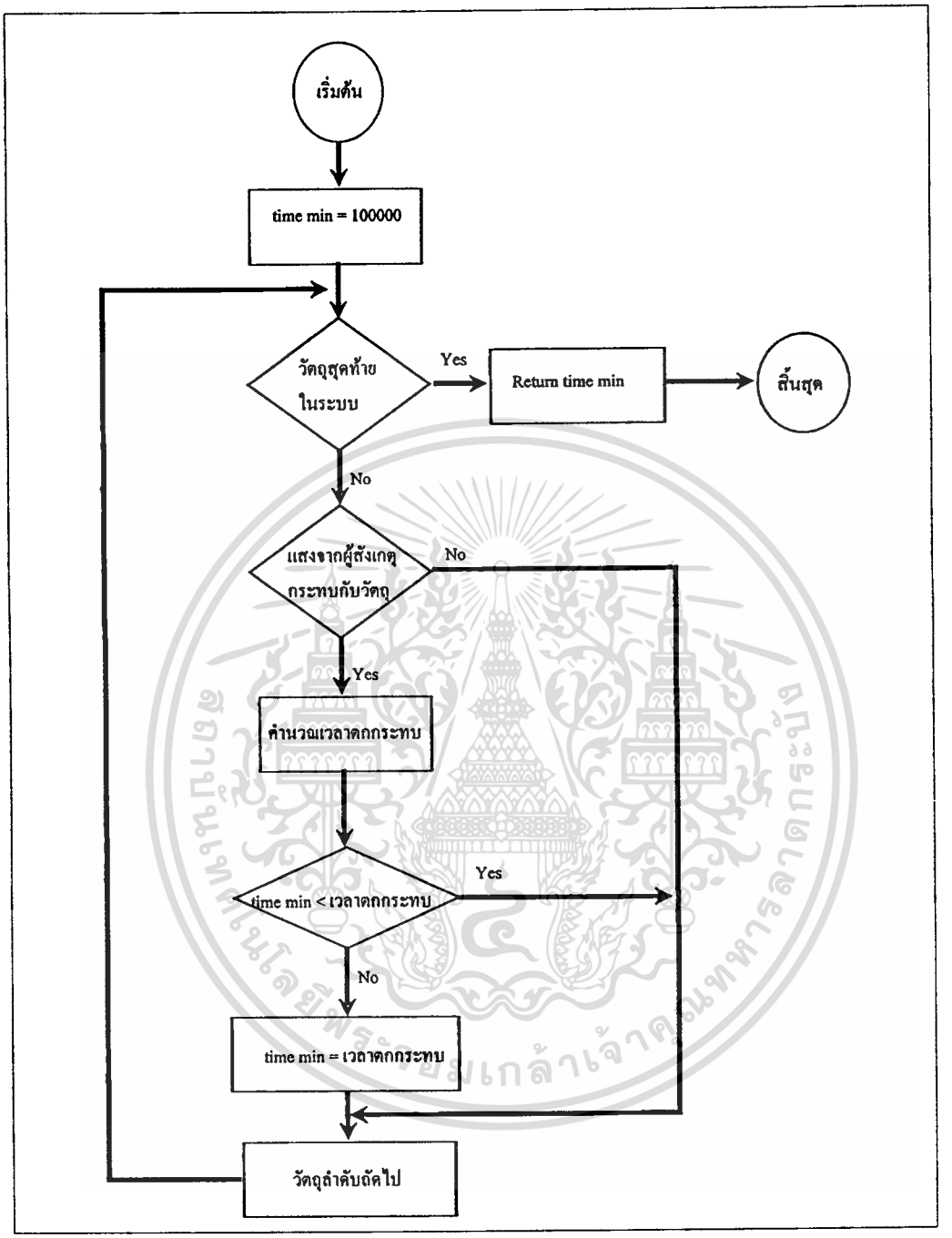
การหาค่าความเข้มของแสง[43] ที่จุดตกกระทบจุดแรกนี้สามารถหาได้โดยวิธี Phong Shading โดยสร้างเวกเตอร์สมมุติที่มีทิศทางจากจุดตกกระทบไปสู่ตำแหน่งของต้นกำเนิดแสง เรียกลำแสงลำดับที่สอง (Secondary ray) จากนั้นจะต้องทดสอบว่า ถ้าแสงลำดับที่สองไม่มีการชนกับวัตถุอื่นๆ ในระบบ แสดงว่าที่จุดตกกระทบแรกนี้ได้รับแสงจากต้นกำเนิดแสงโดยตรง ถ้าแสงลำดับที่สอง มีการชนกับวัตถุในระบบ แสดงว่าที่จุดตกกระทบแรกนี้จะอยู่ในบริเวณส่วนที่เป็นเงา (Shadow) ของวัตถุอื่นๆ [44] ภาพที่ 62 และภาพที่ 63 แสดงขั้นตอนการหาจุดตกกระทบแรกและการหาค่าความเข้มของแสงที่จุดตกกระทบแรก ส่วนภาพที่ 64, 65 และ 66 แสดงภาพจำลองที่ได้เมื่อมีต้นกำเนิดแสงจุดเดียว, ต้นกำเนิดแสงจุดเดียวและการฉายภาพที่พื้นผิวของวัตถุ (texture mapping) (รายละเอียดในภาคผนวก ก และ ภาคผนวก ข) และ ต้นกำเนิดแสง 3 จุด ตามลำดับ

ภาพที่ 61



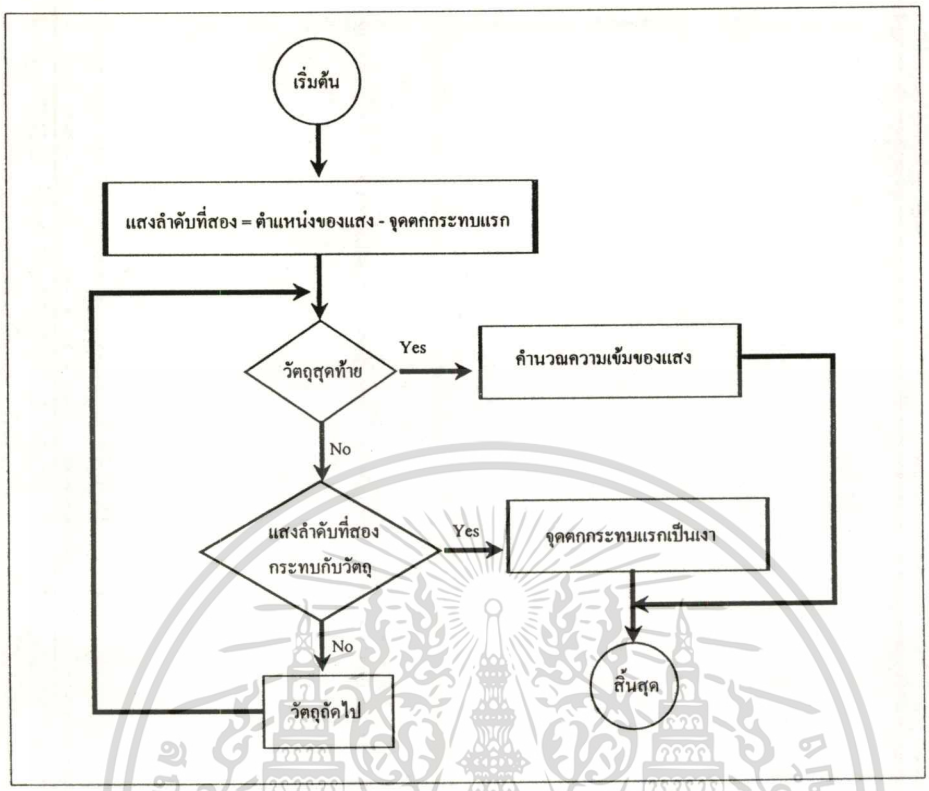
แสดง การจำลองภาพโดยวิธี Ray tracing

ภาพที่ 62



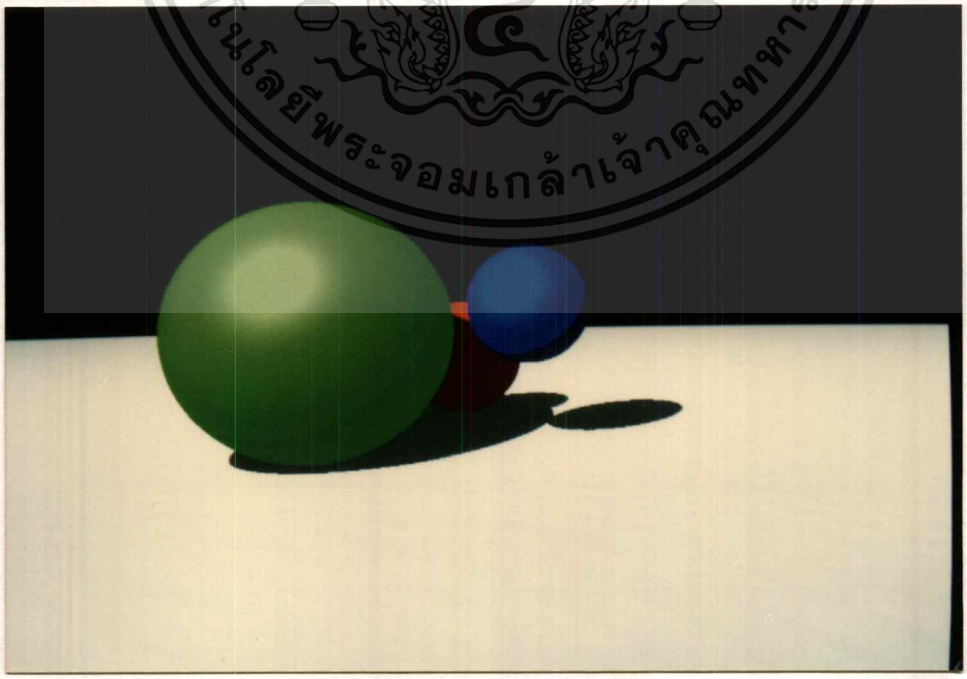
แสดง วิธีการหาจุดตกกระทบแรก

ภาพที่ 63



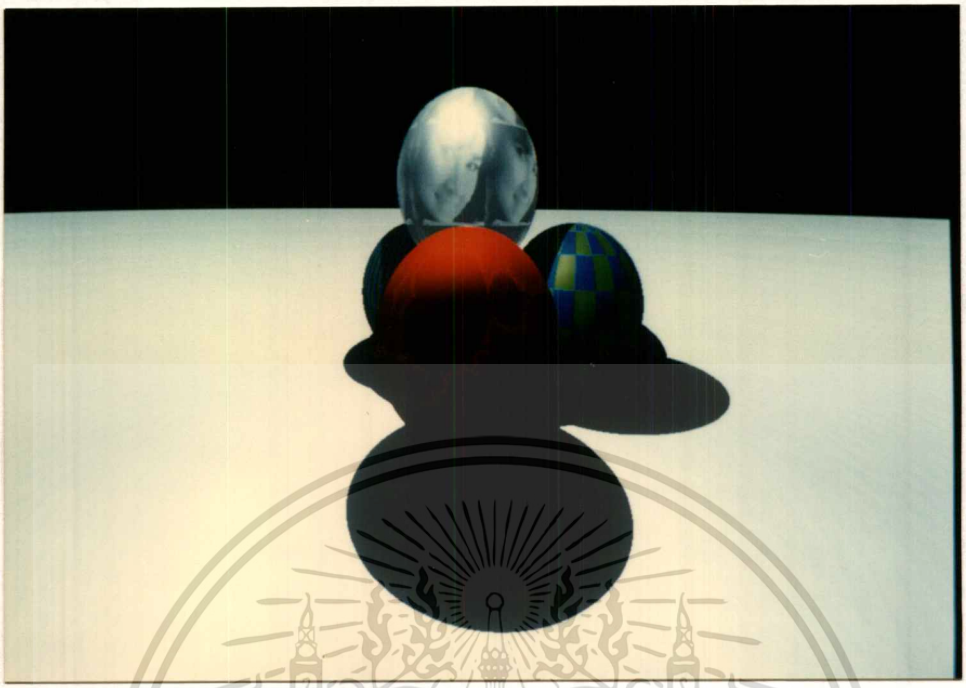
แสดง วิธีการหาค่าความเข้มที่จุดตกกระทบแรก

ภาพที่ 64



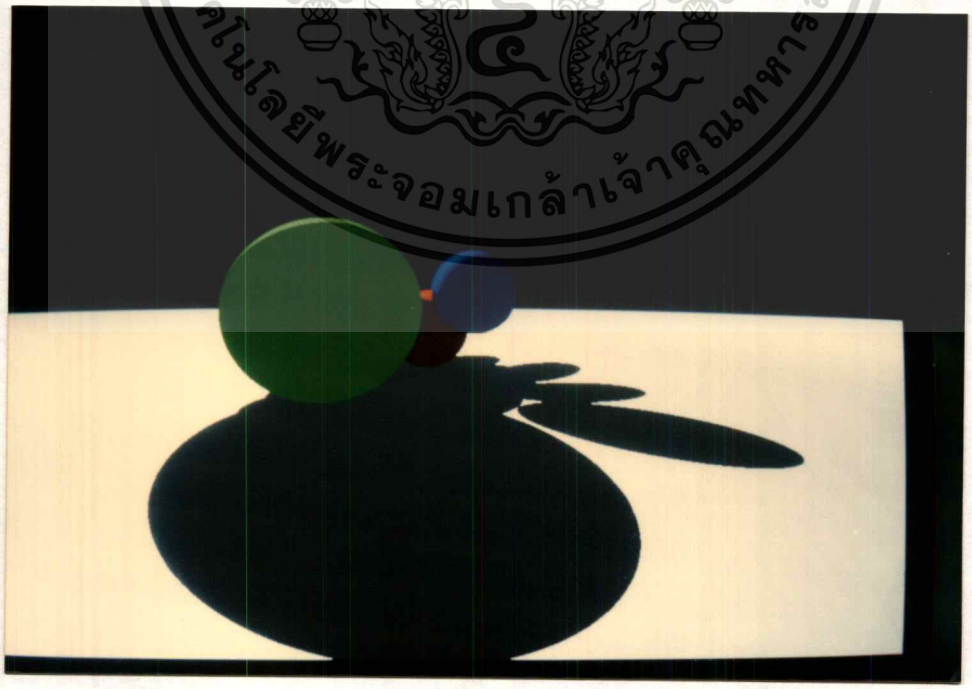
แสดง ภาพจำลองโดยวิธี Ray tracing เมื่อมีต้นกำเนิดแสงจุดเดียว

ภาพที่ 65



แสดงภาพจำลองโดยวิธี Ray tracing เมื่อมีต้นกำเนิดแสงจุดเดียวและการฉายภาพที่วัตถุ

ภาพที่ 66



แสดง ภาพจำลองโดยวิธี Ray tracing เมื่อมีต้นกำเนิดแสง 3 จุด

การชนของแสงและวัตถุ

ในการจำลองภาพโดยวิธี Ray tracing นั้นการคำนวณหาตำแหน่งของการชนกันของลำแสงและวัตถุเป็นส่วนที่สำคัญมาก เนื่องจากเป็นตำแหน่งที่ใช้ในการหาค่าความเข้มของแสงที่มาตกกระทบ เพื่อนำมาสร้างภาพจำลองของวัตถุ วัตถุต่างๆที่ถูกสร้างขึ้นในระบบนั้นมีรูปร่างที่ต่างกันดังนั้น การหาตำแหน่งที่เกิดการชนกันระหว่างแสงและวัตถุจึงมีวิธีการคำนวณหาที่แตกต่างกัน[33]

การชนกันของแสงและระนาบ

ถ้าสมการของระนาบ คือ

$$Ax + By + Cz + D = 0 \qquad (50)$$

โดยที่ $A^2 + B^2 + C^2 = 1$ และ D คือระยะจากจุดกำเนิดของระบบ [0, 0, 0] ถึงระนาบและเวกเตอร์หนึ่งหน่วยที่ตั้งฉากกับระนาบคือ

$$P_n = \begin{bmatrix} A & B & C \end{bmatrix} \qquad (51)$$

และตำแหน่งของต้นกำเนิดแสงสมมุติคือ

$$R_0 = \begin{bmatrix} X_0 & Y_0 & Z_0 \end{bmatrix} \qquad (52)$$

และทิศทางของแสงสมมุติคือ

$$R_d = \begin{bmatrix} X_d & Y_d & Z_d \end{bmatrix} \qquad (53)$$

เมื่อ $X_d^2 + Y_d^2 + Z_d^2 = 1$

สมการของเวกเตอร์แสงสมมุติที่เวลาใดๆคือ

$$R(t) = R_0 + R_d t \qquad (54)$$

เวกเตอร์ของแสงมีจุดเริ่มต้นที่ R_0 ที่ $t = 0$ และเคลื่อนที่ไปในทิศทาง R_d เมื่อเวลา t ที่มากกว่า 0 เวกเตอร์ของแสงจะเคลื่อนที่ผ่านตำแหน่ง R(t)

การคำนวณหาจุดตัดระหว่างแสงและระนาบสามารถหาได้จากเวลาที่ใช้ในการชนกัน ถ้าเวลาที่ใช้ในการชนกันมีค่าน้อยกว่า 0 แสดงให้เห็นว่าแสงและระนาบมีการชนกันก่อนจะถึงเวลาที่ใช้ในการพิจารณา เมื่อ t มีค่ามากกว่า 0 แสดงให้เห็นว่าแสงและระนาบมีการชนกันเมื่อเวลาผ่านไป

จากสมการที่ (50) และ (54) สามาหาค่าของ t ได้โดย

$$t = \frac{-(AX_0 + BY_0 + CZ_0 + D)}{AX_d + BY_d + CZ_d} \qquad (55)$$

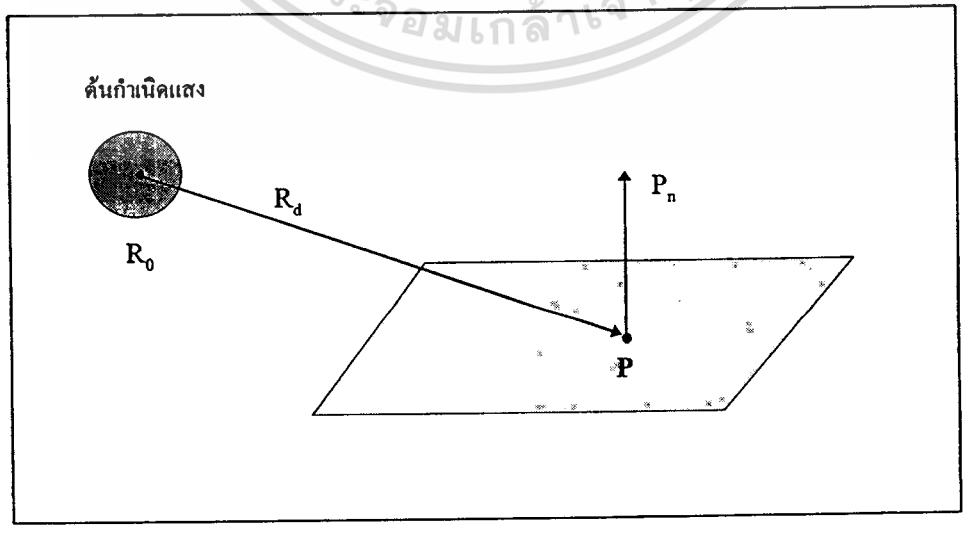
หรือ $$t = \frac{-(P_n.R_0 + D)}{P_n.R_d} \qquad (56)$$

เมื่อ $P_n.R_d$ มีค่าเท่ากับ 0 แสดงว่า แสงและระนาบมีทิศทางที่ขนานกัน ถ้า $P_n.R_d$ มีค่ามากกว่า 0 แสดงว่าเวกเตอร์หนึ่งหน่วยที่ตั้งฉากกับระนาบมีทิศทางที่ชี้ออกจากต้นกำเนิดแสง ในกรณีที่พิจารณาเพียงด้านเดียวของระนาบและจุดตัดกันของแสงและระนาบ คือ

$$R_i = \left[X_0 + X_d t \quad Y_0 + Y_d t \quad Z_0 + Z_d t\right] \qquad (57)$$

ส่วนการชนกันของแสงและระนาบแสดงในภาพที่ 67

ภาพที่ 67



แสดง การชนกันของแสงและระนาบ

การชนกันของแสงและรูปหลายเหลี่ยม

การคำนวณหาตำแหน่งของการชนกันของแสงและรูปหลายเหลี่ยมนั้นสามารถหาได้จากการชนกันของแสงและระนาบ โดยสร้างระนาบใหม่ที่ครอบคลุมรูปหลายเหลี่ยมที่ต้องการจากระนาบที่สร้าง วิธีคำนวณหาจุดตัดของแสงและระนาบขั้นต่อไป คือ การหาว่าจุดที่ตัดนั้นอยู่ภายในรูปหลายเหลี่ยมหรือไม่

กำหนดตำแหน่งของรูปหลายเหลี่ยมเป็นชุดของจุด จำนวณ N จุด

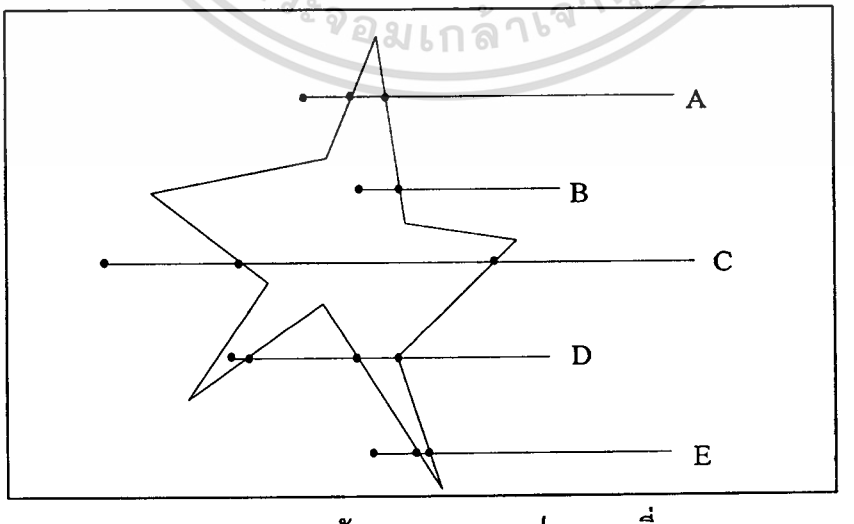
$$\left.\begin{array}{l} G_n = [X_n \quad Y_n \quad Z_n] \\ n = 0,1,2,\ldots\ldots,N-1 \end{array}\right\} \qquad (58)$$

ที่อยู่ภายในระนาบใดๆและเวกเตอร์ที่ตั้งฉากกับระนาบดังสมการที่ (50) และ (51) จากหัวข้อการชนของแสงและระนาบ ถ้าตำแหน่งที่ชนกันของแสงและระนาบคือ

$$R_i = [X_i \quad Y_i \quad Z_i] \qquad (59)$$

การทดสอบว่าตำแหน่งของการชนกันอยู่ภายในรูปหลายเหลี่ยมหรือไม่ สามารถกระทำได้โดยสร้างเส้นตรงจากตำแหน่งที่ชนกันของแสงและระนาบผ่านรูปหลายเหลี่ยม ถ้าเส้นตรงที่สร้างตัดเส้นรอบรูปหลายเหลี่ยมเป็นเลขคี่แสดงว่าตำแหน่งการชนกันของแสงและระนาบอยู่ภายในรูปหลายเหลี่ยมดังแสดงในภาพที่ 68

ภาพที่ 68



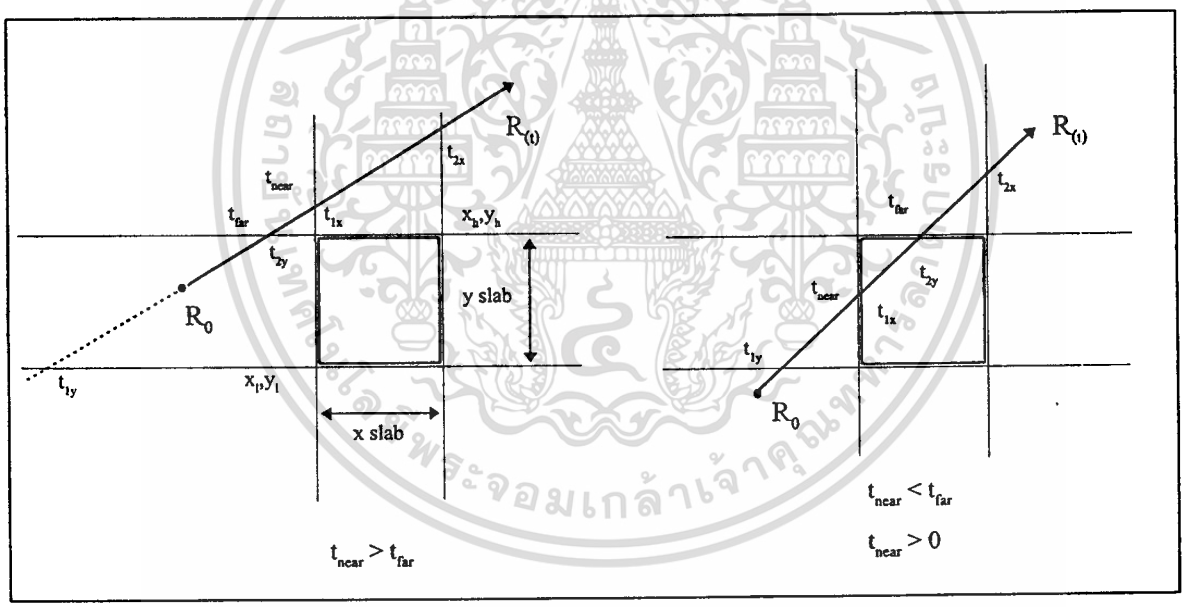
แสดง การชนกันของแสงและรูปหลายเหลี่ยม

จากภาพที่ 68 เส้นตรง A ผ่านด้านต่างๆ ของรูปหลายเหลี่ยมเป็นจำนวณ 2 ด้าน แสดงว่าตำแหน่งของการชนอยู่ภายนอกรูปหลายเหลี่ยม เส้นตรง B ผ่านด้านต่างๆ ของรูปหลายเหลี่ยมเป็นจำนวน 1 ด้าน ดังนั้นตำแหน่งของการชนจึงอยู่ภายในรูปหลายเหลี่ยมสำหรับเส้นตรง D จะตัดผ่านด้านต่างๆของรูปหลายเหลี่ยมเป็นจำนวน 4 ด้าน แต่จุดตัดสุดท้ายเป็นตำแหน่งเดียวกัน ดังนั้นจุดตัดผ่านรูปหลายเหลี่ยมจึงมีเพียง 3 ด้านเท่านั้น จึงทำให้จุดของการชนจึงอยู่ในรูปหลายเหลี่ยม

การชนของแสงและกล่องสี่เหลี่ยมลูกบาศก์

การคำนวณหาตำแหน่งตกกระทบของแสงและกล่องสี่เหลี่ยมลูกบาศก์ มีวิธีการคำนวณคล้ายกับ การหาตำแหน่งการชนกันของแสงและรูปหลายเหลี่ยมโดยการสร้างระนาบที่ผิวของกล่องสี่เหลี่ยมลูกบาศก์ จากนั้นจึงหาตำแหน่งตัดของแสงและระนาบในแนวแกนต่างๆทีละด้านเหมือนกับการหาจุดตัดของแสงกับระนาบที่ขนานกัน 2 ระนาบ (slab) ดังภาพที่ 69

ภาพที่ 69



แสดง การชนกันของแสงและกล่องสี่เหลี่ยมลูกบาศก์

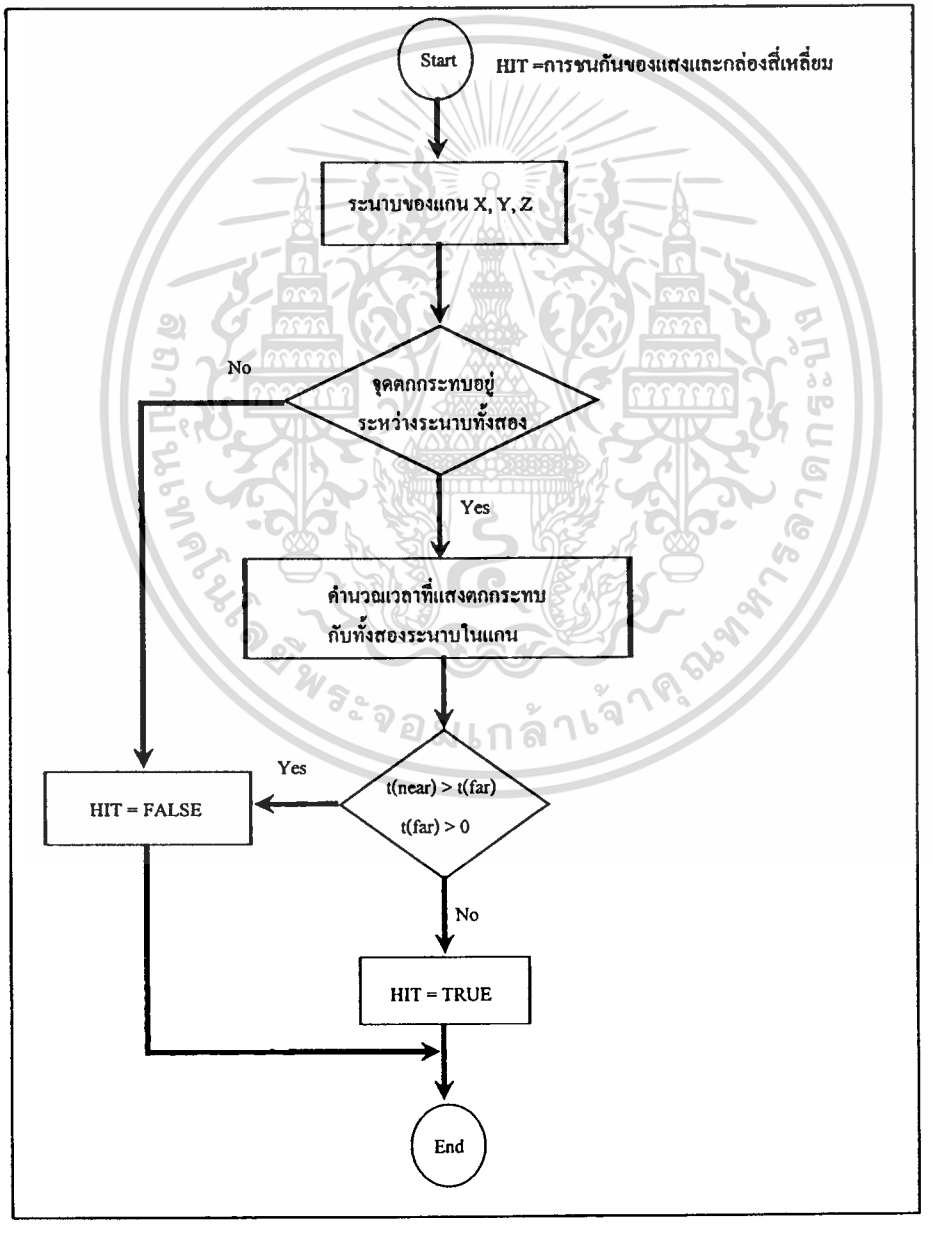
กำหนดตำแหน่งของกล่องสี่เหลี่ยมโดย จุด 2 จุด คือ จุดต่ำสุด B_l และจุดสูงสุด B_h

$$\left.\begin{aligned} B_l &= [X_l \quad Y_l \quad Z_l] \\ B_h &= [X_h \quad Y_h \quad Z_h] \end{aligned}\right\} \qquad (60)$$

ตำแหน่งของต้นกำเนิดแสง,ทิศทางของแสง และสมการของแสงที่เวลาใดๆ คือ สมการที่ (52), (53) และ (54) ตามลำดับ

เวลาที่ใช้ในการชนกันของแสงกับระนาบทั้งสอง จะมีสองค่าด้วยกัน คือ เวลาที่ใช้ในการชนกันของแสงกับระนาบที่อยู่ใกล้กับต้นกำเนิดแสง (t_{near}) และเวลาที่ใช้ในการชนกันของแสงกับระนาบที่อยู่ไกลจากต้นกำเนิดแสง (t_{far}) ถ้าค่าสูงสุดของ t_{near} มีค่ามากกว่าค่าต่ำสุดของ t_{far} แสดงว่าแสงนั้นไม่มีการชนกันกับระนาบ วิธีการหาเวลาที่ใช้ในการชนกันของแสงและกล่องสี่เหลี่ยมแสดงในภาพที่ 70 ส่วนตำแหน่งการชนสามารถหาได้จากหัวข้อที่ 6.2.1

ภาพที่ 70



แสดง วิธีการหาเวลาที่ใช้ในการชนกันของแสงและกล่องสี่เหลี่ยม

การชนของแสงและทรงกลม

กำหนดทรงกลมมีรัศมี S_r และจุดศูนย์กลางอยู่ที่

$$S_c = [X_c \quad Y_c \quad Z_c] \qquad (61)$$

พื้นผิวของทรงกลมจะอยู่ที่ $[X_s \quad Y_s \quad Z_s]$ เมื่อ

$$(X_s - X_c)^2 + (Y_s - Y_c)^2 + (Z_s - Z_c)^2 = S_r^2 \qquad (62)$$

และจากสมการของแสงที่ (52), (53) และ (54) ดังนั้นเวลาที่เวกเตอร์ของแสงใช้ในการเดินทางมาชนกับพื้นผิวของทรงกลม สามารถหาได้จากการแทนค่า $X_s = X_0+X_dt$, $Y_s = Y_0+Y_dt$ และ $Z_s = Z_0+Z_dt$ ในสมการที่ (62) จะได้

$$At^2 + Bt + C = 0 \qquad (63)$$

เมื่อ

$$A = X_d^2 + Y_d^2 + Z_d^2 = 1$$
$$B = 2(X_d(X_0 - X_c) + Y_d(Y_0 - Y_c) + Z_d(Z_0 - Z_c))$$
$$C = (X_0 - X_c)^2 + (Y_0 - Y_c)^2 + (Z_0 - Z_c)^2 - S_r^2$$

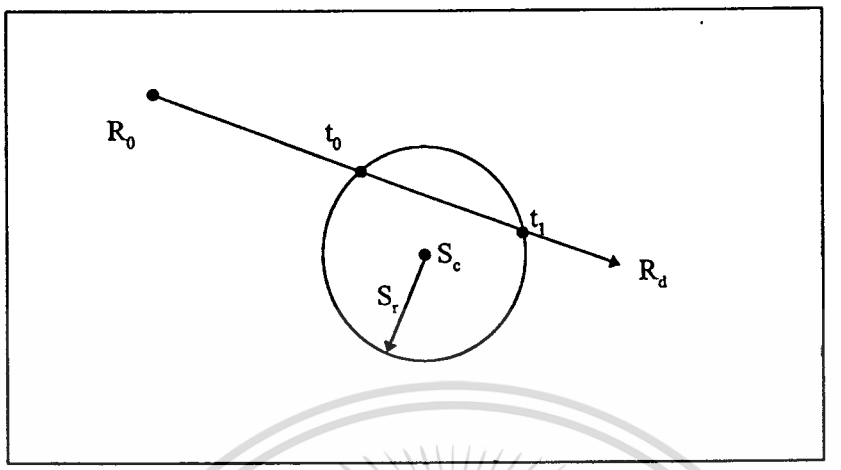
จากสมการที่ (63) คำตอบของสมการจะมีสองค่าด้วยกัน คือ

$$t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \qquad (64)$$

จากสมการที่ (64) ถ้าส่วนของสมการในส่วนรากที่สองมีค่าเป็นลบ หรือสมการไม่สามารถหาค่าได้ หรือถ้า t มีค่าน้อยกว่า 0 แสดงว่าแสงไม่มีการชนกับทรงกลม ส่วนค่า t ที่มีค่ามากกว่า 0 และมีค่าน้อยที่สุดจะหมายถึงเวลาที่แสงใช้ในการเดินทางไปชนกับทรงกลมจุดแรก ภาพที่ 71 แสดงการชนกันของแสงและทรงกลม และตำแหน่งที่แสงชนกับทรงกลม คือ

$$R_i = [X_i \quad Y_i \quad Z_i] = [X_0 + X_dt \quad Y_0 + Y_dt \quad Z_0 + Z_dt] \qquad (65)$$

ภาพที่ 71



แสดง การชนของแสงและทรงกลม

การชนของแสงและวัตถุแบบ QUADRIC

พื้นผิวแบบ Quadric (Quadric surface) เป็นพื้นผิวที่สามารถอธิบายได้โดยสมการกำลังสอง สมการทั่วไปของพื้นผิวแบบ Quadric สามารถเขียนได้รูป

$$F(x, y, z) \equiv Ax^2 + 2Bxy + 2Cxz + 2Dx + Ey^2 + 2Fyz + 2Gy + Hz^2 + 2Iz + J \qquad (66)$$

หรือ

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} A & B & C & D \\ B & E & F & G \\ C & F & H & I \\ D & G & I & J \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0 \qquad (67)$$

สมการที่ (66) เป็นสมการทั่วไปที่สามารถใช้แทนสมการของวัตถุต่าง ๆ ได้ เช่น

ทรงกลม

$$(x-m)^2 + (y-n)^2 + (z-o)^2 = r^2 \qquad (68)$$

เมื่อทรงกลมมีรัศมี r และจุดศูนย์กลางที่ (m, n, o)

ทรงกระบอก

$$(x-m)^2+(y-n)^2 = r^2 \qquad (69)$$

Ellipsoid

$$\frac{(x-m)^2}{\alpha^2}+\frac{(y-n)^2}{\beta^2}+\frac{(z-o)^2}{\gamma^2}-1 = 0 \qquad (70)$$

เมื่อ α, β และ γ คือ semi axis

Elliptic Paraboloid

$$\frac{(x-m)^2}{\alpha^2}+\frac{(y-n)^2}{\beta^2}-z+o = 0 \qquad (71)$$

Hyperboloid

$$\frac{(x-m)^2}{\alpha^2}+\frac{(y-n)^2}{\beta^2}-\frac{(z-o)^2}{\gamma^2}-1 = 0 \qquad (72)$$

จากสมการของแสงที่ (52), (53) และ (54) สามารถคำนวณหาเวลาที่ใช้ในการเดินทางมาตกกระทบกับวัตถุ คือ

$$t = \frac{-B_q \pm \sqrt{B_q^2 - 4A_qC_q}}{2A_q} \qquad (73)$$

โดยที่ $A_q = Ax_d^2 + 2Bx_dy_d + 2Cx_dz_d + Ey_d^2 + 2Fy_dz_d + Hz_d^2$

$B_q = 2(Ax_0x_d + B(x_0y_d + x_dy_0) + C(x_0z_d + x_dz_0) +$
$Dx_d + Ey_0y_d + F(y_0z_d + y_dz_0) + Gy_d + Hz_0z_d + Iz_d)$

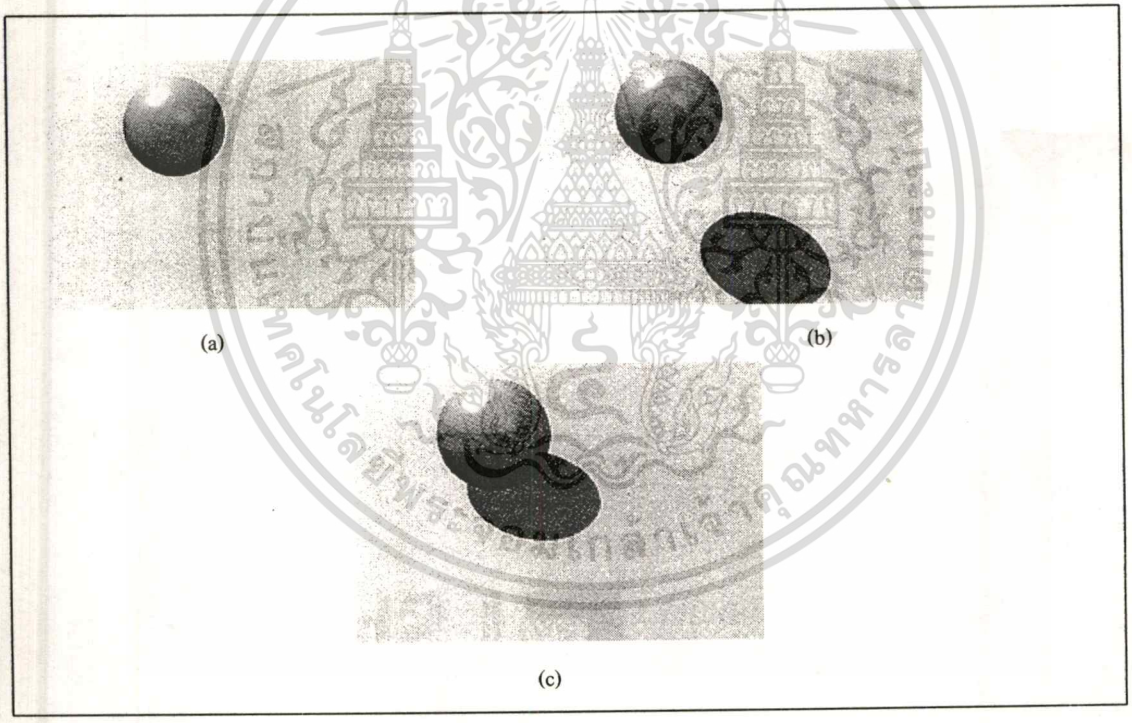
$C_q = Ax_0^2 + 2Bx_0y_0 + 2Cx_0z_0 + 2Dx_0 + Ey_0^2 +$
$2Fy_0z_0 + 2Gy_0 + Hz_0^2 + 2Iz_0 + J$

ถ้า $A_q \neq 0$ และ $B_q^2 - 4A_qC_q < 0$ จะไม่มีการชนกันระหว่างแสงกับวัตถุ ส่วนค่าของ t ที่น้อยที่สุดและมากกว่าศูนย์ จะเป็นเวลาที่แสงใช้เดินทางไปตกกระทบกับวัตถุเป็นตำแหน่งแรก

เงา

เงาของวัตถุเกิดจากการที่มีวัตถุใดๆ ก็ตามมาบังวัตถุนั้นจากต้นกำเนิดแสง แล้วทำให้ไม่ได้รับแสงจากต้นกำเนิดแสง เงาที่เกิดขึ้นบนวัตถุสามารถที่จะแยกออกเป็น 2 ส่วนด้วยกัน คือ เงาที่เกิดโดยตัวเอง เงาชนิดนี้เกิดการบังของแสงโดยตัวของวัตถุเอง และเงาที่เกิดจากวัตถุอื่น เป็นเงาที่เกิดจากการที่มีวัตถุอื่นมาบังวัตถุนั้นไม่ให้ได้รับแสงจากต้นกำเนิดแสง [45] การเกิดเงาของวัตถุนั้นทำให้ทราบถึงความสัมพันธ์ระหว่างวัตถุที่อยู่ในระบบเดียวกันว่ามีตำแหน่งอ้างอิงถึงกันอย่างไร[46] ภาพที่ 72 แสดงถึงความสัมพันธ์ของวัตถุในระบบ ภาพที่ 72a แสดงให้เห็นว่าถ้าไม่มีเงาของวัตถุจะทำให้ไม่ทราบได้เลยว่า รูปทรงกลม และระนาบมีความสัมพันธ์กันแบบใด ภาพที่ 72b และภาพที่ 72c แสดงให้ทราบว่าทรงกลมอยู่ในตำแหน่งที่วางอยู่บนระนาบและอยู่สูงกว่าระนาบตามลำดับ

ภาพที่ 72

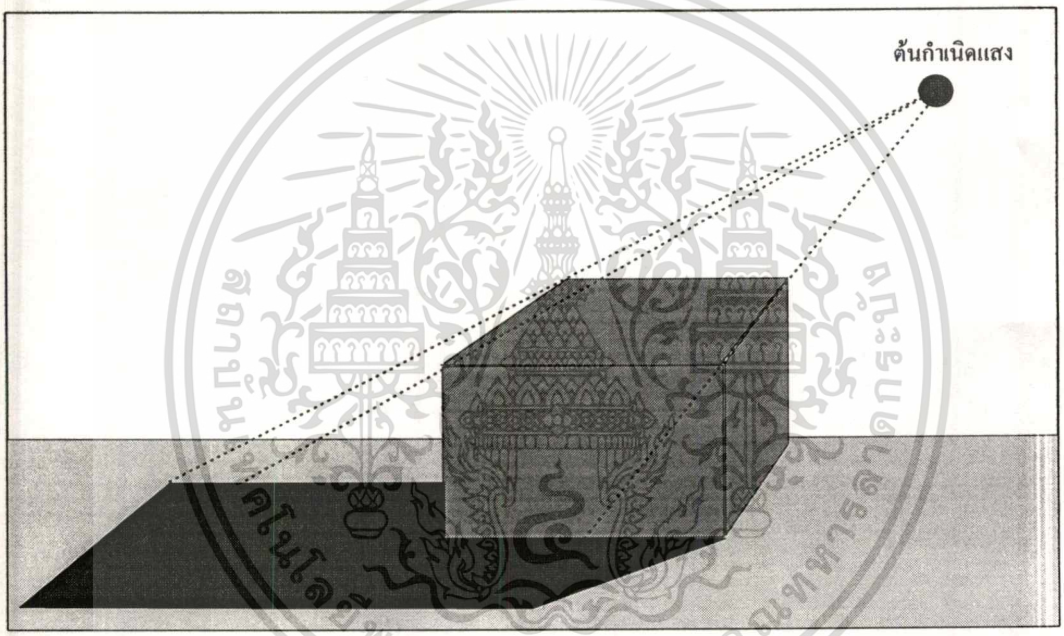


แสดง เงาของวัตถุ

สำหรับต้นกำเนิดแสงแบบจุด และลักษณะของเงาที่เกิดขึ้นนั้นเห็นได้ชัดว่าส่วนที่ต่อกันระหว่างเงาและส่วนที่ไม่ใช่เงาจะมีค่าความเข้มของแสงที่แตกต่างกันอย่างชัดเจน เนื่องจากคุณสมบัติของต้นกำเนิดแสงแบบจุดและวิธีพื้นฐานของการหาส่วนของเงาบนวัตถุดังภาพที่ 72b และ 72c แสดงถึงการเกิดเงาจากต้นกำเนิดแสงแบบจุดหรือเรียกว่า เงาแบบขอบแข็ง (Hard edge shadow) วิธีการพื้นฐานของการหาส่วนของเงาของวัตถุนั้น ได้กล่าวไว้ในหัวข้อที่ 6.1.2 และ

ภาพที่ 63 ได้แสดงถึงขั้นตอนการหาค่าความเข้มของแสงในส่วนที่เป็นเงาด้วย[47] และค่าความเข้มของแสงของเงาจะมีค่าขึ้นอยู่กับค่าความเข้มของแสงที่กระจายอยู่โดยรอบ ซึ่งจะทำให้ความเข้มของแสงในส่วนของเงามีค่าไม่เท่ากับ 0 จึงทำให้ส่วนของภาพบริเวณนั้นมีค่าความเข้มของแสงน้อยกว่าส่วนอื่น ๆ และในบริเวณส่วนของเงานี้เองที่จะมีค่าความเข้มของแสงเท่ากันโดยตลอด เนื่องจากวิธีการตรวจสอบส่วนของเงาของวัตถุ จะมีลักษณะแบบถูก (true) หรือผิด (false) เท่านั้น คือ ถ้าวัตถุไม่ได้รับความเข้มของแสงจากต้นกำเนิดแสงจะมีค่าเป็นผิด หรืออยู่ในบริเวณที่เป็นเงา ภาพที่ 73 แสดงการเกิดเงาแบบขอบแข็ง

ภาพที่ 73

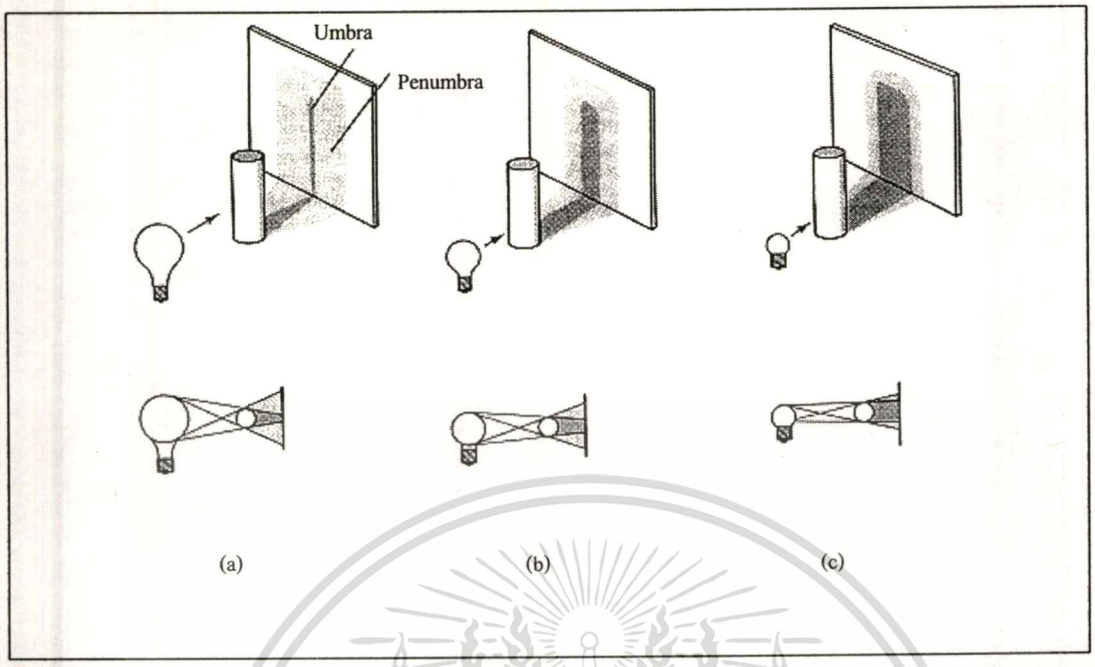


แสดง เงาของวัตถุแบบขอบแข็ง

โดยความเป็นจริงแล้วไม่มีต้นกำเนิดแสงแบบใดเลยที่มีลักษณะเหมือนกับต้นกำเนิดแสงแบบจุด ต้นกำเนิดแสงโดยทั่วไปจะให้ความเข้มของแสงมีค่าต่าง ๆ กันไปในทิศทางต่าง ๆ กัน[48] ซึ่งจากเหตุผลนี้จึงทำให้ลักษณะของเงาที่เกิดขึ้นจริงสำหรับวัตถุต่างๆ มีลักษณะแยกเป็นสองส่วน[45] เรียกว่า umbra และ penumbra ภาพที่ 74 แสดงการเกิดเงาของวัตถุจริง

umbra คือ ส่วนของเงา ที่เกิดจากการที่วัตถุไม่ได้รับแสงจากต้นกำเนิดแสงเลย และ penumbra คือ ส่วนของเงา ที่เกิดจากการที่วัตถุได้รับแสงจากต้นกำเนิดแสงเพียงบางส่วน[49] ซึ่งเงาส่วนนี้จะเป็นส่วนที่กระจายอยู่โดยรอบเงาแบบ umbra และมีค่าความเข้มของแสงมากกว่า ลักษณะการเกิดเงาแบบนี้ เรียกว่า การเกิดเงาแบบอ่อนหรือเรียบ (Soft shadow)

ภาพที่ 74



แสดง การเกิดเงาของวัตถุจริง

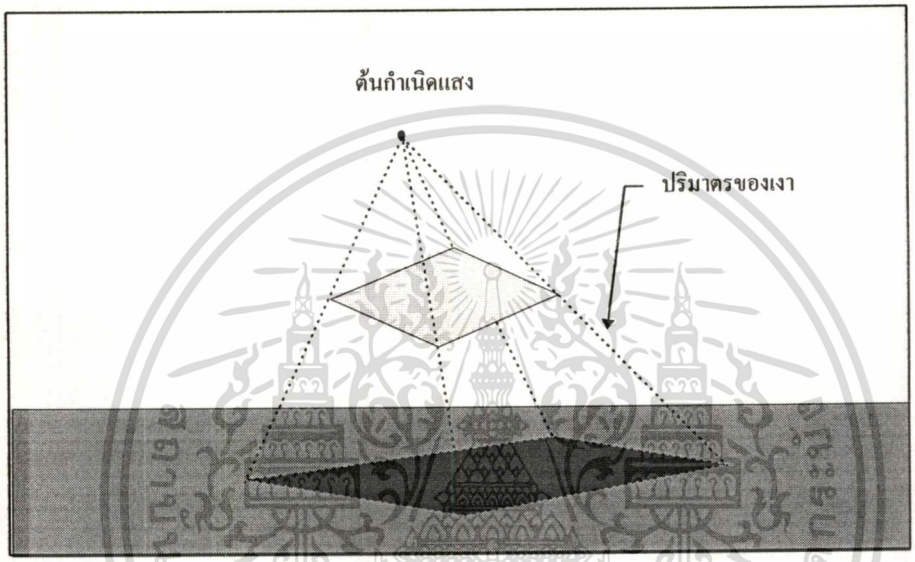
การสร้างเงาแบบเรียบ

จะเห็นได้ว่าในการจำลองภาพของวัตถุในเรื่องเงานั้น ถ้ากำหนดว่าต้นกำเนิดแสงเป็นต้นกำเนิดแสงแบบจุด จะทำให้ส่วนของเงาของวัตถุที่เกิดขึ้นเป็นแบบขอบแข็ง การทำให้เกิดเงาแบบเรียบนั้น [48] วิธีที่นิยมใช้กันมาก คือ การกระจายต้นกำเนิดแสงแบบจุดออกเป็นหลายตำแหน่ง การกระจายต้นกำเนิดแสงจะกระทำต่อเมื่อ ตรวจสอบแล้วว่าที่ตำแหน่งนั้นเป็นส่วนของเงา ค่าความเข้มของแสงที่บริเวณนี้จะมีค่าเป็น ผลรวมของค่าความเข้มของแสงแต่ละต้นกำเนิดที่ถูกกระจาย [50] แต่ในบริเวณอื่นค่าความเข้มของแสงจะมีค่ามาจากความเข้มของแสงของต้นกำเนิดแสงจริง จากวิธีการดังกล่าวนี้จะเห็นได้ว่านอกจากการตรวจสอบบริเวณที่เป็นเงาของส่วนต่างๆ แบบเดิมแล้ว คือ ที่ตำแหน่งนั้นได้รับแสงจากต้นกำเนิดแสงหรือไม่ แต่ยังสามารถตรวจสอบได้ว่าที่ตำแหน่งนั้นได้รับแสงจากต้นกำเนิดแสงเพียงบางส่วนหรือไม่ หมายถึงว่าสามารถที่จะตรวจสอบลักษณะของเงาได้ 3 สถานะด้วยกัน คือ 1. ที่ตำแหน่งนั้นวัตถุได้รับแสงจากต้นกำเนิดแสงโดยตรง 2. วัตถุได้รับแสงจากต้นกำเนิดแสงบางส่วน 3. วัตถุไม่ได้รับแสงจากต้นกำเนิด

การสร้างเงาแบบเรียบโดยวิธีการกระจายต้นกำเนิดแสงเป็น 4 จุด

ปริมาตรของเงาเกิดจากการที่มีวัตถุมาบังแสงจากต้นกำเนิดแสงแบบจุด ซึ่งลักษณะของเงาที่เกิดขึ้นนั้น จะมีลักษณะเป็นปริมาตรที่มีขนาดขึ้นกับขนาดของวัตถุที่บังแสงจากต้นกำเนิดแสงและตำแหน่งของต้นกำเนิดแสง ดังแสดงในภาพที่ 75

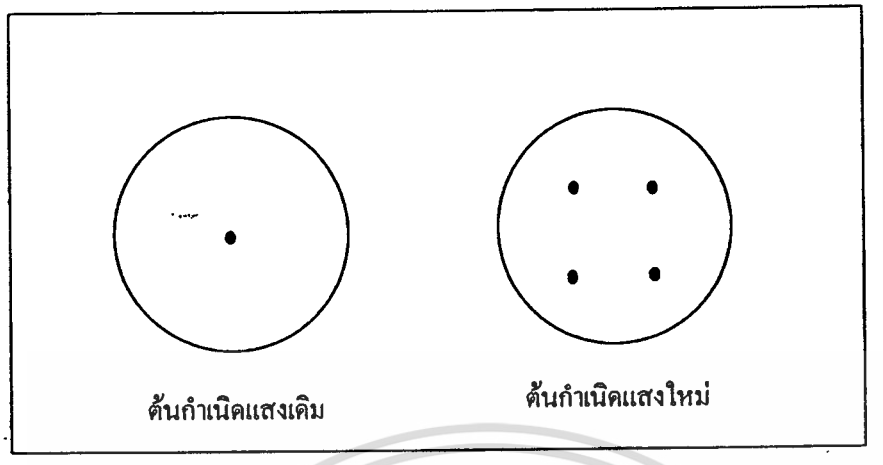
ภาพที่ 75



แสดง ปริมาตรของเงา

จากภาพที่ 75 จะเห็นว่า ไม่ว่าวัตถุใดก็ตามที่มีตำแหน่งอยู่ภายในบริเวณของปริมาตรของเงาจะถือได้ว่าวัตถุนั้นไม่ได้รับแสงจากต้นกำเนิดแสง ลักษณะของเงาที่เกิดขึ้นโดยวิธีนี้จะมีลักษณะเป็นเงาแบบขอบแข็ง ซึ่งยังไม่ตรงกับความต้องการ หรือยังไม่ถูกต้องตามความเป็นจริงของเงาที่เกิดกับวัตถุจริง วิธีการแก้ไขวิธีหนึ่งคือ เมื่อทราบว่าบริเวณใด หรือส่วนใดในระบบเป็นส่วนที่เป็นเงาแล้ว ให้กำหนดแหล่งกำเนิดแสงสมมุติที่มีตำแหน่งสัมพันธ์กับต้นกำเนิดแสงจริงในระบบและผลรวมของค่าความเข้มของแสงของต้นกำเนิดแสงสมมุติที่ตำแหน่งต่างๆมีค่าเท่ากับค่าความเข้มของแสงของต้นกำเนิดแสงจริงในระบบ จากนั้นจึงคำนวณหาค่าความเข้มของแสงที่บริเวณเป็นเงาจากต้นกำเนิดแสงสมมุติทั้งหมดผลรวมของค่าความเข้มของแสงที่ได้จะมีลักษณะคล้ายกับการเกิดเงาแบบเรียบ ภาพที่ 76 แสดงการกระจายตำแหน่งส่องสว่างของต้นกำเนิดแสงออกเป็น 4 ตำแหน่ง ภาพที่ 77 แสดงขั้นตอนการสร้างเงาแบบเรียบ โดยการกระจายตำแหน่งส่องสว่างในต้นกำเนิดแสง

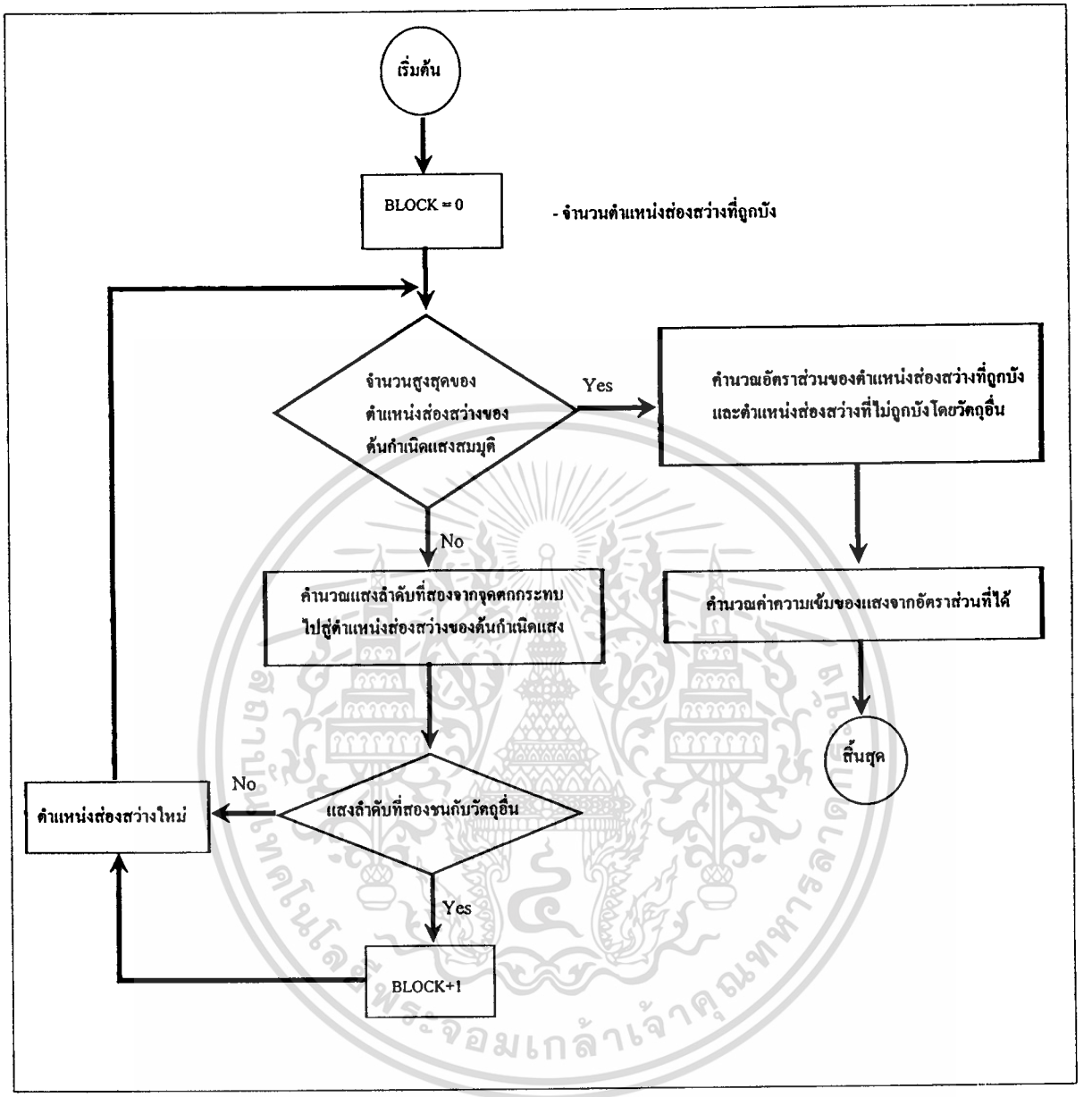
ภาพที่ 76



แสดง การกระจายตำแหน่งส่องสว่างของต้นกำเนิดแสง

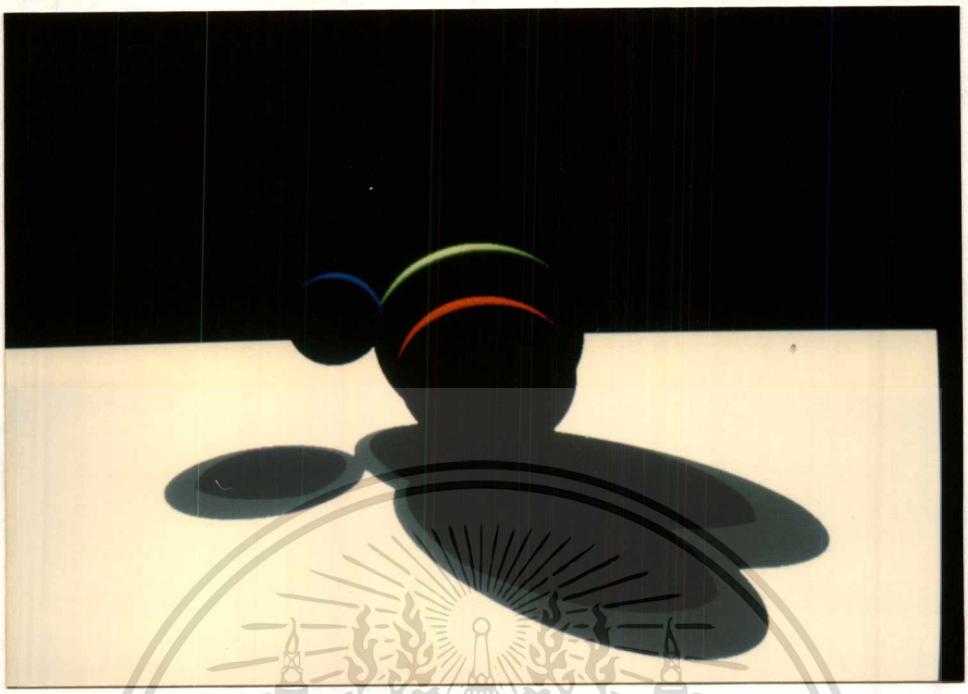
ภาพที่ 78 และภาพที่ 79 เป็นภาพจำลองของวัตถุที่สร้างขึ้นโดยใช้วิธีการสร้างเงาแบบเรียบโดยวิธีกระจายต้นกำเนิดแสงเป็น 4 จุดและรัศมีของต้นกำเนิดแสงสมมุติมีขนาด 25 และ 50 ตามลำดับ

ภาพที่ 77



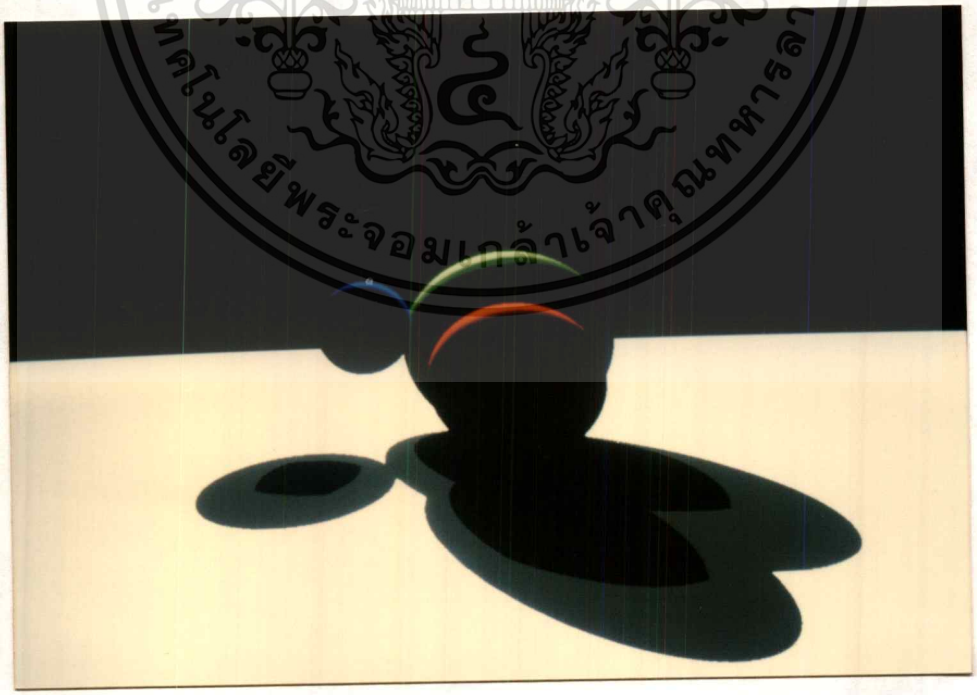
แสดง วิธีการสร้างเงาแบบเรียบ

ภาพที่ 78



แสดง ภาพจำลองเมื่อตำแหน่งส่องสว่าง 4 จุด และ r = 25

ภาพที่ 79

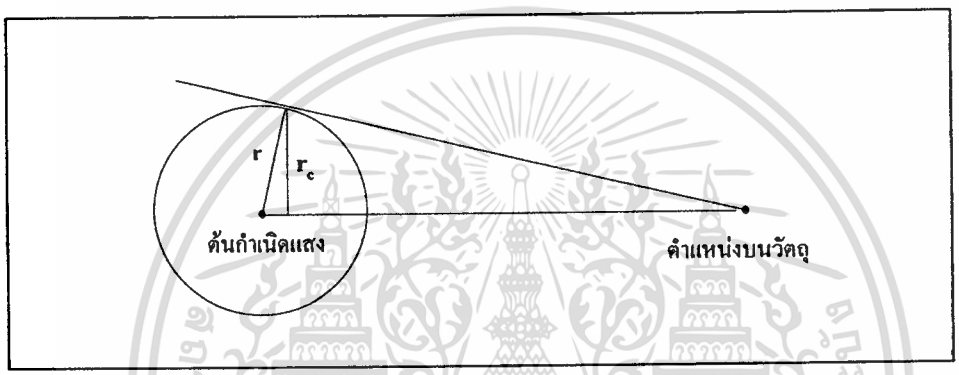


แสดง ภาพจำลองเมื่อตำแหน่งส่องสว่าง 4 จุด และ r = 50

การกระจายต้นกำเนิดแสงสมมุติโดยรูปหกเหลี่ยม

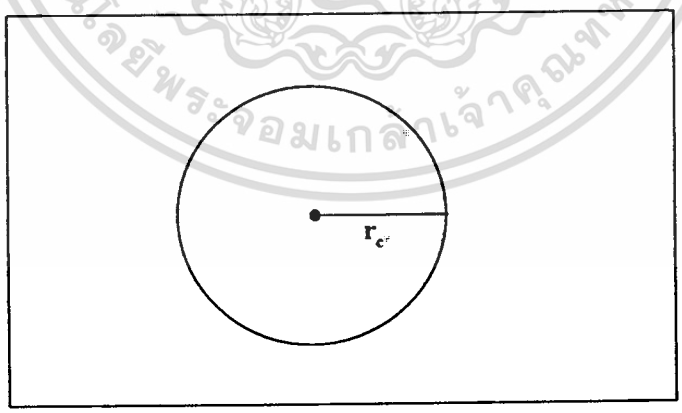
จากวิธีการสร้างแบบเรียบ โดยวิธีปริมาตรของเงานั้นต้นกำเนิดแสงสมมุติที่สร้างขึ้น จะมีการกระจายตัวของตำแหน่งที่ส่องสว่างไม่เป็นระเบียบที่แน่นอนขึ้นอยู่กับการสมมุติ จึงทำให้ลักษณะของเงาที่เกิดขึ้นไม่มีความเป็นระเบียบ และเนื่องจากต้นกำเนิดแสงที่ใช้ในการจำลองภาพนั้นเป็นต้นกำเนิดแสงแบบจุด ซึ่งมีค่าความเข้มของแสงเท่ากันในทุกทิศทาง จึงทำให้มีลักษณะของความเข้มของแสงเป็นรูปทรงกลม โดยที่มีต้นกำเนิดแสงเป็นจุดศูนย์กลางของทรงกลม ภาพที่ 80แสดงลักษณะความเข้มของแสงที่ได้จากต้นกำเนิดแสงแบบจุด

ภาพที่ 80



แสดง ค่าความเข้มของแสงที่ผิวของวัตถุ

ภาพที่ 81

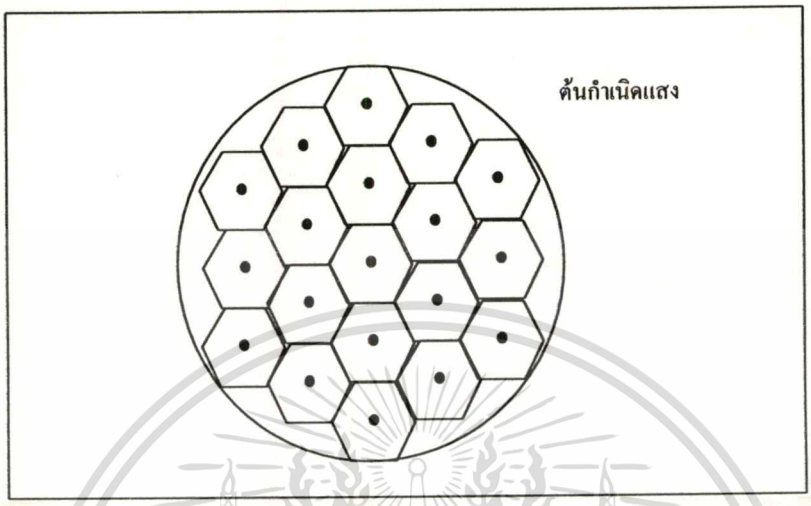


แสดง ค่าความเข้มของแสงที่มองเห็น

ภาพที่ 80 ที่ตำแหน่งใดๆ บนพื้นผิวของวัตถุ จะสามารถมองเห็นความเข้มของแสงในลักษณะที่เป็นวงกลมเท่านั้น ดังภาพที่ 81 จากเหตุผลข้อนี้การกำหนดการกระจายตัวของตำแหน่งส่องสว่างในต้นกำเนิดแสงสมมุติ จึงควรมีรูปแบบที่แน่นอนตามลักษณะของวงกลมที่ได้

วิธีการกำหนดตำแหน่งส่องสว่างในวงกลมที่สะดวกก็คือ การอาศัยรูปหกเหลี่ยมมาช่วยในการจัดตำแหน่ง[45] ภาพที่ 82 แสดงการจัดตำแหน่งโดยรูปหกเหลี่ยม

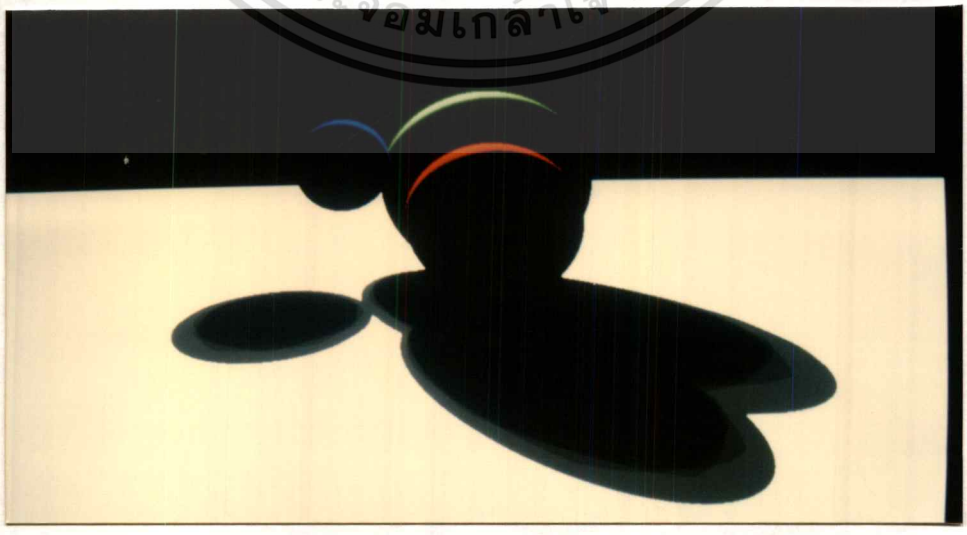
ภาพที่ 82



แสดง การกระจายตำแหน่งส่องสว่างโดยรูปหกเหลี่ยม

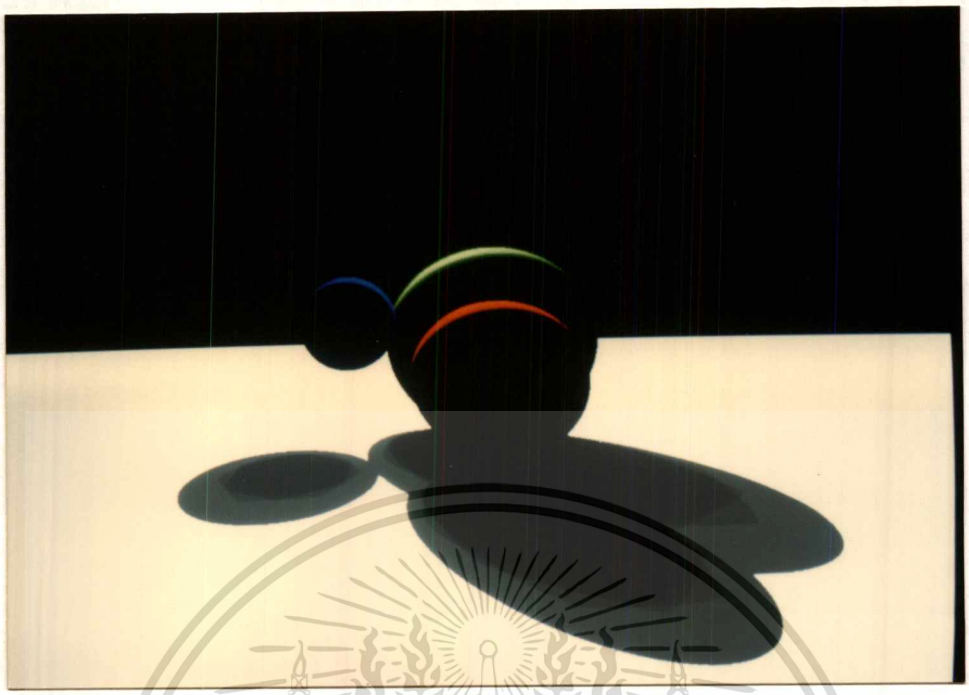
การกำหนดตำแหน่ง โดยรูปหกเหลี่ยมจะทำให้ได้ตำแหน่งส่องสว่างสมมุติที่มีความเป็นระเบียบมากขึ้นและจำนวนของตำแหน่งส่องสว่างก็ยังสามารถกำหนดได้ตามต้องการ ภาพที่ 83และ 84 แสดงภาพที่ได้จากการกำหนดให้มีตำแหน่งส่องสว่างส่วนสมมุติ 19 ตำแหน่งและรัศมีของต้นกำเนิดแสงสมมุติมีค่า 25 และ 50 ตามลำดับ

ภาพที่ 83



แสดง ภาพจำลองเมื่อตำแหน่งส่องสว่าง 19 จุด และ r = 25

ภาพที่ 84



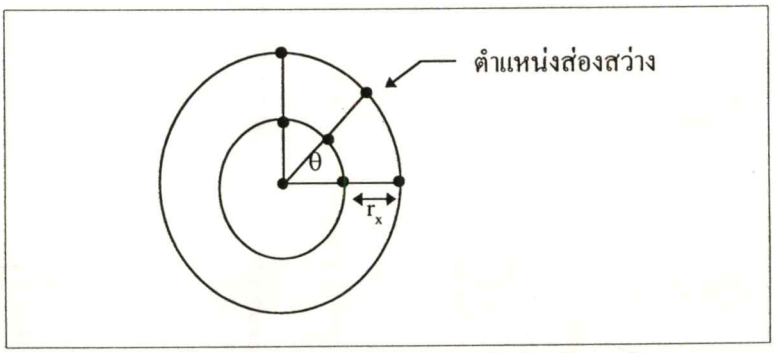
แสดง ภาพจำลองเมื่อตำแหน่งส่องสว่าง 19 จุด และ r = 50

การกระจายต้นกำเนิดแสงโดยอาศัยรัศมีและมุมรอบจุดศูนย์กลาง

การกระจายต้นกำเนิดแสงโดยอาศัยรัศมีและมุมรอบจุดศูนย์กลางแบบที่ 1

จากการกำหนดให้ต้นกำเนิดแสงสมมุติมีลักษณะเป็นวงกลม การกำหนดตำแหน่งส่องสว่างภายในต้นกำเนิดแสงสามารถที่จะหาได้จากคุณสมบัติของวงกลมเอง คือการใช้รัศมีของวงกลมและมุมรอบจุดศูนย์กลางของวงกลม ในการกำหนดตำแหน่งส่องสว่างโดยวิธีนี้ ตำแหน่งส่องสว่างต่างๆ จะถูกกำหนดให้มีขอบเขต (Zone) ที่แยกจากกัน ภาพที่ 85 แสดงการกำหนดตำแหน่งส่องสว่างภายในต้นกำเนิดแสงสมมุติ โดยอาศัยรัศมีและมุมรอบจุดศูนย์กลาง

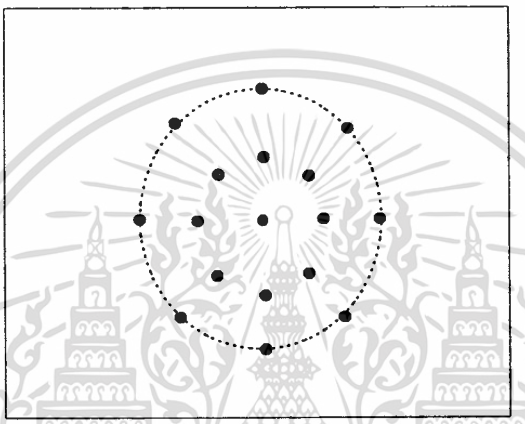
ภาพที่ 85



แสดง การกระจายตำแหน่งส่องสว่างในต้นกำเนิดแสง

จากภาพที่ 85 ตำแหน่งที่หนึ่งของจุดส่องสว่าง คือที่จุดศูนย์กลางของต้นกำเนิดแสงสมมุติ ส่วนตำแหน่งอื่น ๆ นั้นสามารถคำนวณหาได้จากค่ามุมรอบจุดศูนย์กลาง θ และตามสัดส่วนของรัศมี (r_x) ที่ได้กำหนดไว้ เช่น ถ้ากำหนดมุมรอบจุดศูนย์กลาง คือ 45 องศา และสัดส่วนของรัศมีคือ 2 ส่วน ดังนั้นจำนวนของจุดส่องสว่างภายในต้นกำเนิดแสงสมมุติจะมีทั้งหมด 17 ตำแหน่งด้วยกันดังแสดงในภาพที่ 86 และผลรวมของค่าความเข้มของแสงจากตำแหน่งส่องสว่างจะมีค่าเท่ากับค่าความเข้มของแสงของต้นกำเนิดแสง

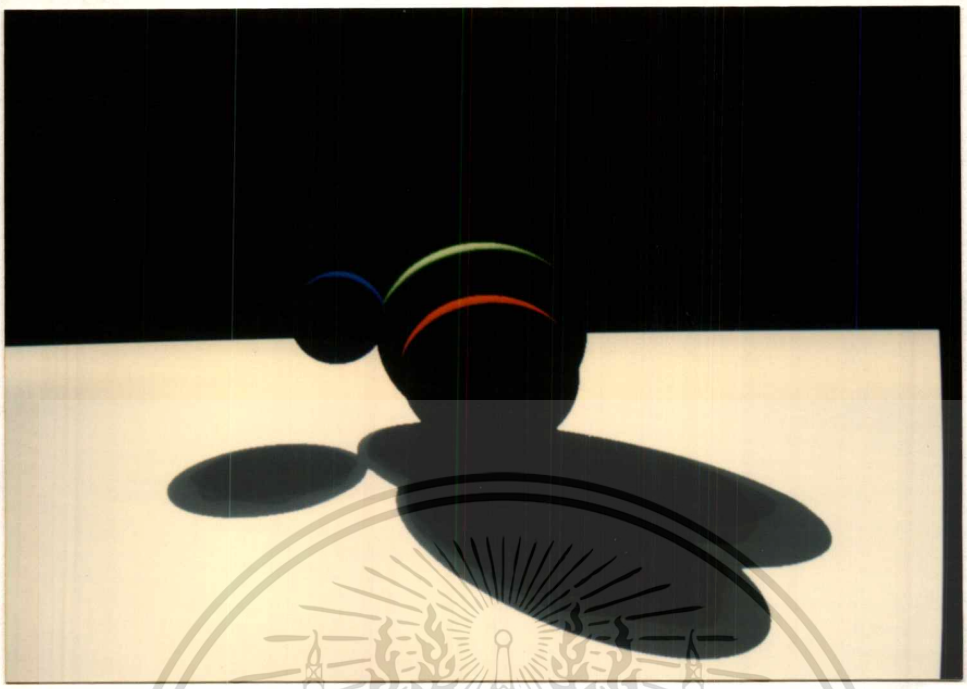
ภาพที่ 86



แสดง จำนวนจุดส่องสว่าง เมื่อ $\theta = 45$ องศาและ $r_x = r/2$

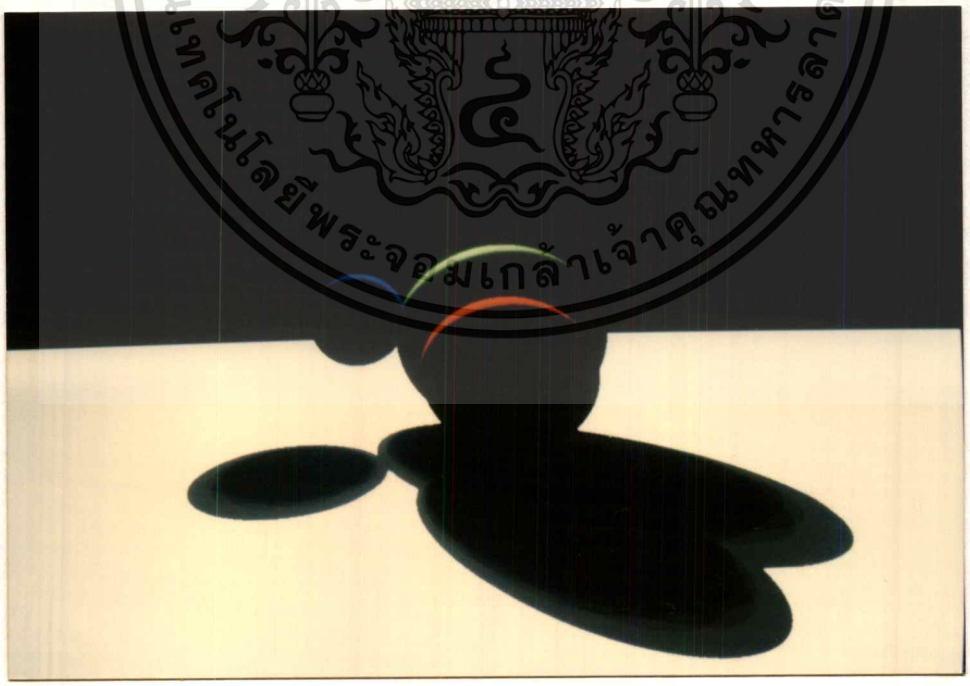
การกำหนดจุดส่องสว่างโดยวิธีนี้จะให้ผลที่คล้ายกับการใช้รูปหกเหลี่ยมแต่ จะมีความสะดวกในการคำนวณตำแหน่งส่องสว่างมากกว่า ภาพที่ 87, 88, 89, 90 และ 91 แสดงภาพที่ได้จากการกระจายตำแหน่งส่องสว่างภายในต้นกำเนิดแสงด้วยมุมรอบจุดศูนย์กลางและรัศมีที่มีค่าต่างกัน

ภาพที่ 87



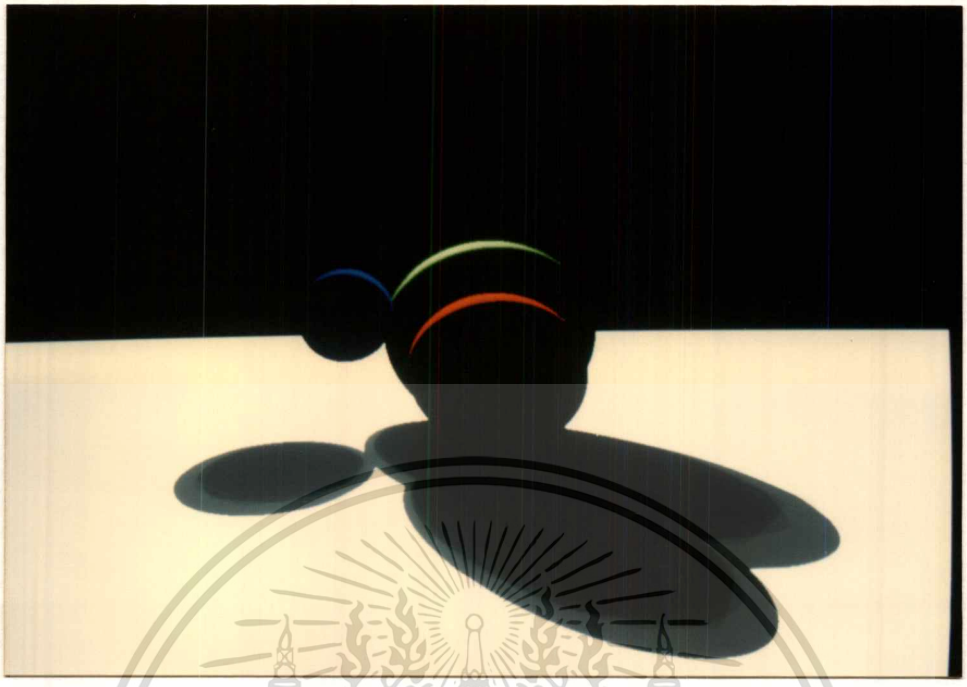
แสดง ภาพจำลองเมื่อ r = 25, θ = 45, Zone = 2

ภาพที่ 88



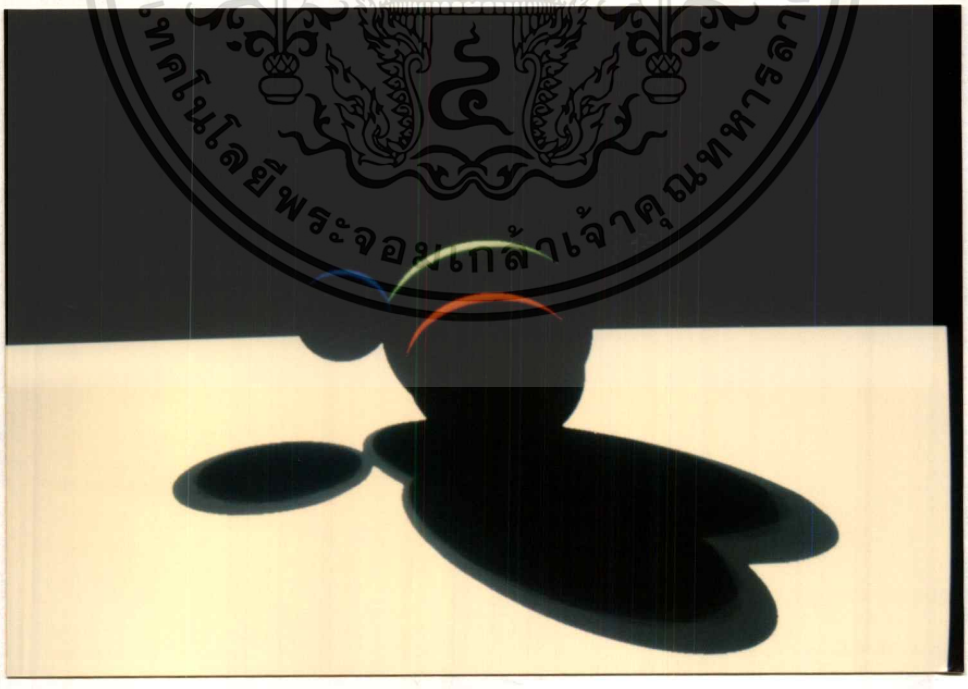
แสดง ภาพจำลองเมื่อ r = 25, θ = 30, Zone = 2

ภาพที่ 89



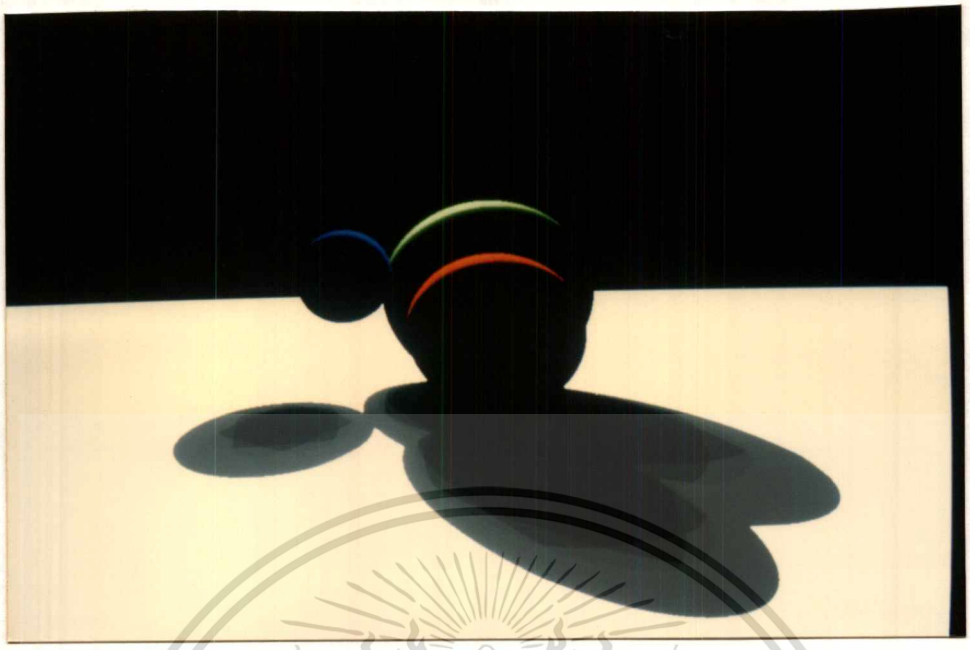
แสดง ภาพจำลองเมื่อ r = 25, θ = 45, Zone = 3

ภาพที่ 90



แสดง ภาพจำลองเมื่อ r = 25, θ = 30, Zone = 3

ภาพที่ 91

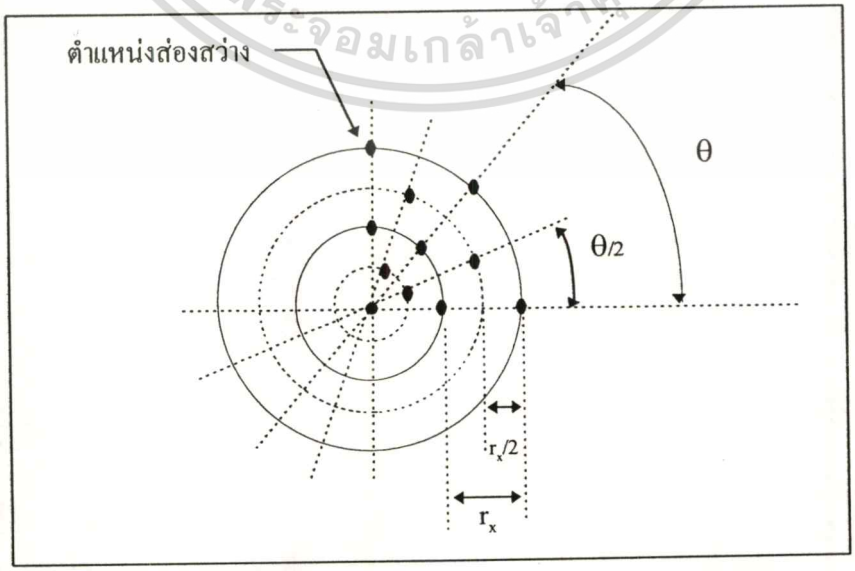


แสดง ภาพจำลองเมื่อ r = 50, θ = 45, Zone = 2

การกระจายต้นกำเนิดแสงโดยอาศัยรัศมีและมุมรอบจุดศูนย์กลางแบบที่ 2

การกระจายตำแหน่งส่องสว่างโดยวิธีนี้เป็นผลมาจากหัวข้อก่อน แต่จะมีคุณสมบัติที่มีความละเอียดขึ้น โดยอาศัยตำแหน่งที่อยู่ระหว่างตำแหน่งของจุดส่องสว่างที่คำนวณได้จากวิธีในหัวข้อการกระจายต้นกำเนิดแสงโดยอาศัยรัศมีและมุมรอบจุดศูนย์กลางแบบที่ 1 ภาพที่ 92 แสดงลักษณะการแบ่งตำแหน่งส่องสว่างในต้นกำเนิดแสงสมมุติ

ภาพที่ 92



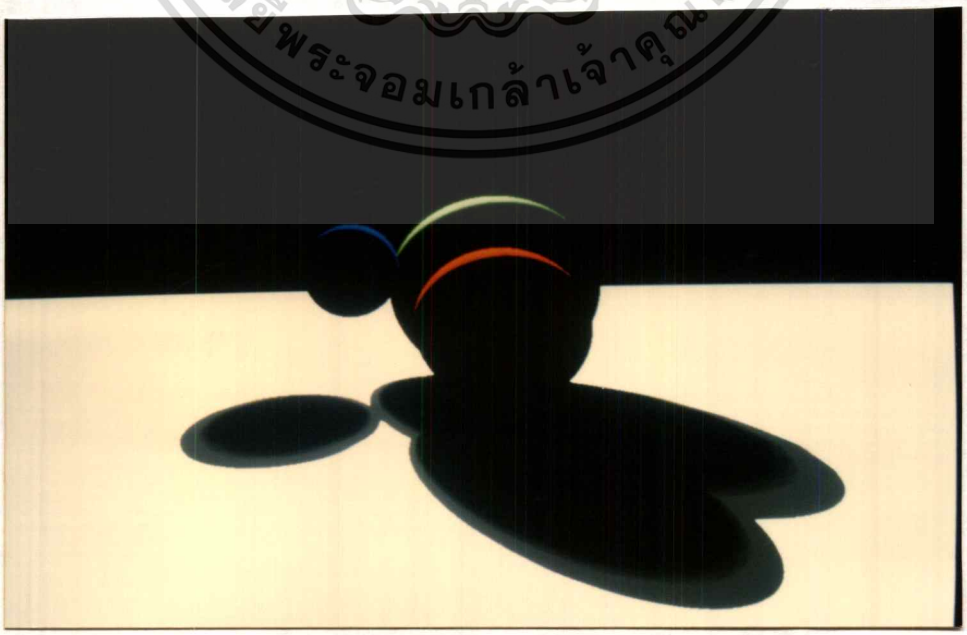
แสดง การกระจายต้นกำเนิดแสงโดยอาศัยรัศมีและมุมรอบจุดศูนย์กลางแบบที่ 2

การกำหนดตำแหน่งส่องสว่างโดยวิธีนี้จะทำให้ได้ตำแหน่งส่องสว่างที่มีความละเอียดมากขึ้น ซึ่งทำให้โอกาสการเกิดเงาแบบเรียบ มีค่ามากขึ้นเนื่องจากโอกาสในการที่แสงจากต้นกำเนิดแสงตกกระทบกับวัตถุมีมากขึ้นแต่ก็จะทำให้ค่าความเข้มของแสงที่ตำแหน่งส่องสว่างมีค่าน้อยลงด้วย ภาพที่ 93, 94, 95, 96 และ 97 แสดงภาพที่ได้จากวิธีการ 6.4.3.2

ภาพที่ 93

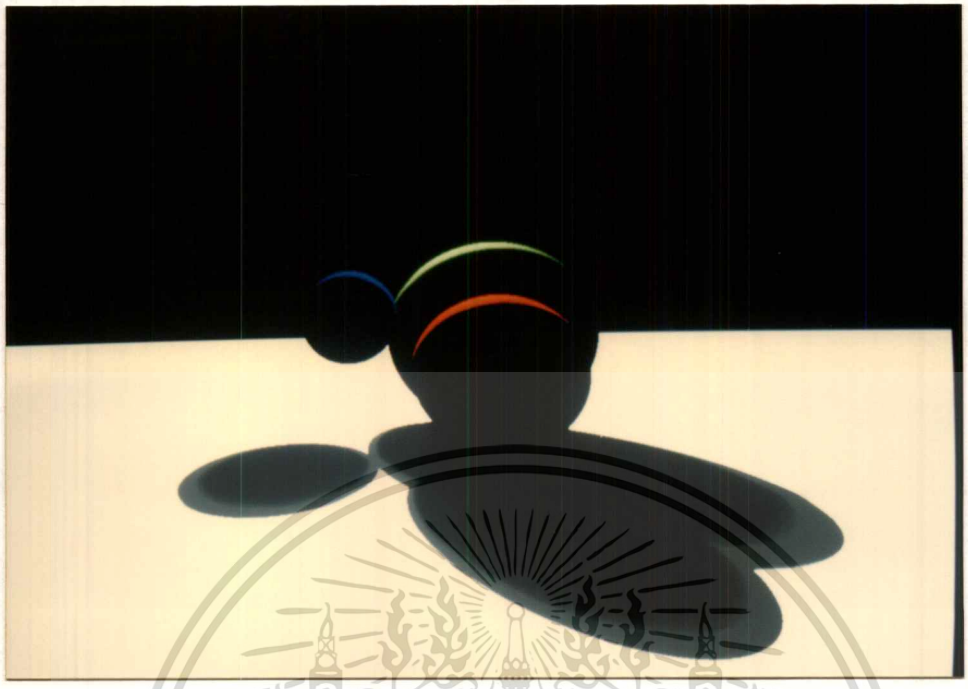
แสดง ภาพจำลองเมื่อ r = 25, θ = 45, Zone = 2

ภาพที่ 94



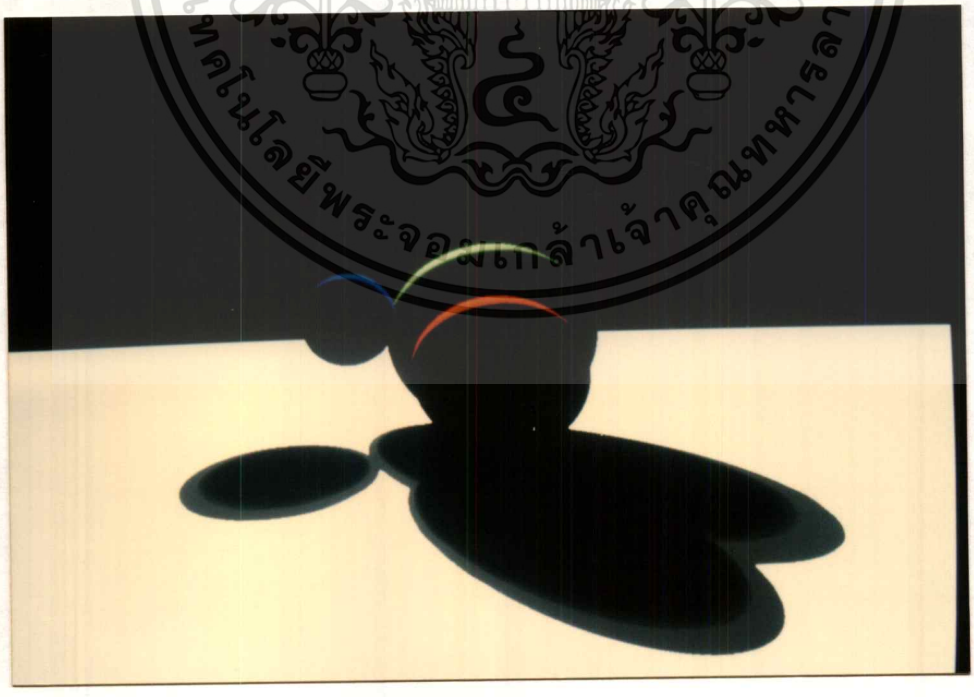
แสดง ภาพจำลองเมื่อ r = 25, θ = 30, Zone = 2

ภาพที่ 95



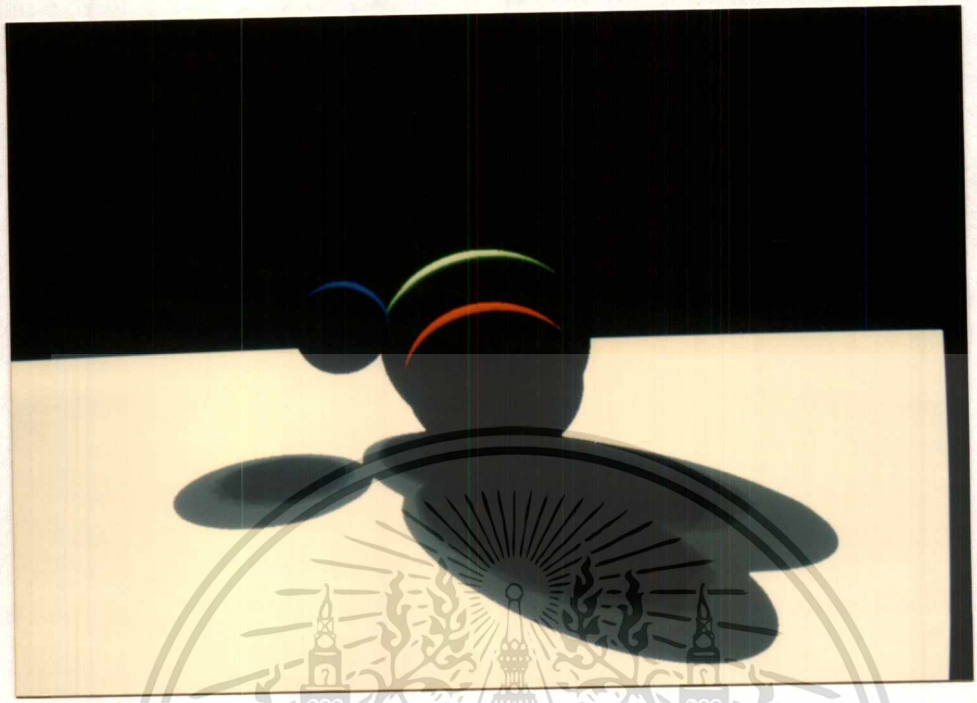
แสดง ภาพจำลองเมื่อ r = 25, θ = 45, Zone = 3

ภาพที่ 96



แสดง ภาพจำลองเมื่อ r = 25, θ = 30, Zone = 3

ภาพที่ 97

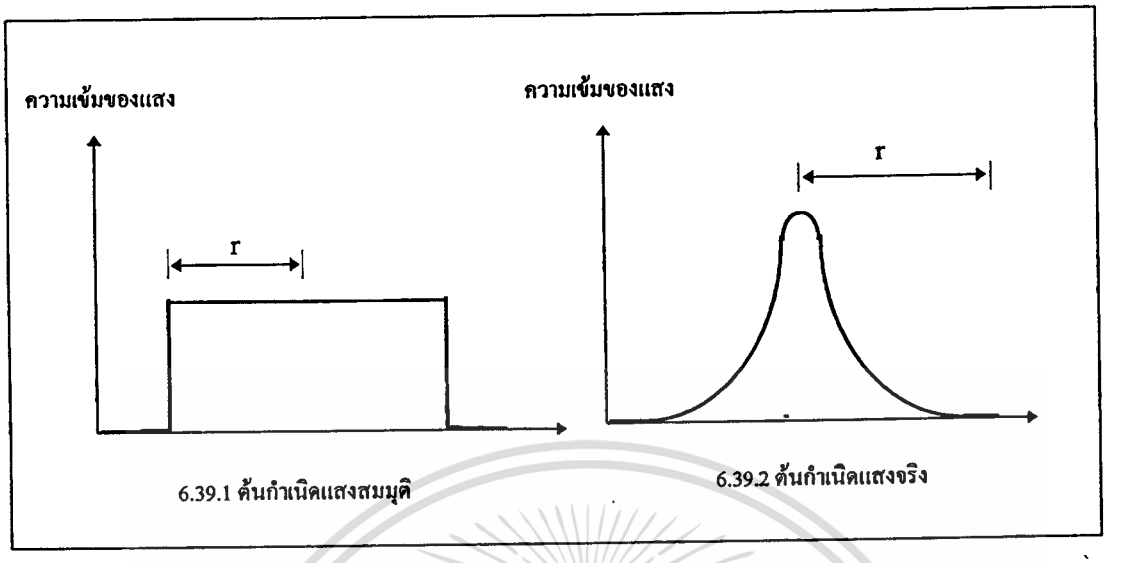


แสดง ภาพจำลองเมื่อ r = 50, θ = 45, Zone = 2

การกระจายค่าความเข้มของแสงในต้นกำเนิดแสง

จากการจำลองภาพโดยอาศัยการกระจายตำแหน่งส่องสว่างของต้นกำเนิดแสงในส่วนที่ผ่านมาได้กระทำภายใต้ข้อกำหนดที่สมมุติให้ตำแหน่งส่องสว่างที่กระจายอยู่ภายในต้นกำเนิดแสงสมมุตินั้น มีค่าความเข้มเท่ากันซึ่งจะเป็นผลทำให้ค่าความเข้มของแสงของบริเวณที่เป็นเงานั้นจะมีค่าเป็นสัดส่วนโดยตรงกับค่าความเข้มของแสงจากต้นกำเนิดแสง และถ้าพิจารณาจากต้นกำเนิดแสงโดยทั่วไปจะพบว่าลักษณะของความเข้มของแสงที่มองเห็นได้นั้น จะมีลักษณะที่บริเวณจุดศูนย์กลางของต้นกำเนิดแสงจะมีค่าความเข้มที่มากที่สุด ส่วนความเข้มของแสงที่ไกลออกไปนั้นจะมีค่าลดลงตามความสัมพันธ์กับรัศมีของต้นกำเนิดแสงที่มองเห็น และจากข้อกำหนดว่าความเข้มของแสงที่เดินทางไปในทิศทางใดๆ จะมีค่าความเข้มของแสงลดลงตามกำลังของระยะทางที่เคลื่อนที่ จากลักษณะการมองเห็นต้นกำเนิดแสงดังภาพที่ 80 และ 81 รวมถึงคุณสมบัติที่กล่าวไปแล้วสามารถที่จะแสดงค่าความเข้มของแสงจากต้นกำเนิดแสงสมมุติและต้นกำเนิดแสงจริงได้ ในภาพที่ 98 เมื่อ r คือรัศมีของต้นกำเนิดแสง

ภาพที่ 98



แสดง ค่าความเข้มของแสงของต้นกำเนิดแสง

การกระจายค่าความเข้มของแสงในต้นกำเนิดแสงแบบเกาส์

การกระจายตัวของเกาส์ (gaussian distribution) เป็นสมการที่ใช้ในการอธิบายการกระจายตัวของความน่าจะเป็น (Probability) ของเหตุการณ์ที่เกิดขึ้น[51] ซึ่งมีสมการดังนี้

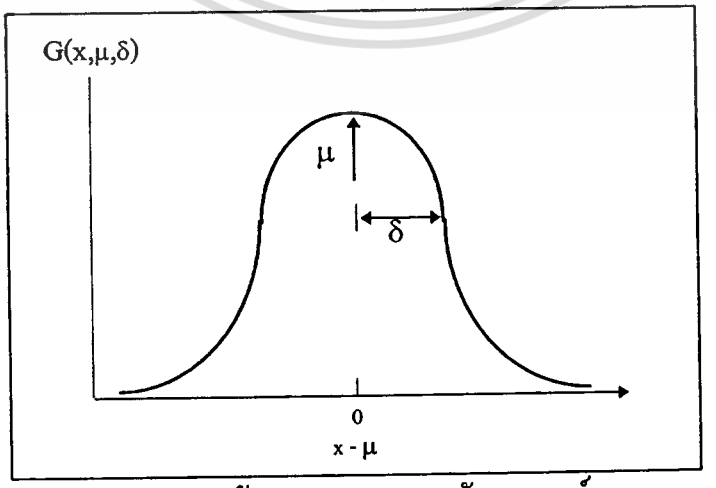
$$G = \frac{1}{\delta\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\delta}\right)^2\right) \qquad (74)$$

เมื่อ

δ = Standard deviation

μ = Mean

ภาพที่ 99

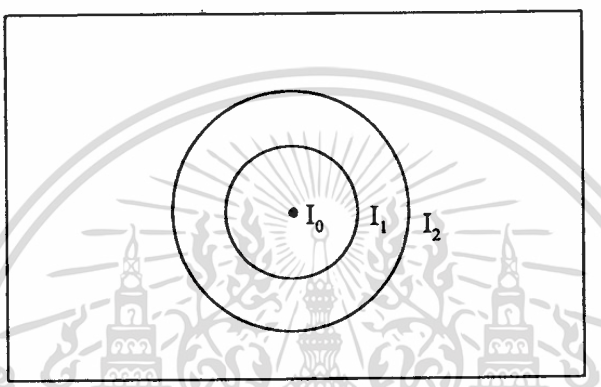


แสดง ลักษณะการกระจายตัวของเกาส์

จากภาพที่ 99 จะเห็นว่าลักษณะของการกระจายตัวที่ได้นั้นคล้ายกับลักษณะการกระจายค่าความเข้มของแสงในต้นกำเนิดแสงที่ใช้ในการจำลองภาพ จากลักษณะที่ได้นี้ จึงได้นำเอาความสัมพันธ์ของการกระจายตัวของเกาส์มาช่วยในการคำนวณค่าความเข้มของแสงในต้นกำเนิดแสง

จากหัวข้อการกระจายต้นกำเนิดแสงโดยอาศัยรัศมีและมุมรอบจุดศูนย์กลางแบบที่ 1 ถ้าแบ่งต้นกำเนิดแสงออกเป็น 2 ขอบเขต ดังภาพที่ 100 ซึ่งจะทำให้ได้ค่าความเข้มของแสงในต้นกำเนิดแสงแยกออกเป็น 3 ระดับ คือ I_0 , I_1 , I_2 ตามลำดับ

ภาพที่ 100



แสดง การแบ่งขอบเขตในต้นกำเนิดแสง

การหาค่าความเข้มของแสง โดยอาศัยการกระจายตัวของเกาส์ในแต่ละขอบเขตสามารถกระทำได้ดังนี้ เมื่อ

I คือ ค่าความเข้มของแสงของต้นกำเนิดแสง

I_g คือ ผลรวมของค่าความเข้มของแสงที่ได้โดยวิธีของเกาส์

I_{g0}, I_{g1} และ I_{g2} คือค่าความเข้มของแสงที่ได้จากวิธีของเกาส์ ดังภาพที่ 101

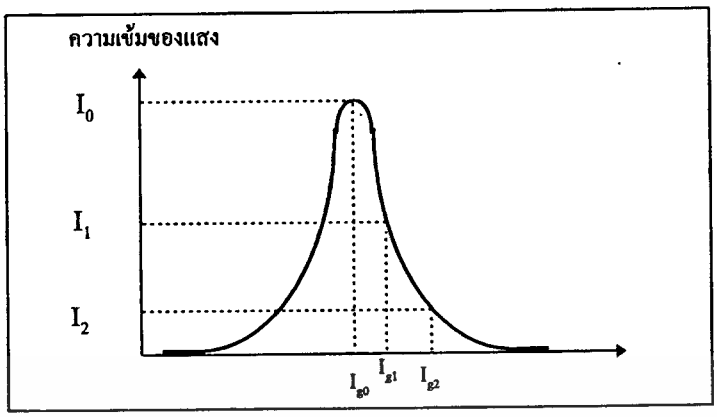
$$I_g = I_{g0} + I_{g1} + I_{g2} \qquad (75)$$

ถ้ากำหนดให้ ค่าความเข้มของแสงในต้นกำเนิดแสงในแต่ละขอบเขตและความเข้มของแสงที่ได้โดยวิธีของเกาส์มีความสัมพันธ์กัน โดยที่

$$I_i \propto I_{gi} \qquad (76)$$

เมื่อ I_i คือ ค่าความเข้มของแสงของต้นกำเนิดแสงในแต่ละขอบเขต

ภาพที่ 101



แสดง การหาค่าความน่าจะเป็นโดยวิธีของเกาส์

และ $$I_g \propto I \qquad (77)$$

$$I_1 \propto \frac{I}{I_g} * I_{g1} \qquad (78)$$

$$I_i \propto \frac{I}{I_g} * I_{gi} \qquad (79)$$

จากสมการที่ (79) จะทำให้สามารถประมาณค่าความเข้มของแสงของต้นกำเนิดแสงในแต่ละขอบเขตได้ และถ้า I_x คือค่าความเข้มของแสงในแต่ละจุดของแต่ละขอบเขตดังนั้น

$$I_i = \sum_{i=0}^{n} I_{xi} \qquad (80)$$

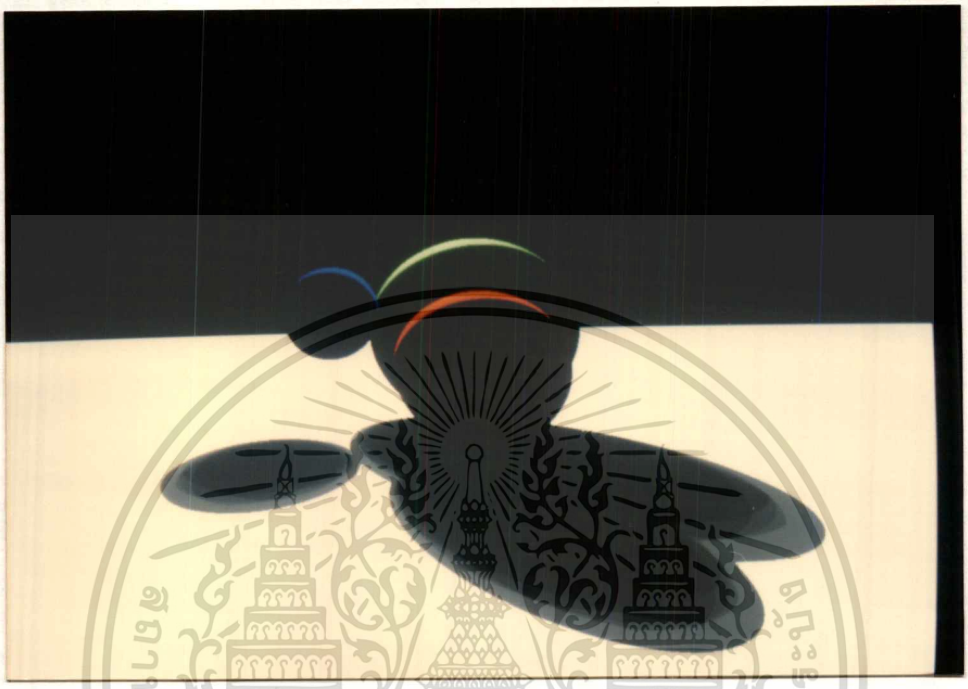
เมื่อ n คือจำนวนตำแหน่งส่องสว่างในแต่ละขอบเขต

ดังนั้น $$I_{xi} = \frac{I_i}{n} \qquad (81)$$

จากสมการที่ (81) เป็นค่าความเข้มของแสงที่ตำแหน่งส่องสว่างต่างๆภายในต้นกำเนิดแสง ซึ่งสามารถนำไปใช้ในการสร้างเงาของวัตถุแบบเรียบได้ ภาพที่ 102, 103, 104 และ 105

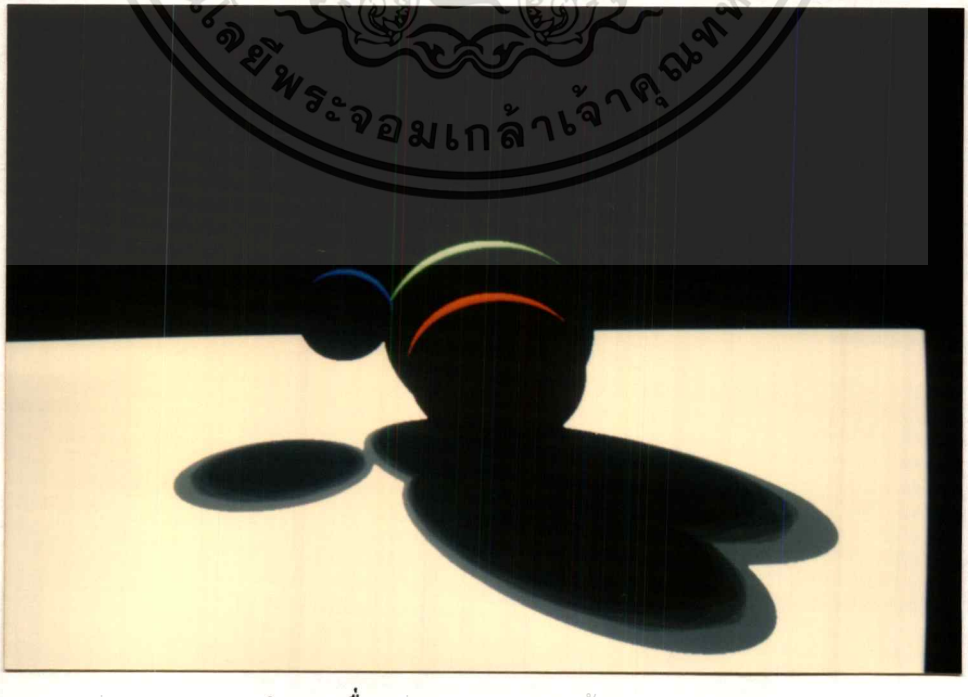
แสดงภาพจำลองที่ได้จากวิธีของเกาส์ เมื่อ มีค่า 20, 40, 60 และ 80 ตามลำดับ ภาพที่ 106, 107 และ 108 แสดงภาพจำลองที่ตำแหน่งส่องสว่างมีค่าแตกต่างกัน

ภาพที่ 102



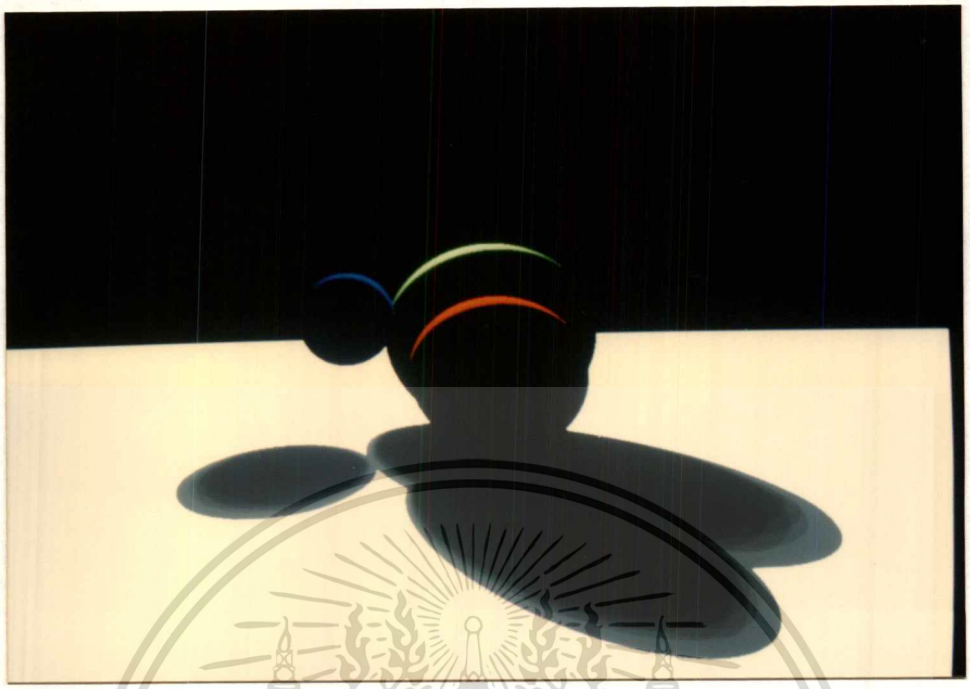
แสดง ภาพจำลองเมื่อ $\delta = 20$, $r = 25$, $\theta = 45$, Zone = 2

ภาพที่ 103



แสดง ภาพจำลองเมื่อ $\delta = 40$, $r = 25$, $\theta = 45$, Zone = 2

ภาพที่ 104

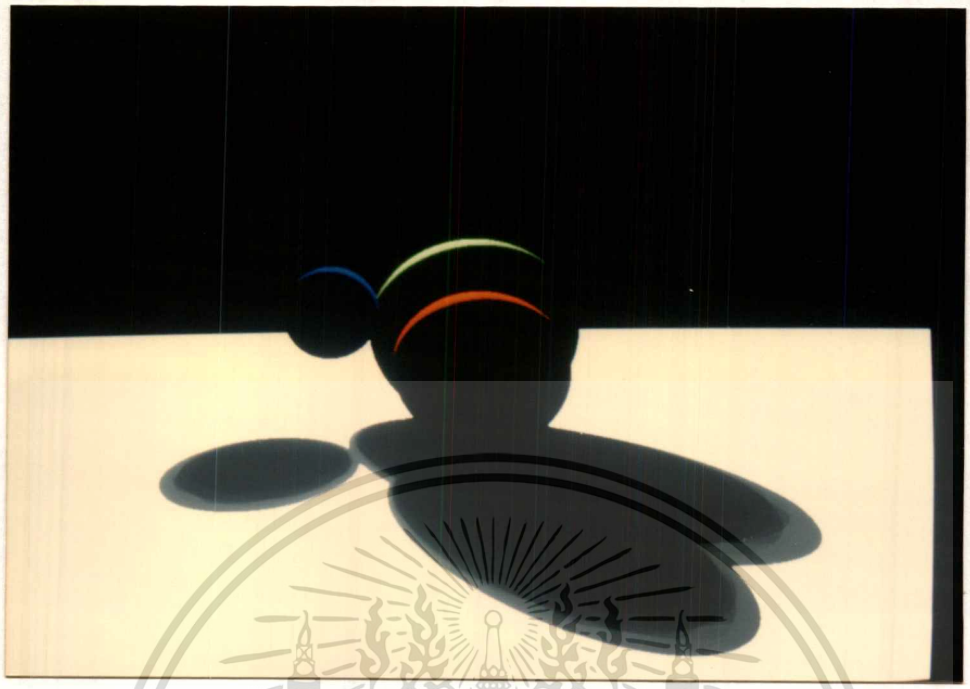


แสดง ภาพจำลองเมื่อ $\delta = 60$, $r = 25$, $\theta = 45$, Zone = 2

ภาพที่ 105

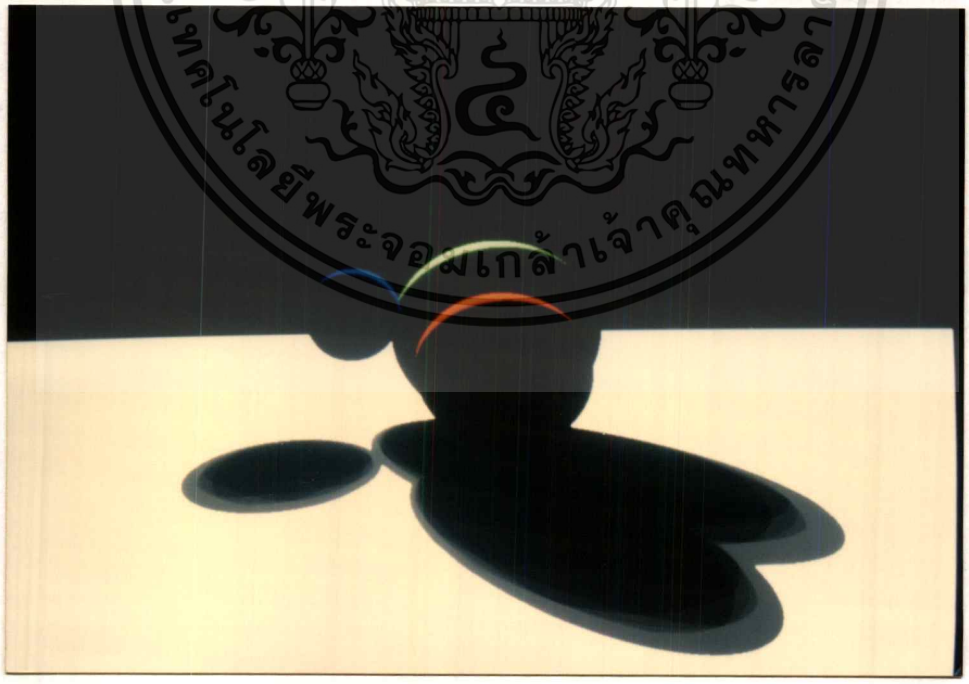
แสดง ภาพจำลองเมื่อ $\delta = 80$, $r = 25$, $\theta = 45$, Zone = 2

ภาพที่ 106



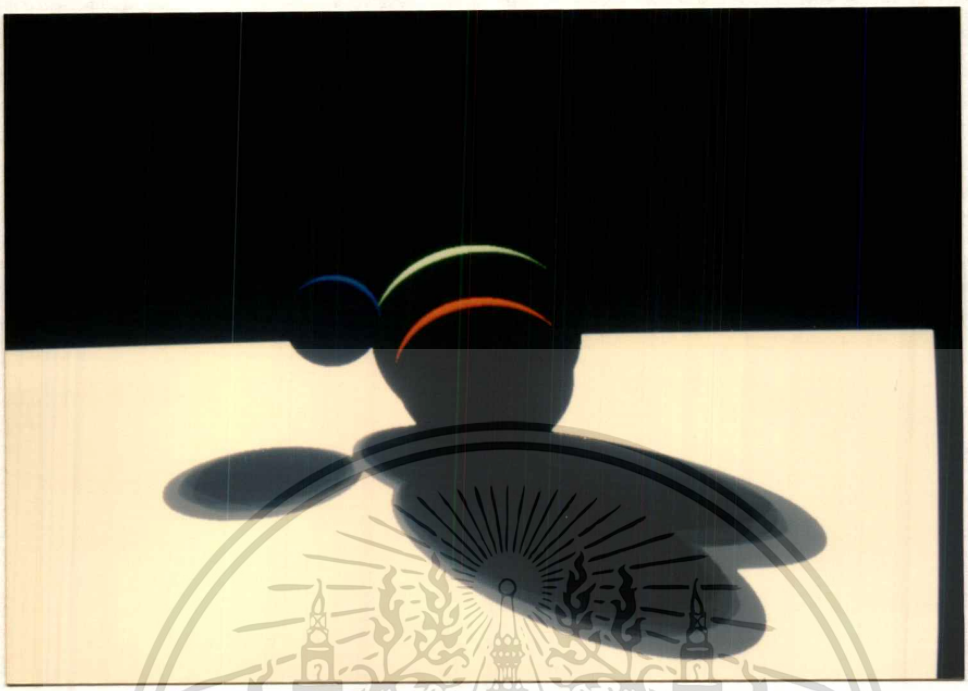
แสดง ภาพจำลองเมื่อ $\delta = 40$, $r = 25$, $\theta = 30$, Zone = 2

ภาพที่ 107



แสดง ภาพจำลองเมื่อ $\delta = 20$, $r = 25$, $\theta = 45$, Zone = 3

ภาพที่ 108



แสดง ภาพจำลองเมื่อ $\delta = 40$, $r = 50$, $\theta = 45$, Zone = 2

การแก้ไขข้อผิดพลาดของการซ้อนกันของเงาแบบเรียบ

ในการสร้างเงาแบบเรียบโดยอาศัยการกระจายตำแหน่งส่องสว่างในต้นกำเนิดแสงนั้น ทำให้ระดับความเข้มของแสงในส่วนที่เป็นเงานั้นมีความแตกต่างกัน แต่เงาแบบเรียบที่ได้นั้นจะมีลักษณะเป็นเงาแบบของแข็งที่มีค่าความเข้มของแสงต่างๆ กันซ้อนไปเรื่อยๆ ซึ่งเป็นผลมาจากการกระจายตำแหน่งส่องสว่างในต้นกำเนิดแสง เนื่องจากตำแหน่งส่องสว่างที่ถูกกระจายไปนั้นกระทำตัวเปรียบเหมือนต้นกำเนิดแสงแบบจุดในตำแหน่งที่ต่างๆกัน ซึ่งจากเหตุผลนี้เองที่ทำให้เงาที่เกิดขึ้นมีลักษณะเป็นเงาแบบขอบแข็งที่ซ้อนกัน

การแก้ปัญหาที่เกิดขึ้นนี้กระทำได้โดยการเปลี่ยนแปลงตำแหน่งส่องสว่างที่ถูกกระจายนั้นไม่ให้มีตำแหน่งที่แน่นอนในการสร้างภาพในส่วนที่เป็นเงาของวัตถุ หรือ ทำให้ตำแหน่งส่องสว่างต่าง ๆ ที่กระจายอยู่นั้นถูกรบกวนให้เปลี่ยนแปลงตำแหน่งตลอดเวลาซึ่งเรียกว่าการใช้ Jittering [33] มาใช้ในการเปลี่ยนแปลงตำแหน่งของตำแหน่งส่องสว่างต่างๆ ดังภาพที่ 109 เมื่อ

$L_i(x_i, y_i)$ คือตำแหน่งส่องสว่างใดๆในต้นกำเนิดแสง

Δx คือ Jitter factor ในแนวแกน x

Δy คือ Jitter factor ในแนวแกน y

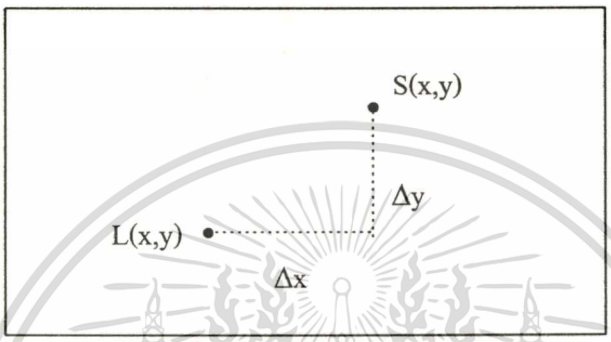
$S_i(x_i, y_i)$ คือตำแหน่งส่องสว่างใดๆในต้นกำเนิดแสงภายหลังการถูกรบกวน

โดยที่

$$S_i(x_i, y_i) = L_i(x_i + \Delta x, y_i + \Delta y) \qquad (82)$$

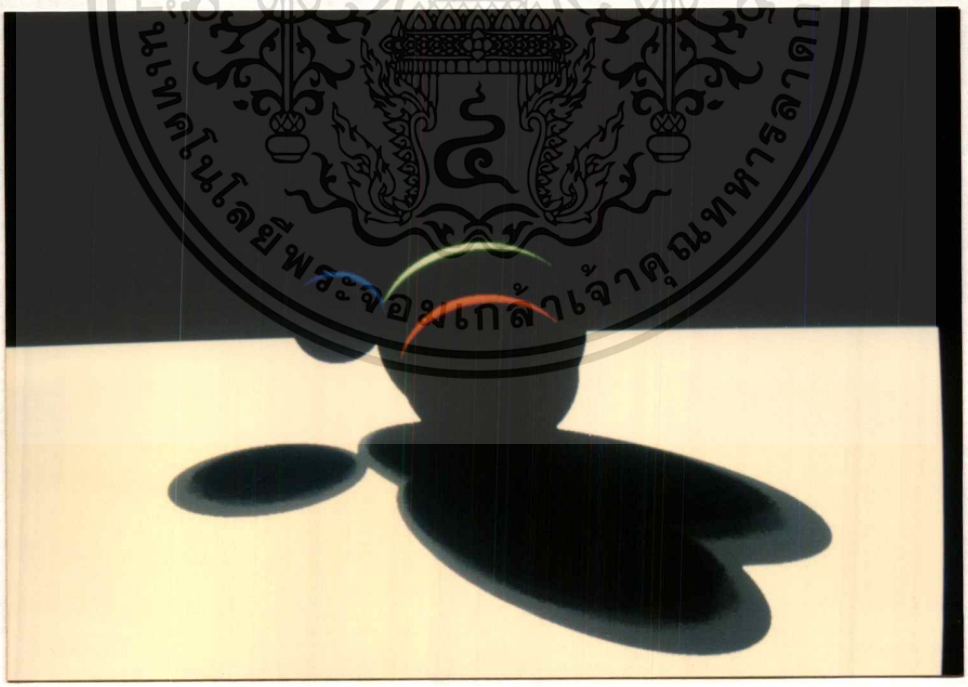
ภาพที่ 110, 111 และ 112 เป็นภาพที่ได้จากการจำลองภาพเมื่อ Jitter factor มีค่า 0.25, 0.50 และ 0.75 ตามลำดับ ภาพที่ 113 แสดงขั้นตอนการสร้างภาพโดยใช้ Jitter factor

ภาพที่ 109



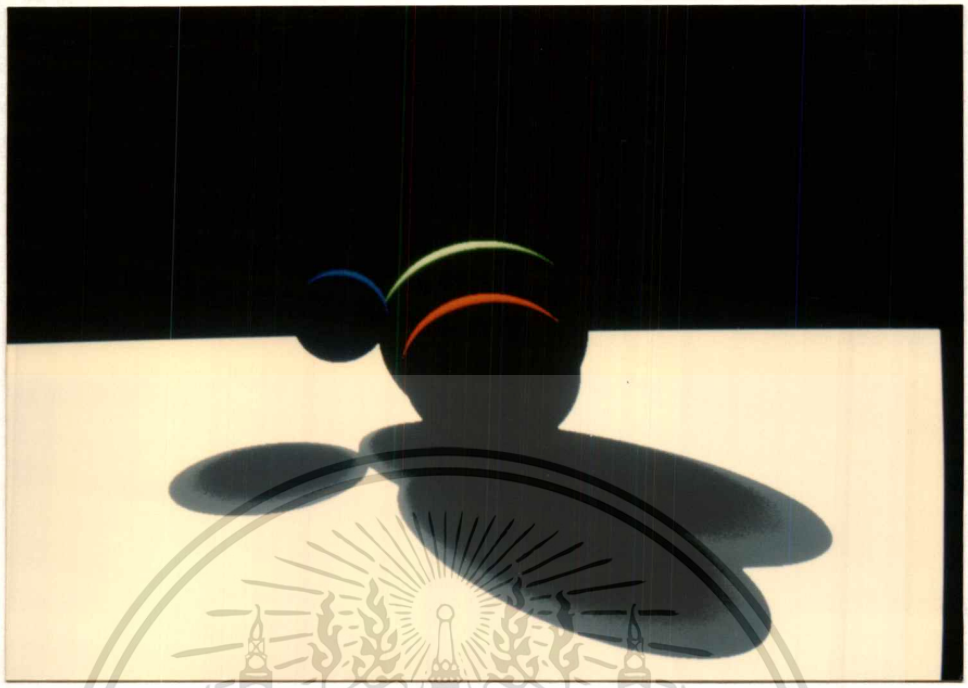
แสดง การเปลี่ยนแปลงตำแหน่งโดย Jitter factor

ภาพที่ 110



แสดง ภาพจำลองเมื่อ $\delta = 40$, $r = 25$, $\theta = 45$, Zone = 2, Jitter factor = 0.25

ภาพที่ 111

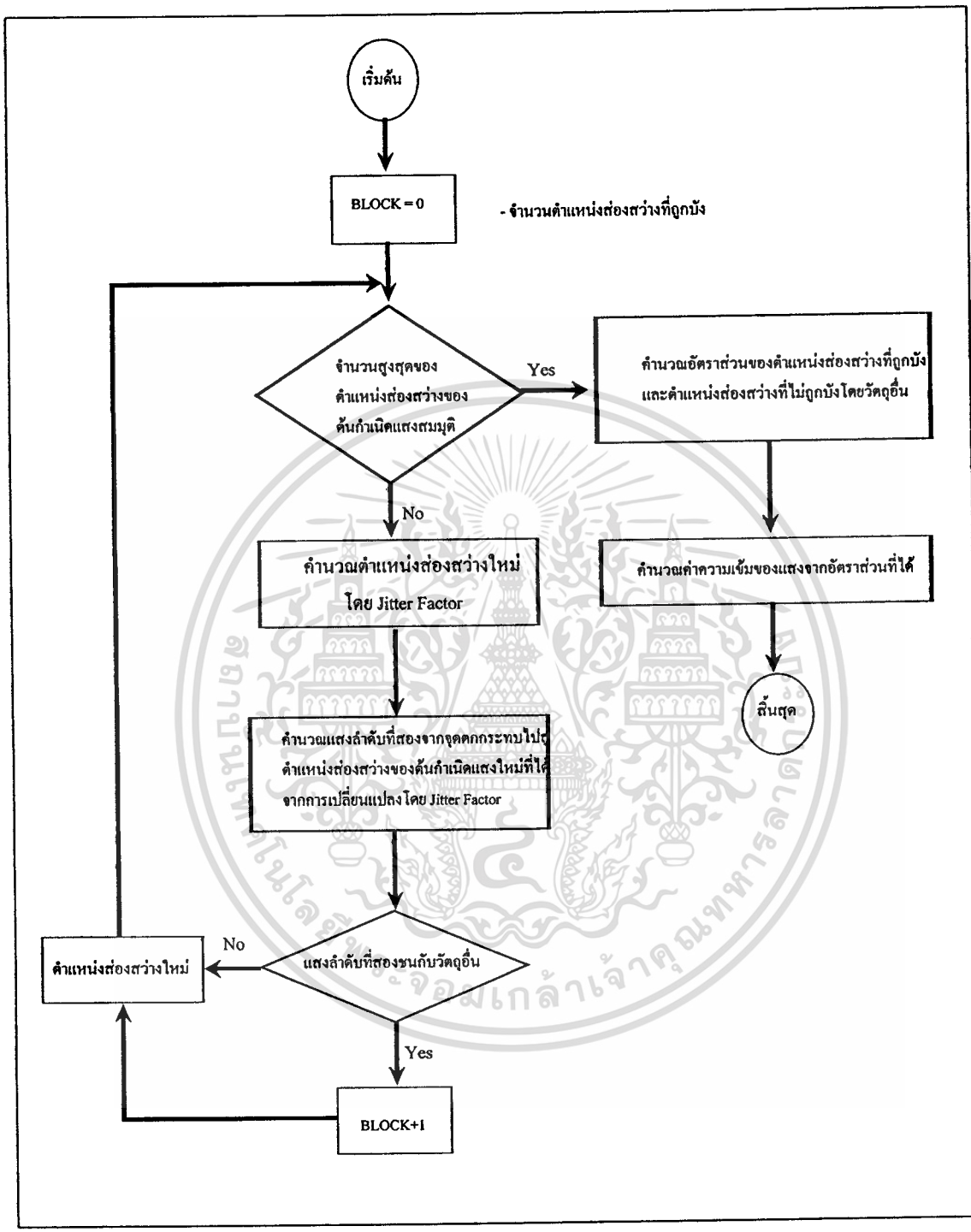


แสดง ภาพจำลองเมื่อ $\delta = 40$, $r = 25$, $\theta = 45$, Zone = 2, Jitter factor = 0.50

ภาพที่ 112

แสดง ภาพจำลองเมื่อ $\delta = 40$, $r = 25$, $\theta = 45$, Zone = 2, Jitter factor = 0.75

ภาพที่ 113



แสดง การสร้างภาพโดยการใช้ Jitter factor

การคำนวณค่าความเข้มของแสงของเงาของวัตถุในระบบ

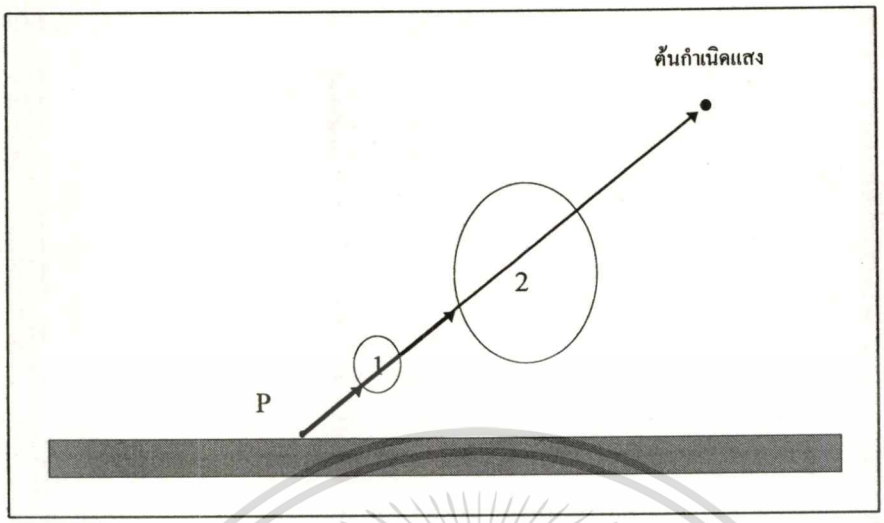
เงาที่เกิดขึ้นนั้นจะเป็นผลมาจากบริเวณที่เกิดเงาจะเป็นบริเวณที่ไม่ได้รับแสงจากต้นกำเนิดแสง เนื่องจากมีวัตถุอื่นๆ มาบดบังแสงจากต้นกำเนิดแสงอยู่ ในการสร้างภาพจำลองของวัตถุที่ในระบบมีวัตถุเพียงชนิดเดียวหรือในระบบที่มีการสร้างเงาแบบขอบแข็งจะไม่เกิดข้อผิดพลาดจากการที่วัตถุในระบบมีการซ้อนกันหรือลักษณะการซ้อนกันของเงาจากวัตถุที่ซ้อนกัน เนื่องจากการสร้างเงาแบบขอบแข็ง ค่าความเข้มของแสงของเงาที่ได้จะมีค่าความเข้มของแสงเท่ากันเสมอ

ในระบบที่มีวัตถุจำนวนมากนั้นวัตถุต่าง ๆ อาจมีตำแหน่งที่ซ้อนๆกันเสมอและจากวิธีการสร้างภาพจำลองนั้นในส่วนของเงาจะถูกกำหนดโดย ตรวจสอบว่าที่ตำแหน่งนั้นลำแสงสมมุติจากจุดที่พิจารณาไปสู่ต้นกำเนิดแสงจะถูกบดบังโดยวัตถุใดๆหรือไม่ การตรวจสอบโดยวิธีนี้จะทำให้เกิดข้อผิดพลาดขึ้น จากภาพที่ 114 จะพบว่าเมื่อทำการตรวจสอบตำแหน่งที่ต้องการ P ว่าเป็นเงาหรือไม่ ผลของการตรวจสอบจะเป็นจริง เนื่องจากลำแสงสมมุตินั้นจะกระทบกับวัตถุที่ 1 ก่อน เพราะค่าเวลาที่ใช้ในการเดินทางจะมีค่าน้อยที่สุด แต่ที่ตำแหน่งที่ต้องการนี้ก็จะมีผลของเงาที่เกิดจากวัตถุที่ 2 ด้วย ในการจำลองภาพโดยวิธีสร้างเงาแบบเรียบจะทำให้เห็นข้อผิดพลาดได้อย่างชัดเจน ภาพที่ 115 แสดงการเกิดเงาแบบขอบแข็ง เมื่อมีวัตถุในระบบเป็นจำนวนมาก ภาพที่ 116 แสดงการเกิดเงาแบบเรียบ ภาพที่ 117 แสดงการเกิดเงาแบบขอบแข็ง โดยตัววัตถุเองและการเกิดเงาแบบเรียบ โดยวัตถุอื่น ภาพที่ 118 แสดงการเกิดเงาแบบเรียบ โดยตัววัตถุเองและผลรวมของการเกิดเงาแบบเรียบ โดยวัตถุอื่น ภาพที่ 119 แสดงการเกิดเงาแบบเรียบ โดยตัวเองจากวัตถุที่ใกล้ที่สุดและค่าเฉลี่ยของเงาแบบเรียบ โดยวัตถุอื่น ภาพที่ 120 แสดงการเกิดเงาแบบเรียบบนตัวเองจากวัตถุที่ใกล้ที่สุดและค่าความเข้มของแสงที่น้อยที่สุดของเงาแบบเรียบโดยวัตถุอื่น และภาพที่ 121 แสดงการเกิดเงาแบบเรียบ ที่มีค่าความเข้มของแสงน้อยที่สุด

เวลาที่ใช้ในการสร้างภาพจำลอง

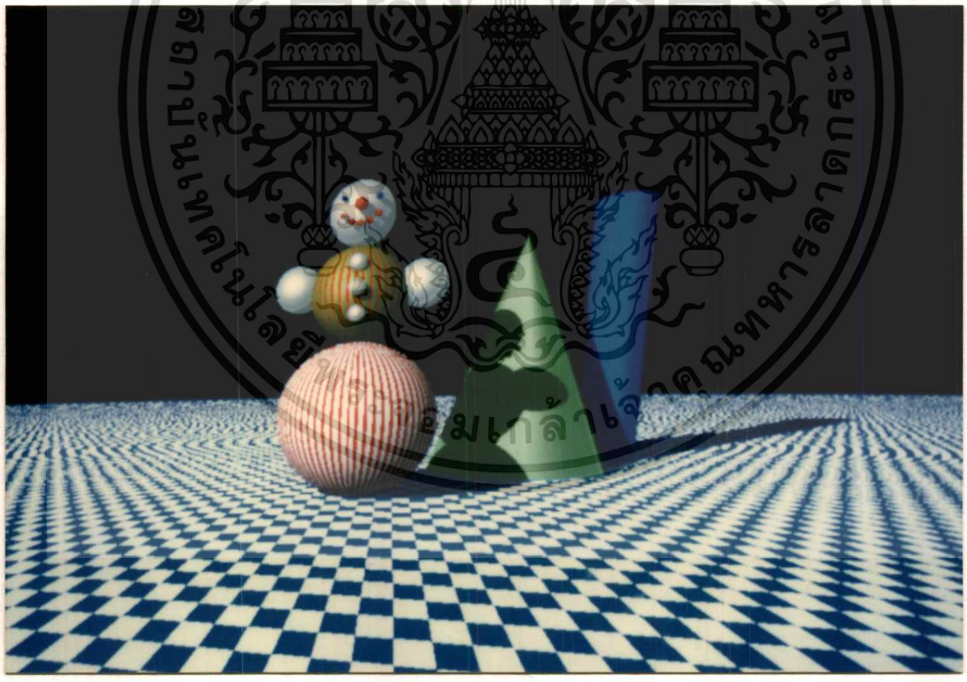
เวลาที่ใช้ในการสร้างภาพจำลองจะมีค่าที่แตกต่างกันขึ้นอยู่กับลักษณะของระบบที่ถูกกำหนดขึ้นเช่น จำนวนของวัตถุในระบบหรือวิธีการต่างๆในการสร้างภาพจำลอง เวลาที่ใช้ในการสร้างภาพแสดงในตารางที่ 2 และวิธีการสร้างภาพรวมแสดงไว้ในภาพที่ 122

ภาพที่ 114



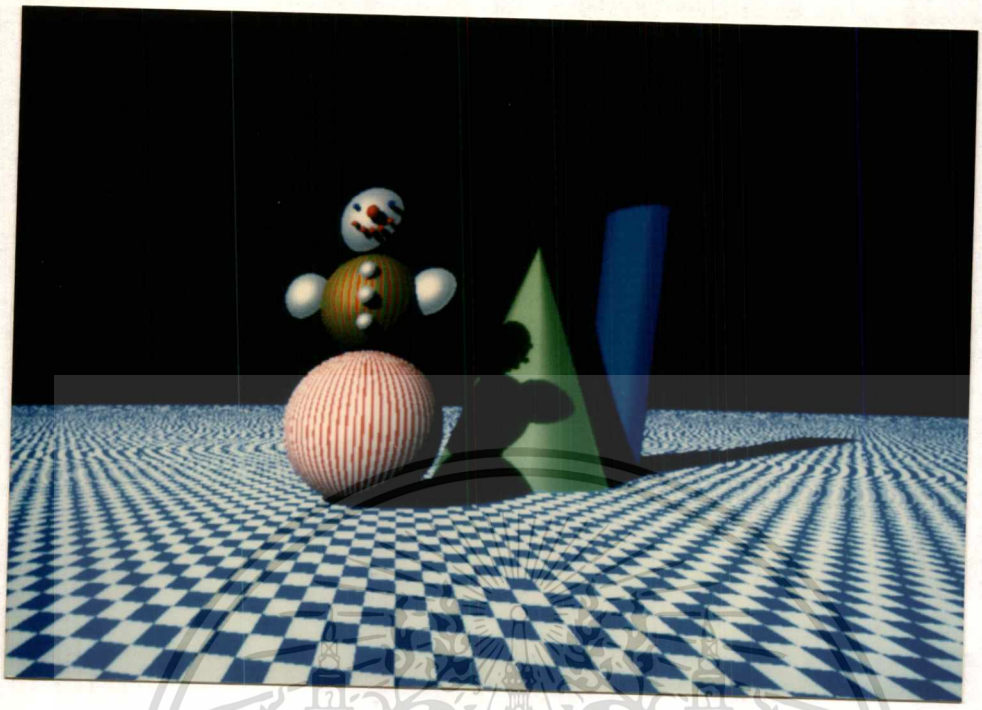
แสดง การเกิดเงาของวัตถุ

ภาพที่ 115



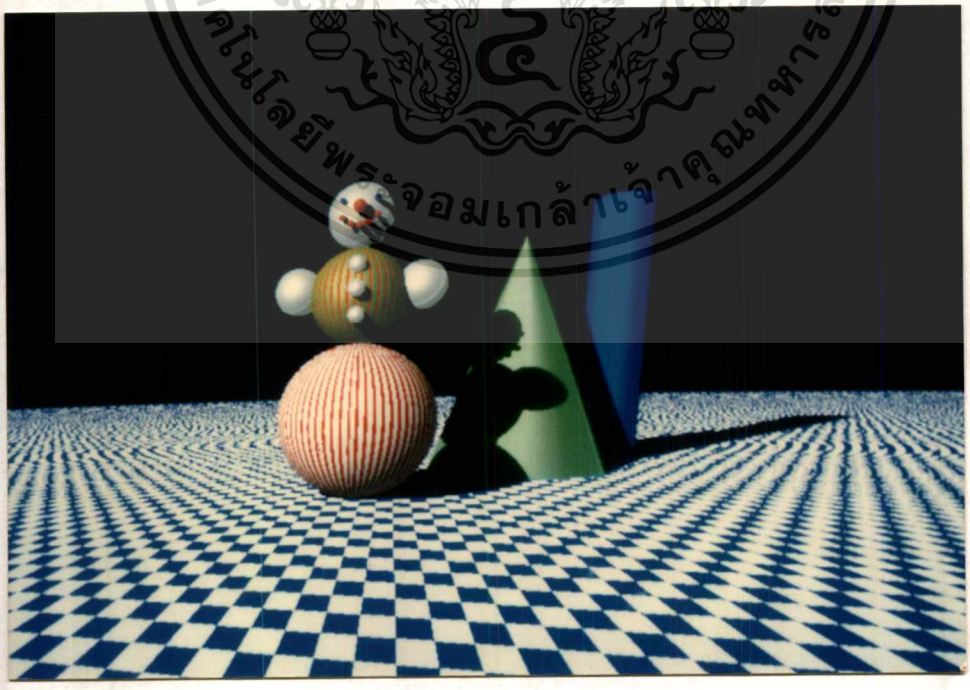
แสดง การเกิดเงาแบบขอบแข็ง

ภาพที่ 118



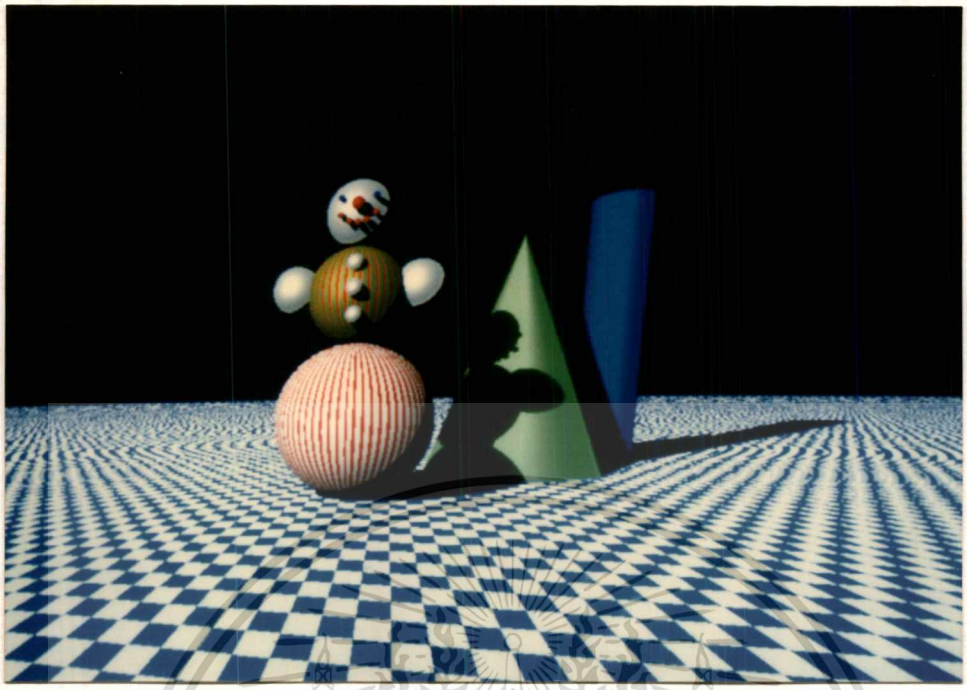
แสดง การเกิดเงาแบบเรียบโดยตัววัตถุเองและผลรวมของการเกิดเงาแบบเรียบโดยวัตถุอื่น

ภาพที่ 119



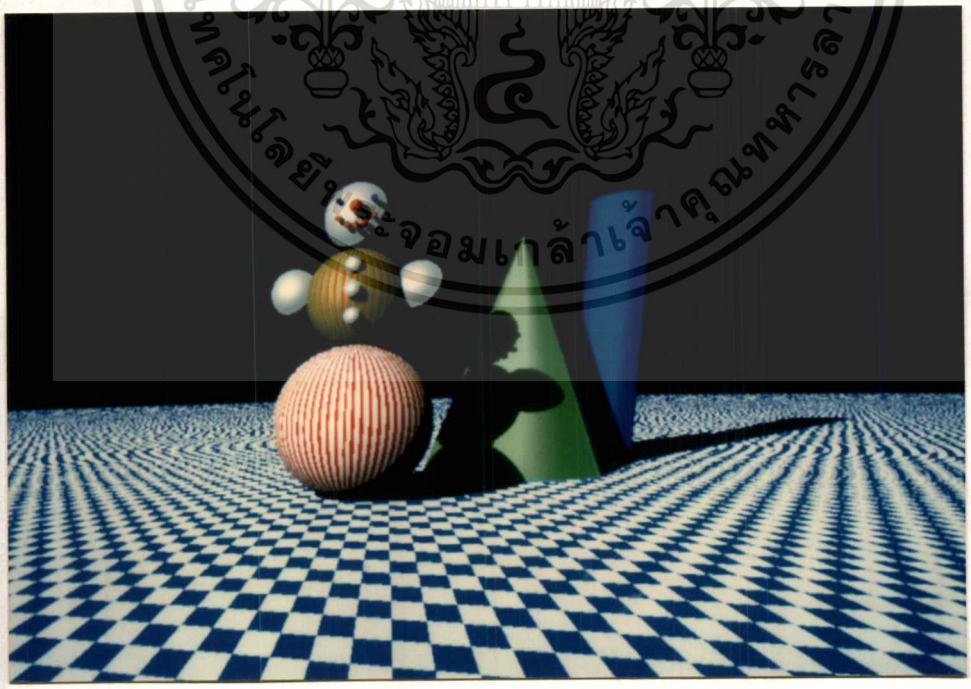
แสดง การเกิดเงาแบบเรียบโดยตัวเองจากวัตถุที่ใกล้ที่สุดและค่าเฉลี่ยของเงาแบบเรียบโดยวัตถุอื่น

ภาพที่ 120



แสดง การเกิดเงาแบบเรียบบนตัวเองจากวัตถุที่ใกล้ที่สุดและ
ค่าความเข้มของแสงที่น้อยที่สุดของเงาแบบเรียบโดยวัตถุอื่น

ภาพที่ 121



แสดง การเกิดเงาแบบเรียบ ที่มีค่าความเข้มของแสงน้อยที่สุด

ตารางที่ 2

ลำดับ	ภาพ	เวลา(นาที)
1	64	8.77
2	65	13.28
3	66	12.6
4	78	9.42
5	79	9.43
6	83	13.82
7	84	13.82
8	87	12.88
9	88	14.98
10	89	14.95
11	90	18.05
12	91	12.90
13	93	17.42
14	94	21.77
15	95	21.05
16	96	27.26
17	97	17.40
18	102	12.32
19	103	12.32
20	104	12.32
21	105	12.30
22	106	14.05
23	107	14.12
24	108	12.30
25	110	12.32
26	111	12.30
27	112	12.32

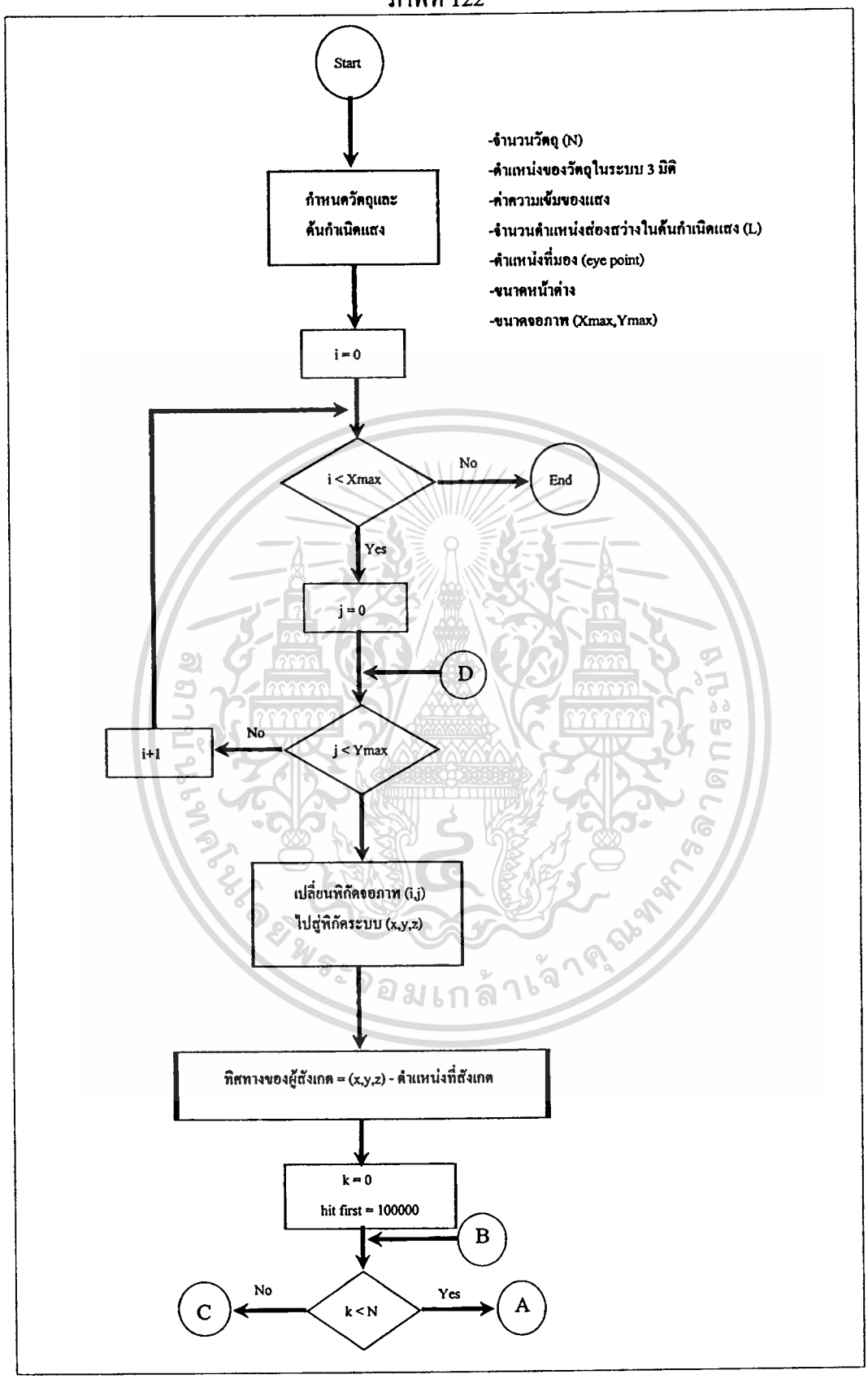
แสดง เวลาที่ใช้ในการสร้างภาพจำลองโดยวิธี Ray Tracing

ตารางที่ 2 (ต่อ)

ลำดับ	ภาพ	เวลา(นาที)
28	115	19.28
29	116	21.29
30	117	19.58
31	118	20.65
32	119	19.48
33	120	19.60
34	121	19.67

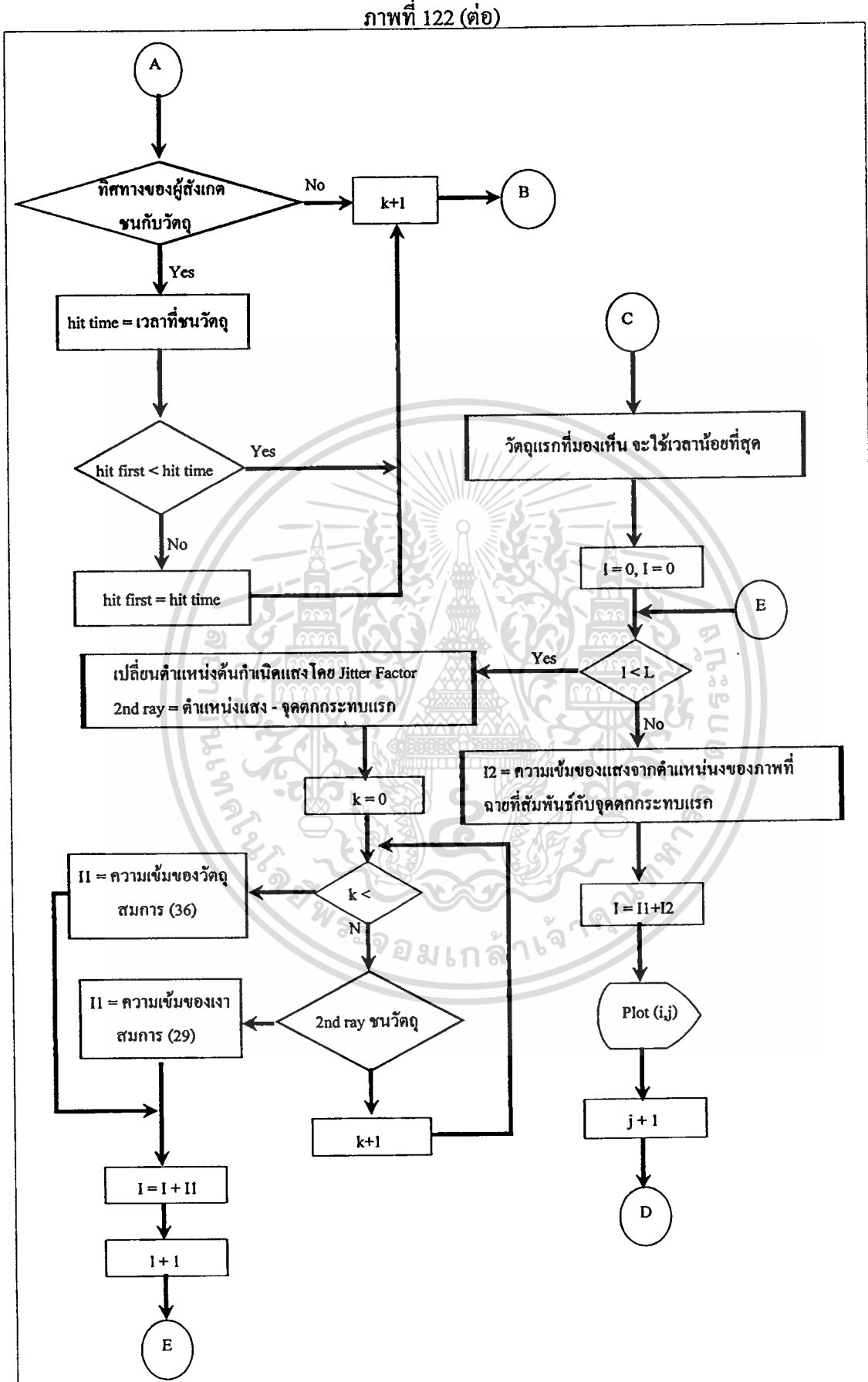
หมายเหตุ : ภาพจำลองมีขนาด 1024 x 768 จุด

ภาพที่ 122



แสดง ขั้นตอนการสร้างภาพจำลองโดยวิธี Ray tracing

ภาพที่ 122 (ต่อ)



สรุป

จากการทดลองพบว่าการสร้างภาพจำลองโดยวิธี Ray tracing นั้น ภาพที่ได้จะมีความเหมือนจริงมากขึ้นเนื่องมาจากการเรียงตัวของค่าความเข้มของแสงที่ปรากฏบนภาพจำลองทำให้มองเห็นว่าลักษณะพื้นผิวของวัตถุมีความต่อเนื่องกัน ส่วนการนำเรื่องการสร้างเงาแบบเรียบมาใช้ในการแก้ไขข้อผิดพลาดเรื่องของเงารวมถึงการใช้วิธีของ Jittering และการจัดส่วนของเงาจะทำให้ภาพจำลองที่ได้มีลักษณะที่ดีขึ้นมากกว่าเดิม จากตารางที่ 2 นั้นทำให้ทราบว่าในการจำลองภาพเหมือนจริงโดยวิธี Ray tracing นั้น เวลาที่ใช้ในการสร้างภาพแต่ละภาพจะขึ้นอยู่กับความซับซ้อนของระบบที่สร้างหรือกำหนดขึ้น ไม่ว่าจะเป็นจำนวนวัตถุ จำนวนต้นกำเนิดแสงหรือ วิธีการคำนวณต่างๆ ในกรณีที่ต้องการความถูกต้องของภาพจำลองให้มีมากขึ้นก็จะทำให้เวลาในการประมวลผลมากตามไปด้วยจากความซับซ้อนของระบบ ดังนั้นในการสร้างภาพจำลองจึงควรที่จะกำหนดระดับของรายละเอียดของภาพจำลองที่ได้ว่าต้องการให้อยู่ในระดับใด

บทที่ 7
สรุปและวิจารณ์

ในการสร้างภาพจำลองที่เหมือนจริงนั้น สามารถแยกออกได้เป็นสองส่วนด้วยกันตามวิธีที่ใช้ในการสร้างภาพ คือ การสร้างภาพจำลองโดยวิธี Radiosity และการสร้างภาพจำลองแบบ Ray Tracing การสร้างภาพจำลองทั้งสองวิธีนั้นให้ผลที่แตกต่างกัน การจำลองภาพแบบ Radiosity จะให้ผลดีกว่าเมื่อวัตถุอยู่ในระบบปิดแต่ความละเอียดของภาพจะขึ้นอยู่กับการแบ่งตัวของพื้นผิวของวัตถุ ในกรณีที่วัตถุถูกแบ่งออกเป็นพื้นที่เล็ก ๆ จำนวนมาก ภาพที่ได้ก็จะละเอียดตามไปด้วย แต่เมื่อใช้การจำลองภาพโดยวิธีนี้บนเครื่องไมโครคอมพิวเตอร์ทั่วไปก็จะมีปัญหาตามมาในเรื่องหน่วยความจำไม่พอที่จะใช้ในการคำนวณ ส่วนการจำลองภาพโดยวิธี Ray Tracing นั้นจะใช้หน่วยความจำน้อยกว่าแต่ในทางตรงกันข้ามเวลาที่ใช้ในการคำนวณค่าความเข้มของแสงก็จะมากขึ้นด้วย

ในการจำลองภาพแบบ Ray Tracing นั้นโดยทั่วไปจะกำหนดต้นกำเนิดแสงให้เป็นต้นกำเนิดแสงแบบจุด ซึ่งเป็นผลทำให้เงาของวัตถุในภาพจำลองมีลักษณะเป็นเงาแบบขอบแข็ง แต่จากการกระจายตำแหน่งส่องสว่างในต้นกำเนิดแสงออกเป็นหลาย ๆ ตำแหน่งสามารถทำให้เงาที่เกิดขึ้นกับวัตถุในภาพจำลองมีลักษณะเป็นเงาแบบเรียบซึ่งเป็นลักษณะของเงาที่เกิดขึ้นจริงกับวัตถุในธรรมชาติ การกระจายตำแหน่งส่องสว่างของต้นกำเนิดแสง โดยกระจายตามมุมรอบจุดศูนย์กลางและแนวรัศมีทั้งสองแบบทำให้การเรียงซ้อนกันของเงาแบบเรียบที่ได้มีความเป็นระเบียบมากขึ้น รวมทั้งการกระจายค่าความเข้มของแสงของต้นกำเนิดแสงตามแบบของเกาส์ จะทำให้ค่าความเข้มของแสงของเงาแบบเรียบมีการเรียงตัวกันได้อย่างดีขึ้น

เงาแบบเรียบที่สว่างขึ้นตามวิธีที่กล่าวมานั้น ลักษณะของเงาที่เกิดขึ้นจะมีการเรียงตัวซ้อนกันเป็นชั้น ๆ คล้ายกับลักษณะของเงาที่เกิดจากต้นกำเนิดแสงแบบจุดหลาย ๆ ต้นกำเนิด การนำเรื่อง jitter factor มาช่วยในการสร้างภาพจะทำให้ลักษณะการซ้อนกันของเงาหายไปและเป็นผลทำให้ภาพจำลองที่ได้มีลักษณะที่เหมือนจริงมากขึ้น ลักษณะของการเลือกค่าความเข้มของเงาที่เกิดขึ้นในภาพจำลองเป็นส่วนสำคัญอีกส่วนหนึ่งในการจำลองภาพเนื่องจากในการเกิดเงาแบบขอบแข็งค่าความเข้มของแสงของเงาจะมีค่าเท่ากัน แต่ในการสร้างเงาแบบเรียบค่าความเข้มของแสงของเงาจะมีค่าไม่เท่ากัน เมื่อมีวัตถุเรียงซ้อนกันเป็นจำนวนมากการสร้างภาพจำลองจะต้องคำนึงถึงการเลือกค่าความเข้มของเงาด้วย เพื่อให้ได้ภาพจำลองที่มีลักษณะเหมือนภาพจริง

ภาพจำลองที่สร้างขึ้นนั้นใช้แบบจำลองของสีแบบ RGB จำนวนสีสูงสุด คือ 4 สีและแต่ละสีสามารถมีระดับความเข้มแตกต่างกันได้ 64 ระดับ เนื่องจากเราสามารถที่จะแสดงผลของสีได้ครั้งละ 256 สีที่แตกต่างกัน

ดังนั้นในการสร้างภาพจำลองของวัตถุที่เหมือนจริงสิ่งที่ต้องคำนึงถึง คือ ขนาดความละเอียดของภาพจำลอง การกำหนดการจัดระดับความเข้มของแสงของสีต่าง ๆ รวมถึงจำนวนของระดับความเข้มของแสงที่ใช้ด้วย และสิ่งที่สำคัญที่สุดในการสร้างภาพจำลองที่เหมือนจริง คือ วิธีการคำนวณหาค่าความเข้มของแสงที่ปรากฏบนพื้นผิวของวัตถุ และ ในส่วนที่เป็นเงาของวัตถุ วิธีการจำลองภาพของวัตถุนั้นจะสามารถทำได้เฉพาะกับค่าความเข้มของแสงที่เกิดขึ้นกับวัตถุเท่านั้นไม่สามารถที่จะกำหนดลักษณะเฉพาะของพื้นผิวของวัตถุได้ เช่น ลวดลายต่าง ๆ การนำเรื่อง Texture mapping มาใช้ร่วมกับการจำลองภาพจึงทำให้ภาพจำลองที่ได้มีลักษณะเป็นภาพที่เหมือนจริงมากขึ้น จากผลของการนำวิธีการต่าง ๆ มาใช้ช่วยในการสร้างภาพจำลองที่เหมือนจริงเหล่านี้จึงทำให้ต้องใช้เวลาในการสร้างภาพแต่ละภาพจำนวนมาก

ภาพจำลองของวัตถุที่เหมือนจริงนั้นสามารถที่จะนำไปใช้ในงานต่าง ๆ ได้มาก เช่น การจำลองลักษณะของวัตถุต้นแบบเพื่อใช้ในงานต่าง ๆ หรือการเปลี่ยนแปลงลักษณะของวัตถุในรูปร่างต่าง ๆ กันโดยใช้ภาพจำลอง แบบจำลองการส่องสว่างของวัตถุนั้นสามารถที่จะพัฒนาไปได้โดยต่อเนื่องเพื่อให้ได้ผลที่ดีขึ้นหรือใช้เวลาในการสร้างภาพให้น้อยลงการสร้างภาพเคลื่อนไหว (animation) ของวัตถุก็เป็นส่วนหนึ่งของการพัฒนาและจากความแตกต่างกันของแบบจำลองการส่องสว่างของวัตถุทั้งสองแบบนั้นสามารถที่จะนำหลักการมาพัฒนาแบบจำลองการส่องสว่างของวัตถุแบบใหม่หรือไม่ เช่น การสร้างภาพจำลองแบบ Ray tracing โดยอาศัยการคำนวณค่าความเข้มของแสงตามวิธีของ Radiosity หรือ การสร้างภาพจำลองแบบ Radiosity โดยคำนวณค่าความเข้มของแสงในทุก ๆ จุดแทนการแบ่งพื้นผิวออกเป็นชิ้นเล็ก ๆ จะเห็นได้ว่าแนวทางการพัฒนาแบบจำลองการส่องสว่างของวัตถุนั้นยังมีเรื่องต่าง ๆ อีกมากมายที่ควรจะต้องทำการศึกษาและพัฒนาต่อไปเพื่อให้ได้ภาพจำลองที่มีคุณสมบัติที่ดีขึ้น

เอกสารอ้างอิง

[1] Edward Angel, **Computer Graphics**, Addison-Wesley Publishing Company, New York, 1990.

[2] กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, เศรษพล ลิ้มปราชญา, **"การประยุกต์การแสดงผลภาษาไทยกับกราฟิกมอดูล"**, เอกสารการประชุมวิชาการ วิศวกรรมไฟฟ้า ประจำปี 2534, หน้า 89-97.

[3] James D. Foley, Andries Van Dam, Steven K. Feiner, John F. Hughes and Richard L. Philips, **Introduction to Computer Graphics**, Addison-Wesley Publishing Company, New York, 1994.

[4] กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"คอมพิวเตอร์กราฟิก 2 มิติ"**, บิซิเนส คอมพิวเตอร์ แมกะซีน, ปีที่ 4, ฉบับที่ 41, กรกฎาคม 2535, หน้า 195-202.

[5] Steven Harrington, **Computer Graphics a Programming Approach**, second edition, Singapore, McGraw-Hill International editions, 1987.

[6] William M. Newman and Robert F. Sproull, **Principles of Interactive Computer Graphics**, second edition, Singapore, McGraw-Hill International editions, 1981.

[7] David F. Rogers and J. Alen Adams, **Mathematical Elements For Computer Grapahics**, second edition, Singapore, McGraw-Hill International editions, 1990.

[8] กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"คอมพิวเตอร์กราฟิก 3 มิติ"**, บิซิเนส คอมพิวเตอร์ แมกะซีน, ปีที่ 5, ฉบับที่ 50, เมษายน 2536, หน้า 263-267.

[9] David Pinedo, **"Window Clipping Methods in Graphics Accelerators"**, IEEE Computer Graphics and Application, May, 1991, pp. 75-84.

[10] James F. Blinn, **"A Trip Down the Graphics Pipeline: Grandpa, What Does "Viewport" Mean?"**, IEEE Computer Graphics and Application, January, 1992, pp. 83-87.

[11] David F. Rogers, **Procedural Elements For Computer Graphics**, Singapore, McGraw-Hill International Editions, 1985.

[12] Donald Hearn and M. Pauline Baker, **Computer Graphics**, USA., Prentice-Hall, Inc., 1986.

[13] F.S. Hill, JR., **Computer graphics**, New York, Maxwell Macmillan International Editions, 1990.

[14] Bruce Mielke, **Integrated Computer Graphics**, St. Paul, West Publishing Company, 1991.

[15] กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"Light and Color Model แสงและแบบจำลอง ของสี"**, วารสารสมาคมคอมพิวเตอร์แห่งประเทศไทย ในพระบรมราชูปถัมภ์, ปีที่ 20, ฉบับที่ 105, มกราคม-กุมภาพันธ์, 2537, หน้า 58-66.

[16] James D. Foley, Andries Van Dam, Steven K. Feiner and John F. Hughes, **Computer Graphics Principles and Practice**, second edition, Singapore, Addison-Wesley Publishing Company, 1990.

[17] Gary W. Meyer and Donald P. Greenberg, **"Color-Defective Vision and Computer Graphics Displays"**, IEEE Computer Graphics and Application, September, 1988, pp. 28-40.

[18] Haim Levkowitz and Gabor T. Herman, **"Color Scales for Image Data"**, IEEE Computer Graphics and Application, January, 1992, pp. 72-80.

[19] James F. Blinn, **"NTSC:Nice Technology, Super Color"**, IEEE Computer Graphics and Application, March, 1993, pp.17-23.

[20] James D. Foley and Jack Grimes, **"Using Color in Computer Graphics"**, IEEE Computer Graphics and Application, September, 1988, pp. 25-27.

[21] กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"แสงและการจัดลำดับความเข้ม"**, วารสารสมาคมคอมพิวเตอร์แห่งประเทศไทย ในพระบรมราชูปถัมภ์, ปีที่ 20, ฉบับที่ 102, กรกฎาคม-สิงหาคม, 2536, หน้า 27-34.

[22] Philip K. Robertson, **"Visualizing Color Gamuts: A User Interface for the Effective Use of Perceptual Color Spaces in Data Displays"**, IEEE Computer Graphics and Application, September, 1988, pp. 50-64.

[23] Maria Lurdes Dias, **"Ray Tracing Interference Color"**, IEEE computer Graphics and Application, March, 1991, pp. 54-60.

[24] Maria Lurdes Dias, **"Ray Tracing Interference Color: Visualizing Newton's Rings"**, IEEE computer Graphics and Application, May, 1994, pp. 17-20.

[25] Daneil R. Baum and James M. Winget, **"Real Time Radiosity Through Parallel Processing and Hardware Acceleration"**, Computer Graphics, Volumn 24, Number 2, March, 1990, pp. 67-75.

[26] Michael F. Cohen and John R. Wallace, **Radiosity and Realistic Image Synthesis,** Academic Press Professional, New York, 1993.

[27] Steven Gortler, Michale F. Cohen and Philipp Slusailele, **"Radiosity and Relaxation Methods"**, IEEE Computer Graphics and Applications, November, 1994, pp. 48-58.

[28] จเร สุรวัฒน์ปัญญา, **การคำนวณเชิงตัวเลขด้วย BASIC**, บริษัท ซีเอ็ดยูเคชั่น จำกัด, กรุงเทพ, 2531.

[29] Peter Schroder and Pat Hanrahan, **"On The Form Factor Between Two Polygons"**, Computer Graphics Proceedings, 1993, pp. 163-164.

[30] Dani Lischinski, Filippo Tampieri and Donald P. Greenberg, **"Discontinuity Meshing for Accurate Radiosity"**, IEEE Computer Graphics and Applications, November, 1992, pp. 25-39.

[31] Alan Watt, **Fundamentals of Three Dimensional Computer Graphics,** Addison-Wesley Publishing Company, 1989.

[32] Andrew S. Glassner, **"Ray Tracing for Realism"**, Byte, December, 1990, pp. 263-271.

[33] Andrew S. Glassner, **An Introduction to Ray Tracing**, Academic Press Limited, 1989.

[34] David Halliday, Robert Resnick and Jearl Walker, **Fundamentals of Physics,** Fourth edition, John Wiley & Sons INC., New York, 1993.

[35] Robert L. Cook, Thomas Porter and Loren Carpenter, **"Distributed Ray Tracing"**, Computer Graphics Proceedings, Volumn 18, Number 3, July, 1984, pp. 137-145.

[36] Alain Mangen, **"RAY: A Ray-Tracing Program in C++"**, Dr Dobb's Journal, July, 1994, pp. 40-43.

[37] Didier Badouel, Kadi Bouatouch and Thierry Priol, **"Distributing Data and Control for Ray Tracing in Parallel"**, IEEE Computer Graphics and Application, July, 1994, pp. 69-77.

[38] Roni Yagel, Daniel Cohen and Arie Kanfman, **"Discrete Ray Tracing"**, IEEE Computer Graphics and Application, September, 1992, pp. 19-28.

[39] Craig A. Lindley, **"Ray tracing and POV-Ray Toolkits"**, Dr Dobb's Journal, July, 1994, pp. 68-76.

[40] กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"Objects' Illumination Model in Computer Graphics by Ray Tracing, แบบจำลองการส่องสว่างของวัตถุในคอมพิวเตอร์กราฟิกโดย Ray Tracing"**, วารสารสมาคมคอมพิวเตอร์แห่งประเทศไทย ในพระบรมราชูปถัมภ์, ปีที่ 20; ฉบับที่ 109, กันยายน-ตุลาคม, 2537, หน้า 57-64.

[41] Takaaki Akimoto, Kenji Mase and Yasuhito Suenaga, **"Pixel-Selected Ray Tracing"**, IEEE Computer Graphics and Application, July, 1991, pp. 14-22.

[42] Paul Pitot, **"The Voxar Project"**, IEEE Computer Graphics and Application, January, 1993, pp. 27-33.

[43] Takami Yasuda, Shigeki Yokoi, Jun-Ichiro Toriwaki and Katsuhiko Inagaki, **"A Shading Model for Cloth Objects"**, IEEE Computer Graphics and Application, November, 1992, pp. 15-24.

[44] Andrew J. Hanson and Pheng A. Heng, **"Illumination The Fourth Dimension"**, IEEE Computer Graphics and Application, July, 1992, pp. 54-62.

[45] Alan Watt and Mark Watt, **Advanced Animation and Rendering Techniques : Theory and Practice,** Addison - Wesley Publishing Company, New York, 1992

[46] Chris Schoeneman, Julie Dorsey, Brian Smits, James Arvo and Donald Greenberg, **"Painting with Light"**, Computer Graphics Proceedings, August, 1993, pp. 143-146.

[47] Psul Hseberli, **"Fast Shadows and Lighting Effects Using Texture Mapping"**, Computer Graphics Proceedings, Volumn 26, Number 2, July, 1992, pp. 249-252.

[48] Kevin P. Picott, **"Extensions of The Linear and Area Lighting Models"**, IEEE Computer Graphics and Application, March, 1992, pp. 31-38.

[49] Seth J. Teller, **"Computing The Antipenumbra of an Area Light Source"**, Computer Graphics Proceedings, Volumn 26, Number 2, July, 1992, pp. 139-148.

[50] Andrew Woo, **"Efficient Shadow Computations in Ray Tracing"**, IEEE Computer Graphics and Application, September, 1993, pp. 78-83.

[51] Philip R. Bevington, **Data Reduction and Error Analysis for the Physical Sciences,** McGraw - Hill Book Company, New York, 1969.

[52] Benoit B. Mandelbrot, **The Fractal Geometry of Nature**, W. H. Freeman and Company, New York, 1983.

[53] Robert L. Devaney, **A First Course in Chaotic Dynamical Systems Theory and Experiment**, Addison-Wesley Publishing Company, Inc., second Printing, Canada, 1993.

[54] Michael Barnsley, **Fractals Everywhere**, Academic Press, Inc. San Diego, 1988.

[55] กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"The Beauty of Fractal Images"**, บิซิเนส คอมพิวเตอร์ แมกะซีน, ปีที่ 3, ฉบับที่ 35, มกราคม, 2535, หน้า 125-129.

[56] H. O. Peitgen and P. H. Richter, **The Beauty of Fractals Images of Complex Dynamical Systems**, Springer-Verlag, Berlin Heidelberg, Germany, 1986.

[57] Jeremy Spiller, **"3-D Texture Mapping"**, Dr. Dobb's Journal, July, 1994, pp. 32-37.

[58] Jerome Maillot, Hussein Yahia and Anne Verroust, **"Interactive Texture Mapping"**, Computer Graphics Proceedings, 1993, pp. 26-34.

[59] กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"ภาพและรายละเอียดพื้นผิวของวัตถุ"**, วารสารสมาคมคอมพิวเตอร์แห่งประเทศไทย ในพระบรมราชูปถัมภ์, ปีที่ 21, ฉบับที่ 113, พฤษภาคม-มิถุนายน, 2538, หน้า 29-36.

ภาคผนวก ก

Fractal Images

ขบวนการต่างๆ ที่เกิดขึ้นในระบบพลวัต (dynamical sytems) นั้น เป็นขบวนการที่ไม่สามารถกำหนดได้อย่างแน่นอนว่าอะไรจะเกิดขึ้นในระบบนั้นบ้าง[52] ตัวอย่างของ dynamical sytems เช่น ระบบทางเคมีต่าง ๆ ระบบของภูมิอากาศ หรือ ระบบของการเพิ่มขึ้นหรือลดลงของประชากร ขบวนการที่เกิดขึ้นในระบบนี้มีทั้งที่สามารถประเมินผลได้และขบวนการที่ไม่สามารถประเมินผลได้ ขบวนการที่ไม่สามารถประเมินผลได้นั้น ได้แก่ การผสมกันของสารเคมีหลายชนิดซึ่งจะทำให้เกิดการระเบิดหรือไม่ ภูมิอากาศในอีกหนึ่งปีข้างหน้าจะเป็นเช่นไร หรือจำนวนประชากรจะเป็นเท่าใดเมื่อเวลาผ่านไปหนึ่งเดือน ส่วนขบวนการที่สามารถประเมินผลได้ คือ พระอาทิตย์จะขึ้นในวันพรุ่งนี้ เมื่อผสมน้ำกับน้ำมันจะเป็นอย่างไร ขบวนการต่าง ๆ ที่เกิดขึ้นนั้น จะมีส่วนหนึ่งของขบวนการที่เกิดขึ้น ที่ไม่สามารถกำหนดได้แน่นอนว่าจะมีผลเป็นเช่นไร จากเหตุผลของการที่ไม่สามารถจะประเมินผลของขบวนการใดๆได้ นักคณิตศาสตร์จึงเรียกขบวนการนี้ว่า ระบบของความสับสน (chaos systems) ระบบของความสับสนนี้มีโอกาศที่จะเกิดขึ้นในระบบที่มีโครงสร้างแบบง่าย ๆ การศึกษาระบบของความสับสนให้เข้าใจจะทำให้สามารถนำไปประยุกต์ เพื่อให้เข้าใจพฤติกรรมของระบบที่มีโครงสร้างซับซ้อนมากขึ้น เช่น การพยากรณ์ภูมิอากาศ การพยากรณ์จำนวนประชากร

ตัวอย่างง่าย ๆ ของระบบของความสับสน คือ ถ้าต้องการที่จะศึกษาพฤติกรรมของจำนวนประชากรของสิ่งมีชีวิตชนิดหนึ่งว่าจะมีการเพิ่มขึ้นหรือลดลงเมื่อเวลาผ่านไป หรือ ในยุค (generation) อื่น ถ้ากำหนดให้ประชากรที่มีชีวิตในยุค n คือ P_n คำตอบที่ต้องการคือจะสามารถทำนายจำนวนประชากรได้หรือไม่ถ้า n มีค่ามากขึ้น P_n จะมีค่ามากขึ้นจนไม่มีขอบเขต หรือ P_n จะล้มตายไปทั้งหมด ถ้าตั้งข้อสมมุติที่ว่าจำนวนประชากรในยุคต่อไปจะขึ้นอยู่กับจำนวนประชากรที่มีชีวิตอยู่ในยุคนี้ ดังนั้น

$$P_{n+1} = rP_n \qquad \text{(a 1)}$$

เมื่อ r คือค่าคงที่

ถ้ากำหนดให้ P_0 เป็นจำนวนประชากรเริ่มต้น จำนวนประชากรในยุคที่ต้องการสามารถคำนวณหาได้โดยการคำนวนแบบการเวียนเกิด (Recursive)

$$\left.\begin{aligned} P_1 &= rP_0 \\ P_2 &= rP_1 = r^2P_0 \\ P_3 &= rP_2 = r^3P_0 \\ &\vdots \\ P_n &= rP_{n-1} = r^nP_0 \end{aligned}\right\} \qquad \text{(a 2)}$$

จากสมการ (a2) จำนวนของประชากรจะขึ้นกับค่าของ r ดังนั้นถ้า r มากกว่า 1 จะทำให้ P_n จะมีค่าเป็นอนันต์เมื่อ n มีค่ามากขึ้น ถ้า r น้อยกว่า 1 จำนวนของประชากร P_n ก็จะมีค่าเป็น 0 หรือถ้า r เท่ากับ 1 จะทำให้จำนวนประชากร P_n มีค่าไม่เปลี่ยนแปลงเลยซึ่งจากเหตุผลนี้จึงไม่สามารถที่จะเป็นจริงได้ ดังนั้นถ้ากำหนดว่าจำนวนของประชากรจะถูกกำหนดด้วยสภาพแวดล้อมด้วยโดยที่ถ้าจำนวนประชากรมีค่าถึงค่าที่กำหนดนี้จะทำให้จำนวนประชากรทั้งหมดเสียชีวิต อาจเพราะขาดอาหารหรือจำนวนประชากรเกินไป จึงกำหนดให้ P_n เป็นอัตราส่วนของจำนวนสูงสุดของประชากรที่รอดชีวิตในยุค n ดังนั้น

$$0 \leq P_n \leq 1 \qquad \text{(a 3)}$$

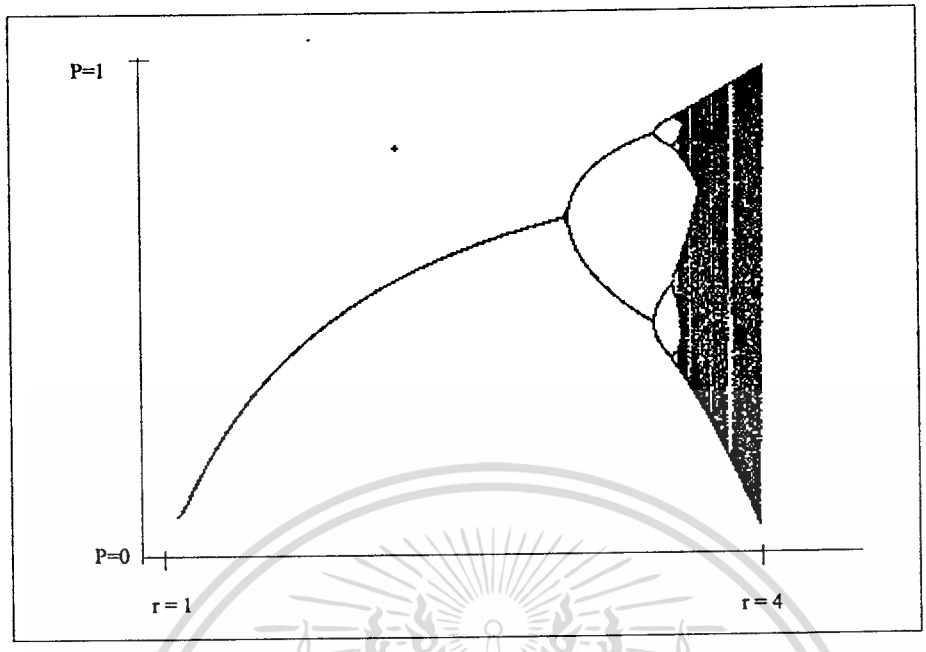
และ

$$P_{n+1} = rP_n(1 - P_n) \qquad \text{(a4)}$$

จากสมการ (a4) ยังคงรักษาแบบของจำนวนประชากรไว้ แต่มีค่าความถูกต้องมากขึ้น คือถ้า $P_n = 1$ และ $P_n = 0$ แสดงว่าประชากรทั้งหมดรอดชีวิตจะทำให้ $P_{n+1} = 0$ ซึ่งตรงตามข้อกำหนด การหาการเปลี่ยนแปลงของจำนวนประชากรสามารถกระทำได้โดย การคำนวนแบบซ้ำซ้อน (iterate) ของสมการ (a4) การคำนวนแบบซ้ำซ้อน คือ การคำนวนฟังก์ชันใดๆ ซ้ำไปมาโดยใช้ผลลัพธ์ของข้อมูลก่อนหน้าเป็นค่าเริ่มต้นสำหรับการคำนวนครั้งต่อไป[53]

จากสมการ (a4) ถ้านำมาแสดงผลในรูปของกราฟ โดยให้ P เป็นค่าในแกนตั้ง และ r เป็นค่าในแกนนอน ดังแสดงในภาพที่ a1 จากภาพพบว่าเมื่อมีการเปลี่ยนแปลงค่า r จะทำให้จำนวนของประชากรเพิ่มขึ้นเรื่อยๆ จนกระทั่ง r มีค่ามากกว่าค่าคงที่ค่าหนึ่ง จำนวนประชากรจะแยกออกเป็น 2 กลุ่ม หรือเกิดเหตุการณ์ที่เรียกว่าการแยกเป็นสองส่วน (Bifurcation) และเมื่อเพิ่มค่า r ไปเรื่อย ๆ ก็จะทำให้เกิดการแยกเป็นสองส่วนซ้ำ ๆ กันอีก

ภาพที่ a1



แสดง การแยกกลุ่มของจำนวนประชากร

ภาพ Fractal

ภาพ Fractals เป็นภาพที่มีลักษณะการซ้ำซ้อนกันของโครงสร้างในภาพ ดังนั้นไม่ว่าจะพิจารณาส่วนใดของภาพก็จะพบว่า ภาพที่เห็นนั้นมีลักษณะที่ซ้ำๆกัน แม้ว่าภาพนั้นจะถูกขยายออกไปเท่าใดก็ตาม[54] โครงสร้างของภาพที่ปรากฏก็จะมีรูปร่างหรือโครงสร้างที่คล้ายๆกันเสมอ ภาพ Fractal จะเป็นภาพที่ไม่มีรูปแบบที่แน่นอนและไม่มีที่สิ้นสุด สมการทางคณิตศาสตร์หลายๆสมการสามารถที่จะแสดงให้เห็นถึงลักษณะของ Fractal ได้

เซตของ Mandelbrot

เซตของ Mandelbrot (Mandelbrot Set) เป็นเซตของสมการทางคณิตศาสตร์ที่สามารถใช้ผลของสมการมาสร้างเป็นภาพที่มีลักษณะที่ไม่มีที่สิ้นสุดโดยอาศัยคุณสมบัติของเลขจำนวนเชิงซ้อน และการคำนวนค่าของสมการแบบเวียนเกิดมาช่วยในการสร้างภาพ[55] ซึ่งเป็นลักษณะของภาพ Fractal ภาพหนึ่ง และสมการที่ใช้ คือ

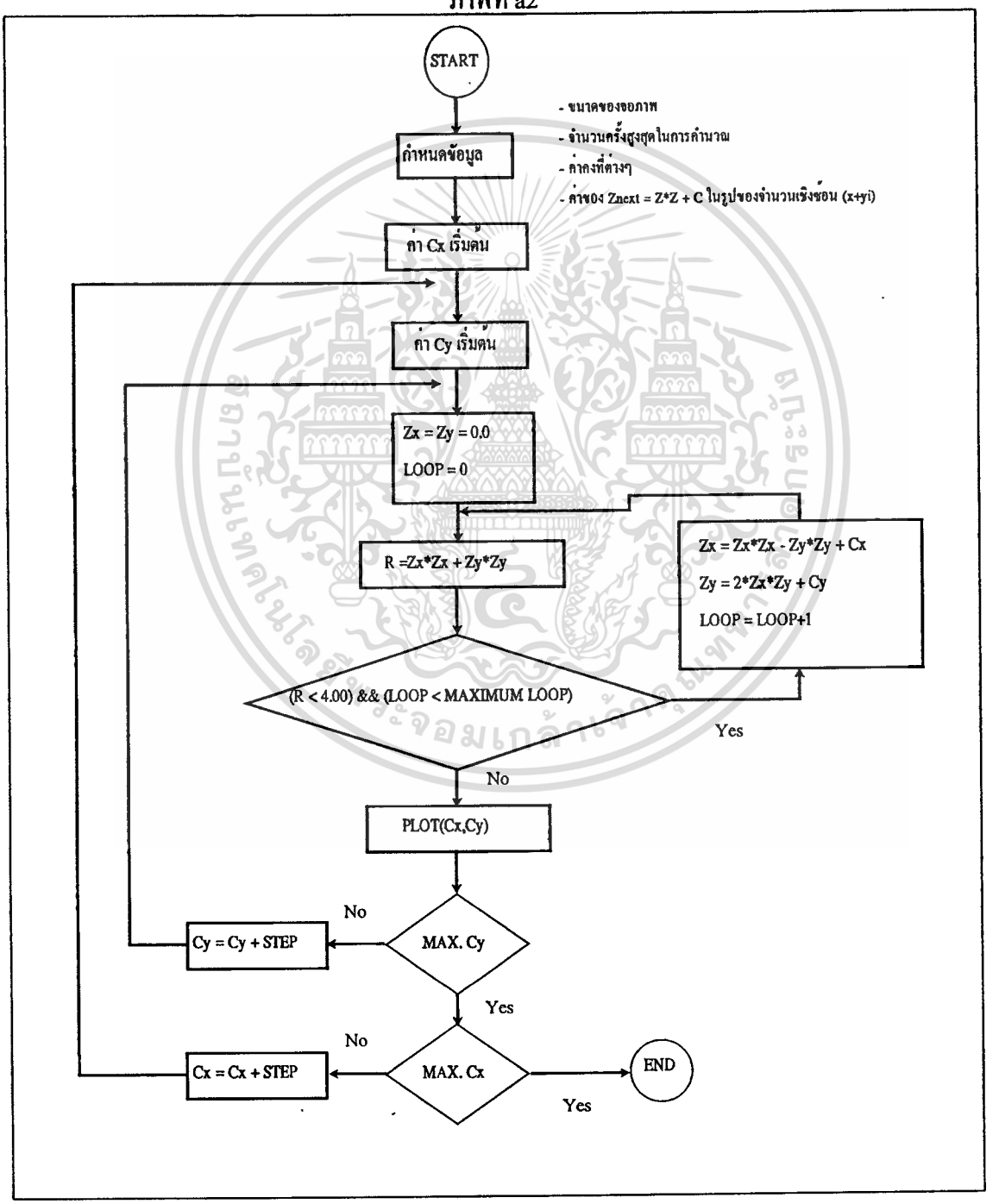
$$F(z) = Z_n^{\,2} + c = Z_{n+1} \qquad (a5)$$

เมื่อ Z และ c เป็นเลขจำนวนเชิงซ้อน a+bi และ $i^2 = -1$

เนื่องจากเมื่อ $|c| > 2$ จะทำให้ค่าของ Z_{n+1} มีค่าเข้าสู่อนันต์ซึ่งจะทำให้สมการไม่เป็นจริง ดังนั้น การพิจารณา จะพิจารณาในช่วงของ $|c| \geq 2$ เท่านั้น หลักการสร้างภาพ Fractal จากสมการ (a5) นั้น กระทำได้โดยการกำหนดค่าเริ่มต้น $Z_0 = 0$ จากนั้นจึงทำการเปลี่ยนแปลงค่า

จำนวนเชิงซ้อนของตัวแปร c แล้วทำการคำนวณแบบเวียนเกิดไปเรื่อยๆ จนกว่าจะถึงค่าที่กำหนดไว้ ภาพที่ a2 แสดงขั้นตอนการสร้างภาพ Fractal ของ Mandelbrot Set ภาพที่ได้นั้นจะแทนค่าจำนวนจริงในแกนตั้ง และค่าของจำนวนเชิงซ้อนในแกนนอน ส่วนค่าของสีที่ใช้จะได้จากจำนวนครั้งของการคำนวณแบบเวียนเกิด ภาพที่ a3, a4 และ a5 แสดงภาพที่ได้จากสมการที่ตำแหน่งและขนาดต่าง ๆ กัน

ภาพที่ a2



แสดง ขั้นตอนการสร้างภาพ Fractal ของ Mandelbrot Set

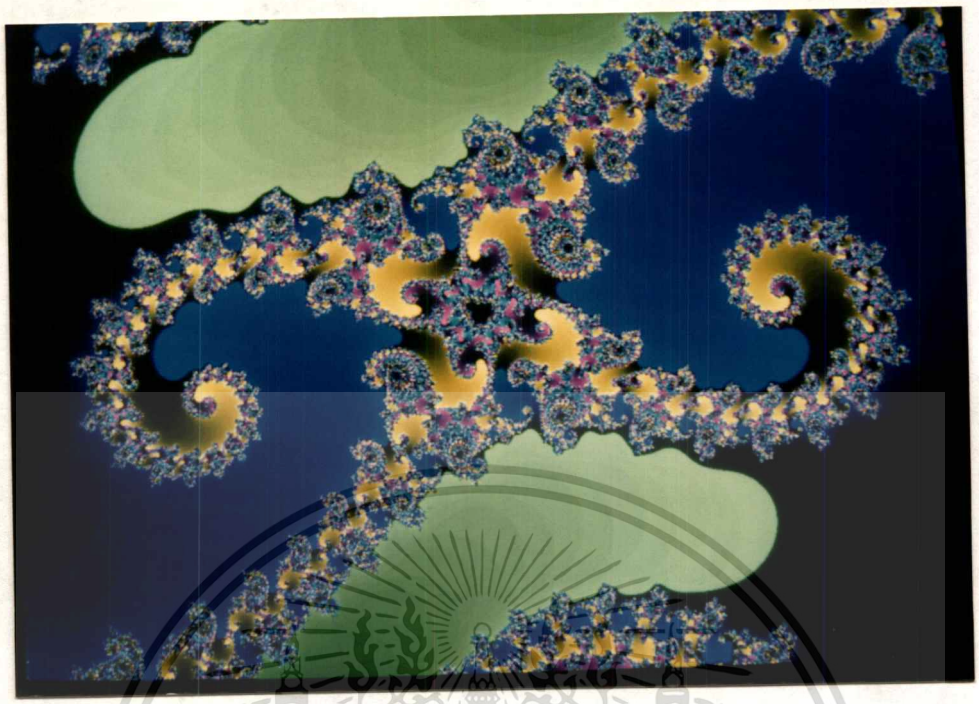
ภาพที่ a3

แสดง Mandelbrot set : Real = -2.25.....0.75 Imaginary = -1.5.....1.5

ภาพที่ a4

แสดง Mandelbrot set : Real = -0.713.....-0.4028 Imaginary = 0.49216.....0.71429

ภาพที่ a5



แสดง Mandelbrot set : Real = -0.745468.....-0.745385 Imaginary = 0.112979.....0.113039

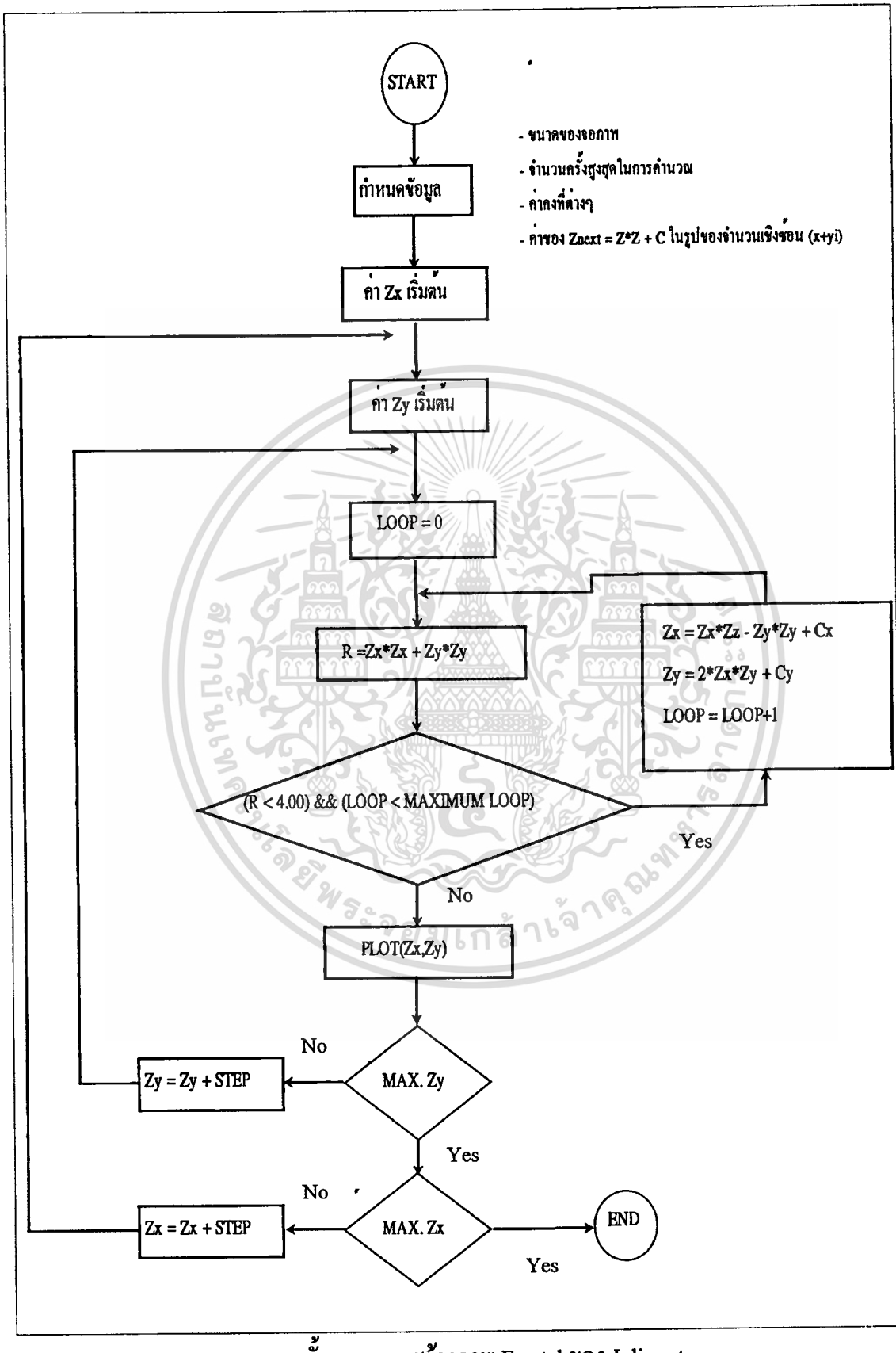
เซตของ Julia

เซตของ julia (julia set) เป็นสมการทางคณิตศาสตร์ที่สามารถสร้างภาพที่มีลักษณะเป็น fractal ด้วย โดยสมการจะเหมือนกับสมการของ Mandelbrot set สมการ (a5) แต่ขั้นตอนในการสร้างภาพนั้นจะต่างไปจากวิธีการของ Mande

lbrot set [56] โดยในวิธีการของ Mandelbrot set นั้นจะกำหนดค่าคงที่เริ่มต้นให้ $Z_0 = 0$ และเปลี่ยนแปลงค่าตัวแปร c เพื่อสร้างภาพ แต่ในวิธีการของ julia set จะกำหนดค่าคงที่เริ่มต้นให้กับตัวแปร c และเปลี่ยนแปลงค่าจำนวนเชิงซ้อนของตัวแปร Z เพื่อสร้างภาพ Fractal ภาพที่ a6 แสดงขั้นตอนการสร้างภาพโดย julia set ภาพที่ a7, a8 และ a9 แสดงภาพ Fractal ที่ได้จาก julia set

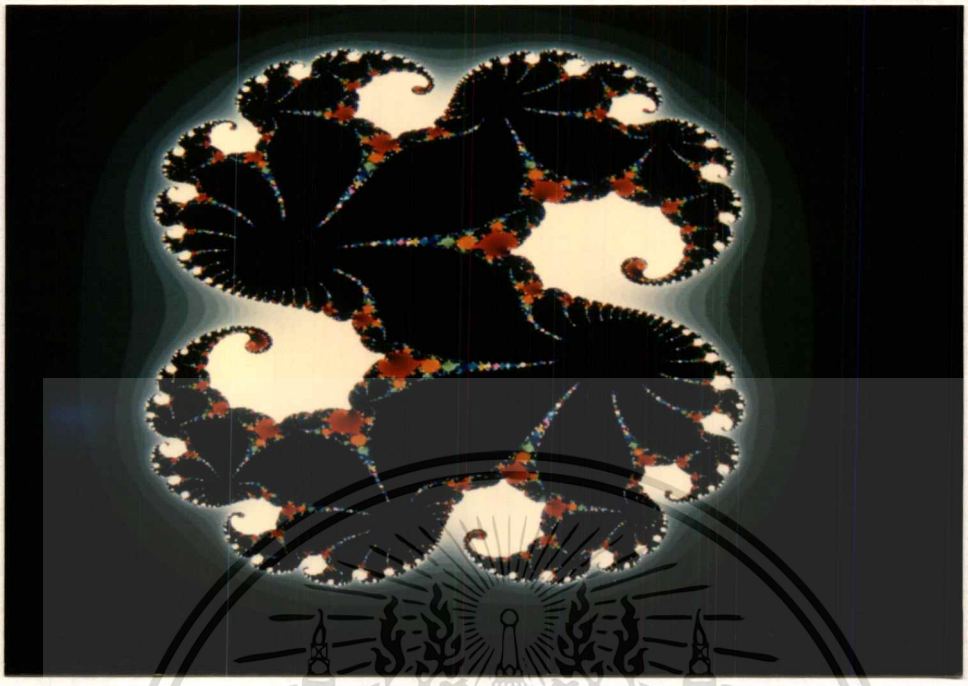
จากที่กล่าวมานั้นเป็นเพียงส่วนหนึ่งเกี่ยวกับภาพ Fractal เท่านั้น ภาพของ fractals สามารถที่จะสร้างได้จากสมการทางคณิตศาสตร์อื่น ๆ ได้อีก เช่น sine, cosine หรือ ฟังก์ชัน exponential

ภาพที่ a6



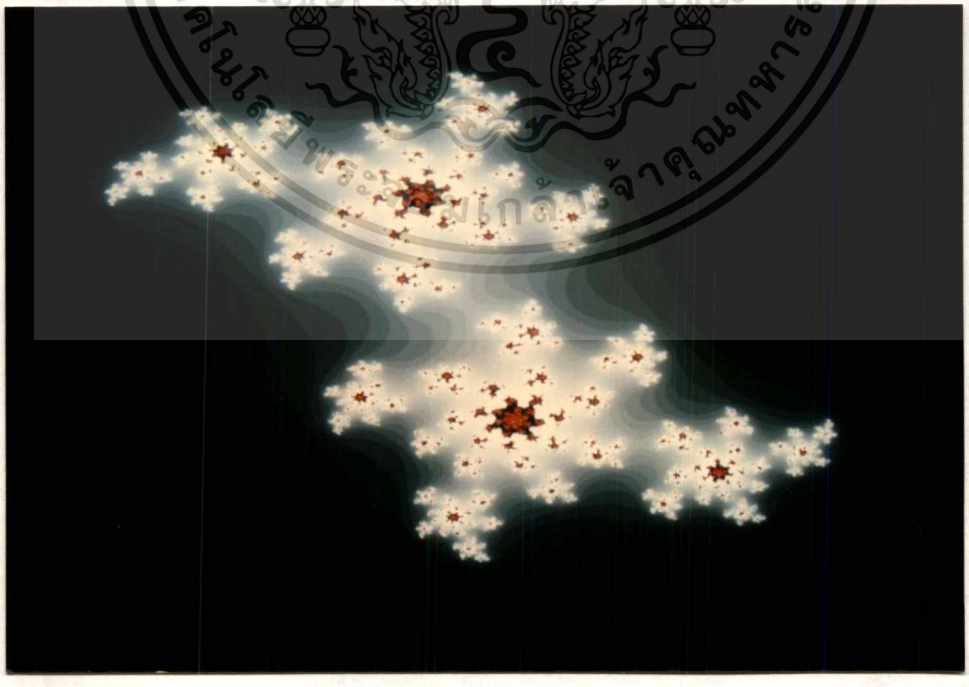
แสดง ขั้นตอนการสร้างภาพ Fractal ของ Julia set

ภาพที่ a7



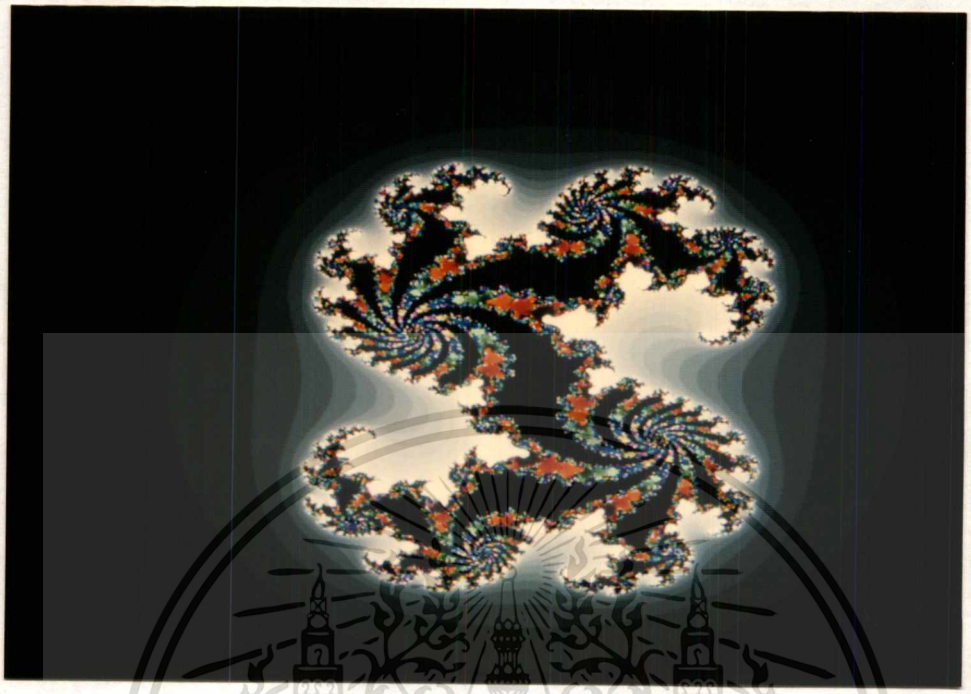
แสดง Julia set : c = 0.27334 + 0.00742i

ภาพที่ a8



แสดง Julia set : c = 0.11031 - 0.67037i

ภาพที่ a9



แสดง Julia set : c = 0.3200 + 0.0430i

ภาคผนวก ข

ความสัมพันธ์ระหว่างภาพและพื้นผิวของวัตถุ

เนื่องจากวิธีการจำลองภาพโดยวิธี Ray tracing และ Radiosity ไม่สามารถที่จะทำการจำลองลักษณะของพื้นผิวของวัตถุได้แต่เป็นการแสดงค่าความเข้มของแสงที่พื้นผิวของวัตถุเท่านั้น แต่จากความสัมพันธ์ระหว่างภาพและรายละเอียดพื้นผิวของวัตถุ (Texture mapping) [45]ซึ่งหมายถึงการฉายภาพของลักษณะที่ต้องการลงบนพื้นผิวของวัตถุใดๆ เช่นการแสดงลักษณะเฉพาะบางอย่างที่เป็นรายละเอียดของพื้นผิวของวัตถุซึ่งได้แก่ ลวดลายต่างๆที่ปรากฏอยู่บนพื้นผิวของวัตถุ ดังนั้นการนำความสัมพันธ์ระหว่างภาพและพื้นผิวของวัตถุมาใช้ร่วมกับการจำลองภาพแบบ Ray Tracing เพื่อแสดงลวดลายต่างๆของพื้นผิวในการจำลองภาพจึงทำให้ภาพที่ได้มีความเหมือนจริงมากขึ้น

หลักการของการฉายภาพบนพื้นผิวของวัตถุนั้น[57] ภาพที่จะถูกฉายไปที่พื้นผิวของวัตถุจะถูกกำหนดให้อยู่ในระบบพิกัด 2 มิติ แต่วัตถุในระบบจะถูกกำหนดในระบบพิกัด 3 มิติ เนื่องจากการแตกต่างกันของระบบพิกัดที่ใช้ ดังนั้นจึงต้องอาศัยความสัมพันธ์บางอย่างเพื่อที่จะแปลงค่าตัวแปรในระบบพิกัดทั้งสองให้สามารถใช้ร่วมกันได้ ถ้าตัวแปรที่ใช้ในการอธิบายคุณสมบัติของภาพในระบบพิกัด 2 มิติคือ (u, v) และตัวแปรที่ใช้ในการอธิบายคุณสมบัติของวัตถุในระบบพิกัด 3 มิติคือ (x, y, z) ความสัมพันธ์ของระบบพิกัดทั้งสองสามารถเขียนได้ดังนี้

$$\left.\begin{aligned} u &= f(x,y,z) \\ v &= g(x,y,z) \end{aligned}\right\} \qquad (b1)$$

หรือ

$$\left.\begin{aligned} x &= r(u,v) \\ y &= s(u,v) \\ z &= t(u,v) \end{aligned}\right\} \qquad (b2)$$

ความสัมพันธ์ระหว่างภาพและวัตถุ

การฉายภาพบนพื้นผิวของวัตถุนั้น[58]จำเป็นที่จะต้องทราบความสัมพันธ์ระหว่างระบบพิกัดทั้งสองก่อน โดยทั่วไปแล้วภาพจะถูกกำหนดในระบบพิกัด 2 มิติที่ใช้ตัวแปร 2 ตัวแปรในการอธิบายคุณสมบัติของภาพ และวัตถุจะถูกกำหนดในระบบพิกัด 3 มิติ ซึ่งการหาความสัมพันธ์ของตัวแปรต่างๆจากทั้งสองระบบพิกัดสามารถการทำได้ดังนี้ [45]

ความสัมพันธ์ของภาพและระนาบ

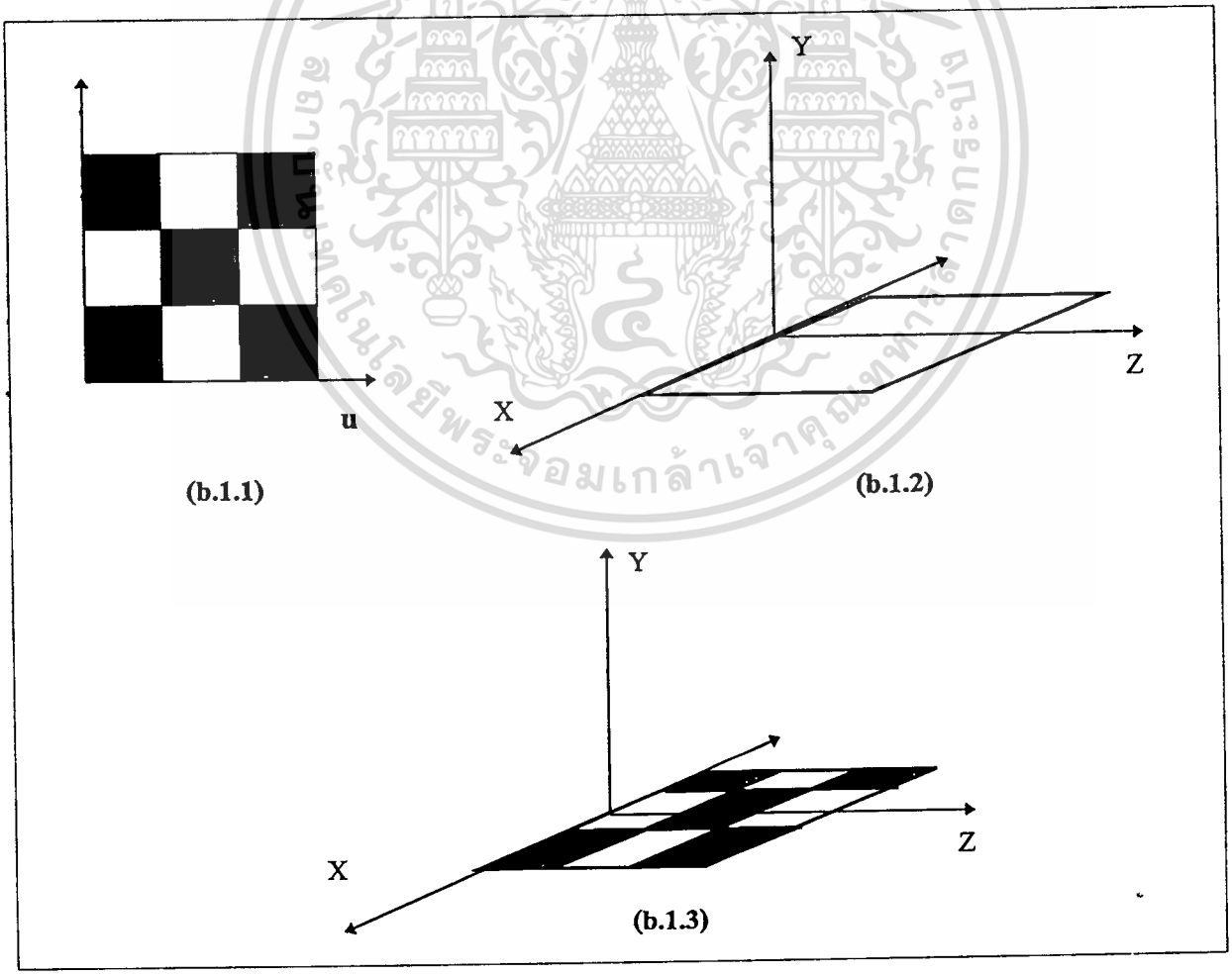
จากภาพที่ b.1.1 และ ภาพที่ b.1.2 ซึ่งแสดงลักษณะของภาพและระนาบตามลำดับ ระนาบที่ถูกกำหนดในระบบพิกัด 3 มิตินั้นจะใช้ตัวแปรเพียง 2 ค่าเท่านั้นในการสร้างภาพ จากภาพที่ b.1.2 คือตัวแปร x และ z ดังนั้นในการฉายภาพลงบนระนาบจะทำให้ได้ความสัมพันธ์ระหว่างระบบพิกัดทั้งสองคือ

เมื่อเวคเตอร์หนึ่งหน่วยที่ตั้งฉากกับระนาบอยู่ในแนวแกน Y

$$\left.\begin{aligned} u &= x \\ v &= z \end{aligned}\right\} \qquad \text{(b3)}$$

ภาพที่ b.1.3 แสดงภาพที่ได้จากการฉายภาพบนระนาบ

ภาพ b.1



แสดงการฉายภาพบนพื้นผิวของระนาบ

และเมื่อเวคเตอร์หนึ่งหน่วยที่ตั้งฉากกับระนาบอยู่ในแนวแกน Z

$$\left.\begin{aligned} u &= x \\ v &= y \end{aligned}\right\} \qquad (b4)$$

และเมื่อเวคเตอร์หนึ่งหน่วยที่ตั้งฉากกับระนาบอยู่ในแนวแกน X

$$\left.\begin{aligned} u &= z \\ v &= y \end{aligned}\right\} \qquad (b5)$$

ความสัมพันธ์ของภาพและทรงกลม

วัตถุทรงกลมจะมีลักษณะที่แตกต่างจากระนาบคือ ลักษณะของพื้นผิวของทรงกลมจะถูกกำหนดโดยตัวแปร 3 ตัวคือ x, y และ z หรือสามารถกำหนดในรูปของความสัมพันธ์ระหว่างรัศมีของทรงกลมและมุมต่างๆรอบจุดศูนย์กลางโดยที่

$$\left.\begin{aligned} P &= (x, y, z) \\ P &= (r\cos\theta\ \sin\phi,\ r\cos\phi\ ,\ r\sin\theta\ \sin\phi\ ) \end{aligned}\right\} \qquad (b6)$$

เมื่อ P คือตำแหน่งใดๆบนพื้นผิวของทรงกลม

r คือรัศมีของทรงกลม

θ คือมุมรอบแกน Y

ϕ คือมุมรอบแกน X

พิจารณาจากทรงกลมขนาด 1 หน่วยในบริเวณที่ (x, y, z) มีค่ามากกว่าศูนย์ สมการของความสัมพันธ์ระหว่างตัวแปรในระบบพิกัดคือ

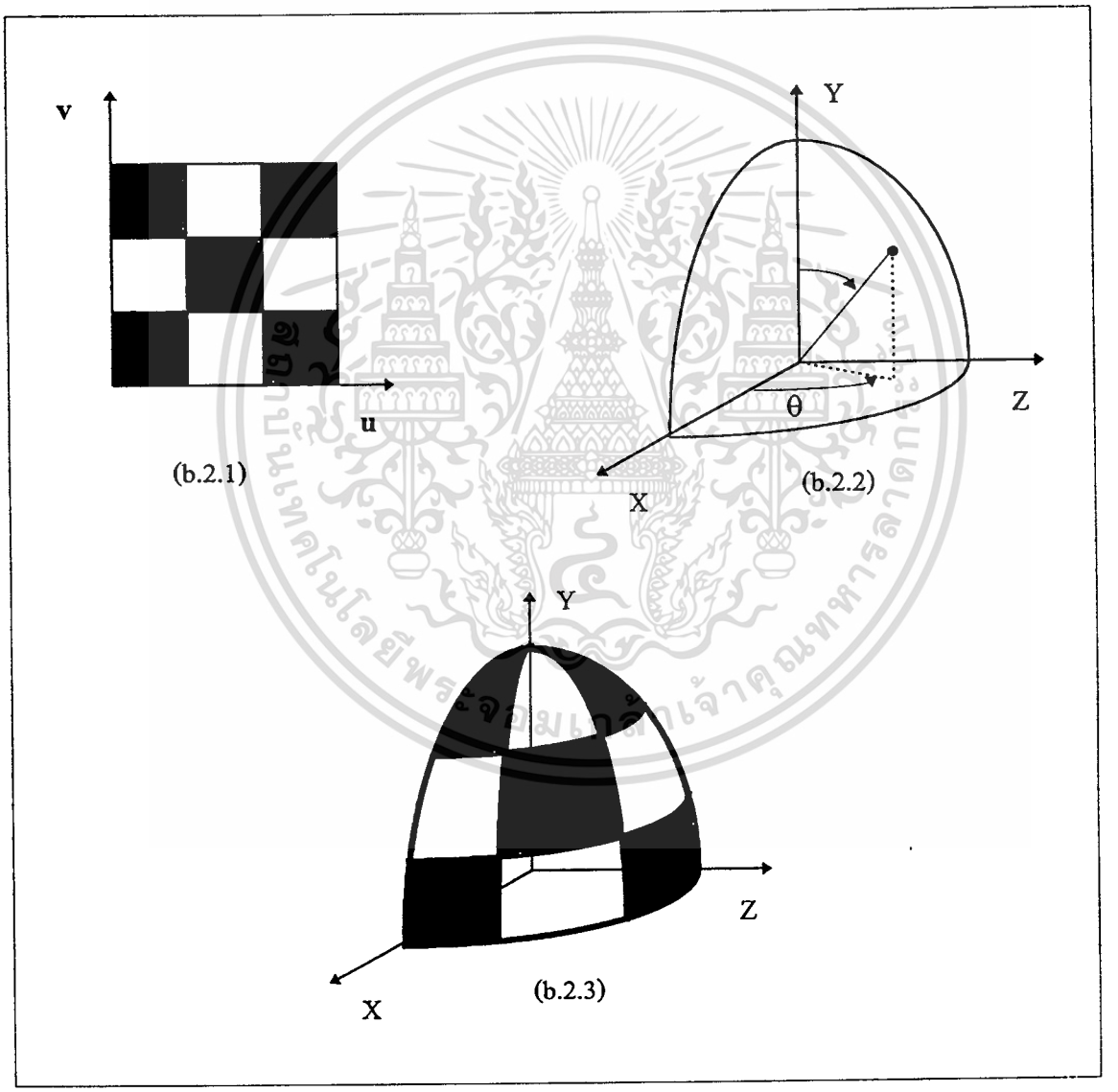
$$\left.\begin{aligned} x &= r\cos\theta\ \sin\phi \\ y &= r\cos\phi \qquad ; 0 \le \theta \le \frac{\pi}{2} \\ z &= r\sin\theta\ \sin\phi \qquad ; \frac{\pi}{4} \le \phi \le \frac{\pi}{2} \end{aligned}\right\} \qquad (b7)$$

และ

$$\left.\begin{aligned} u &= \frac{\theta}{\pi/2} \\ v &= \frac{\pi/2 - \phi}{\pi/4} \end{aligned}\right\} \qquad \text{(b8)}$$

ภาพที่ b.2.1 และภาพที่ b.2.2 แสดงถึงลักษณะของภาพและพื้นผิวของทรงกลม ส่วนภาพที่ b.2.3 แสดงภาพที่ได้จากการฉายภาพ

ภาพ b.2



แสดง การฉายภาพบนพื้นผิวของทรงกลม

การฉายภาพบนพื้นผิวของวัตถุในระบบคอมพิวเตอร์กราฟฟิก

การฉายภาพที่ต้องการลงบนพื้นผิวของวัตถุนั้น[59] สามารถแยกตามลักษณะของภาพได้เป็น 2 ส่วนด้วยกันคือ ภาพที่ฉายมีค่าความเข้มของแสงเพียงค่าเดียว และภาพที่ฉายมีค่าความเข้มของแสงหลายค่า ภาพที่มีค่าความเข้มของแสงเพียงค่าเดียวนั้นจะมีลักษณะที่ทุกๆตำแหน่งในภาพนั้นมีค่าความเข้มของแสงเท่ากันทุกตำแหน่ง ลักษณะของภาพจะถูกกำหนดโดยใช้จุดมืดหรือจุดสว่างเท่านั้น ภาพที่ b.3 แสดงภาพที่มีค่าความเข้มของแสงในภาพเพียงค่าเดียว

ขั้นตอนการฉายภาพที่ต้องการลงบนพื้นผิวของวัตถุนั้นแสดงไว้ในภาพที่ b.5 และจากวิธีการสร้างภาพเหมือนจริงโดยวิธี Ray tracing นั้นและการนำผลของการฉายภาพมาใช้ ทำให้ภาพที่ได้มีลักษณะที่ดีขึ้นเนื่องจากสามารถที่จะกำหนดลักษณะของพื้นผิวของวัตถุได้ แต่การฉายภาพจะทำให้เวลาที่ใช้ในการสร้างภาพมีค่ามากขึ้นตามลักษณะของความซับซ้อนของภาพที่ฉาย ภาพที่ b.6 และภาพที่ b.7 แสดงการฉายภาพที่มีค่าความเข้มของแสงค่าเดียวบนพื้นผิวของวัตถุ ภาพที่ b.8 และภาพที่ b.9 เป็นการฉายภาพที่มีค่าความเข้มของแสงหลายค่าบนพื้นผิวของวัตถุตามลำดับ

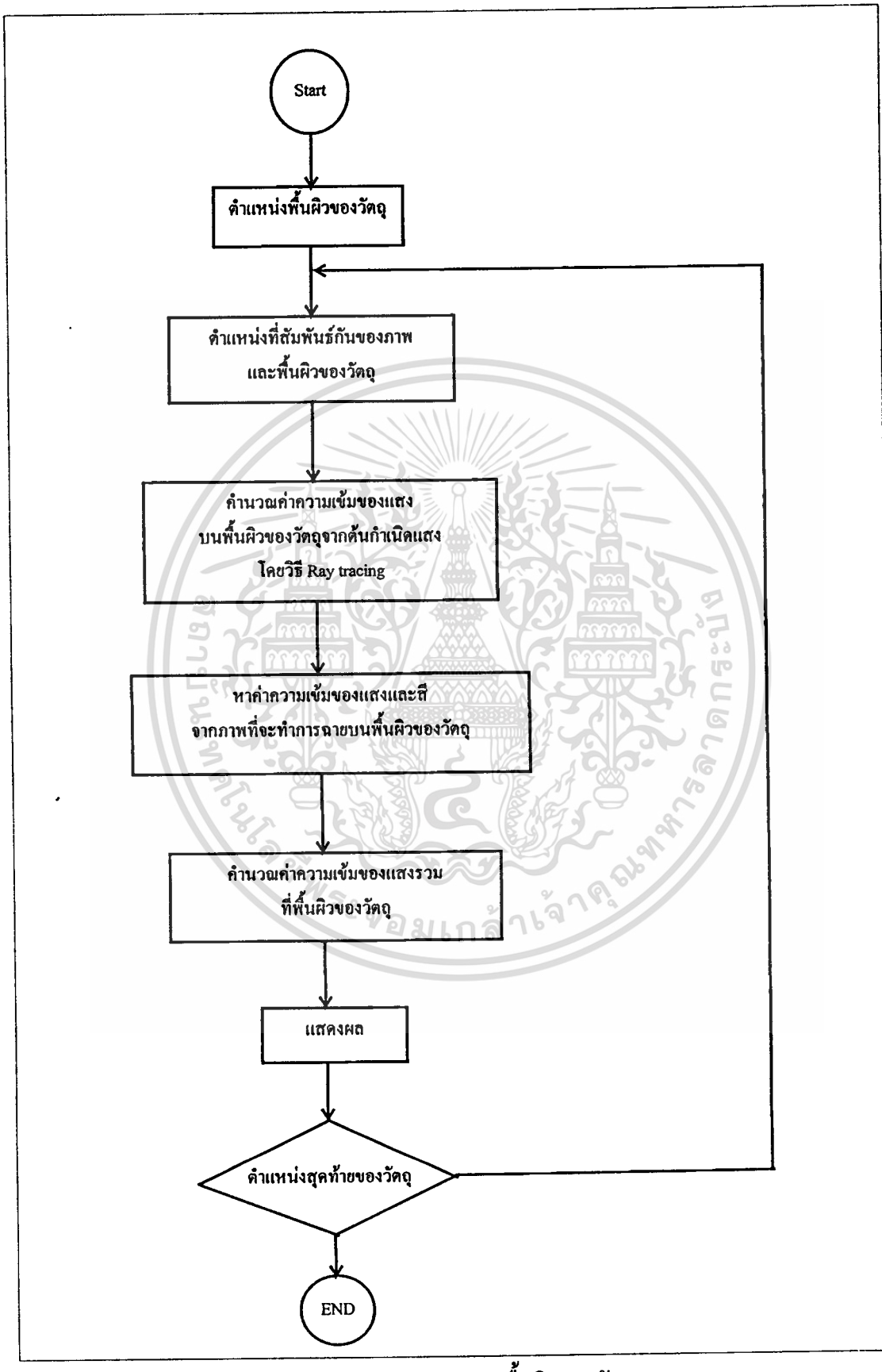
ภาพ b.3

แสดง ภาพที่มีค่าความเข้มของแสงเพียงค่าเดียว

ภาพ b.4

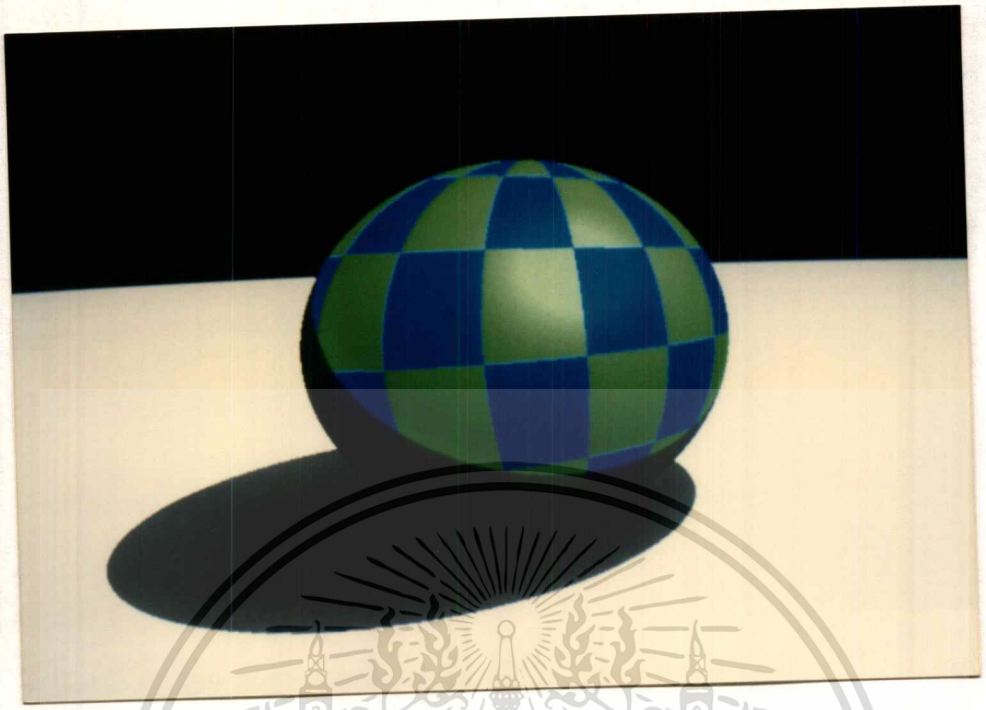
แสดง ภาพที่มีค่าความเข้มของแสงหลายค่า

ภาพ b.5



แสดง วิธีการฉายภาพบนพื้นผิวของวัตถุ

ภาพ b.6

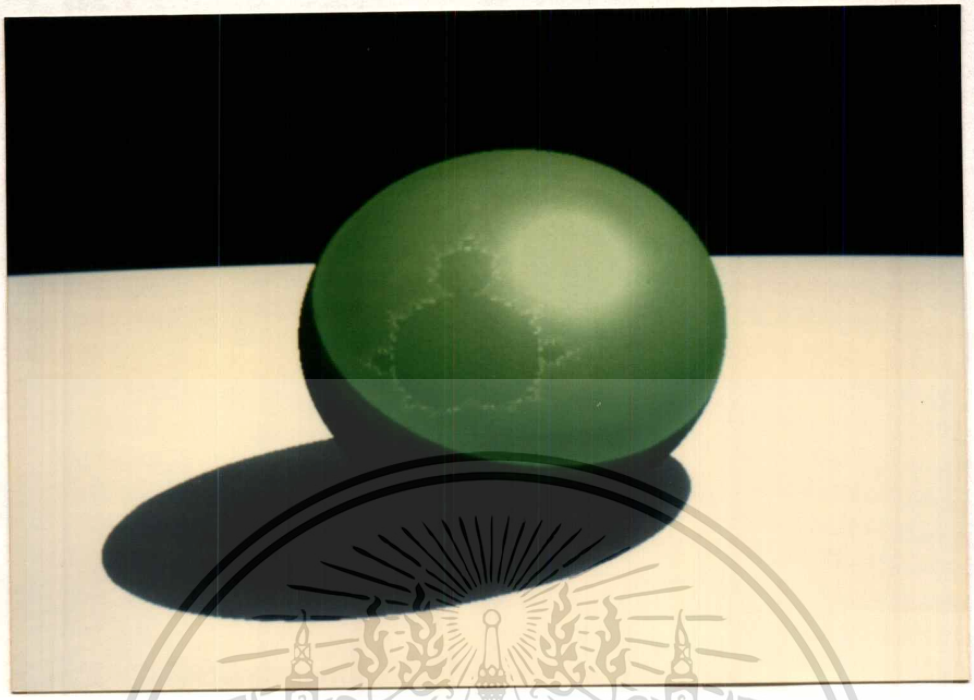


แสดง การฉายภาพที่มีค่าความเข้มของแสงค่าเดียว

ภาพ b.7

แสดง การฉายภาพที่มีค่าความเข้มของแสงค่าเดียว

ภาพ b.8



แสดง การฉายภาพที่มีค่าความเข้มของแสงหลายค่า

ภาพ b.9

แสดง การฉายภาพที่มีค่าความเข้มของแสงหลายค่า

ภาคผนวก ค

โปรแกรมต้นฉบับ

```
/*****************************************
RAY TRACING  by CHADIN KEOPLUNG
TEXTURE MAPPING
SOFT SHADOW - GAUSSIAN DISTRIBUTION
PROGRAM - Borland c++
                - Objected Oriented Program
                -Model small
*****************************************/
# include <dos.h>
# include <stdlib.h>
# include <stdio.h>
# include <conio.h>
# include <fcntl.h>
# include <io.h>
# include <alloc.h>
# include <string.h>
# include <time.h>
# include <GRAPHICS.h>      // โปรแกรมจัดระบบกราฟิก
# include <Vector.h>        // โปรแกรมจัดระบบตัวแปร 3 มิติ
# define MAX_X      1024
# define MAX_Y      768
# define JITTER     0.01    // jitter coefficient
# define DEGREE     45      // degree of point in light source
# define ZONE       3       // Zone in light source
# define SIGMA      40      //gaussian distribution
# define min(a,b)   (a<b) ? a : b
# define max(a,b)   (a>b) ? a : b
# define TRUE       1
# define ON         1
```

```
# define FALSE 0
# define OFF            0
# define PLANE          1
# define SPHERE         2
int HIT;
int SHADOW;
//****************************************
//***** data structure of light source*****
//****************************************
class Light {
        public :
                Vector location ; // location of light
                double intensity ; // intensity of light
                double radius ; // radius of point light source
                Light *NEXT; // address of next light source
                Light();
                Light(Vector location,double intensity,double radius);
                Vector Light::return_loc();
                double Light::return_int();
                double Light::return_rad(); };
Light::Light()  {
 location = Vector(0,0,0);
 intensity = MAX_INTENSITY-1;
 radius = 0;       }
Light::Light(Vector x1,double x2,double x3)      {
 location = x1;
 intensity = x2;
 radius = x3;      }
double Light::return_int()       {
 return(intensity);          }
Vector Light::return_loc()       {
 return(location);           }
```

```
double Light::return_rad()      {
return(radius);   }
//*****************************
//**data structure of objects**
//*****************************
class OBJECT {
        public :
                        int     TYPE ; // type of object
                        char    NAME[16] ; // name of object
                        Vector  location , // location of object
                                        vect1,
                                        vect2,
                                        vect3,
                                        norm ; // normal vector of object
                        double  Rd , //diffure reflection coefficient
                                        Rs , //specular reflection coefficient
                                        Ra ; //ambient light coefficient
                        int             f  ; //factor in specular reflection <1-200>
                        int     COLOR ; // color of object
                        int     TEXTURE ; //texture mapping ON-OFF
                        int     TEXTURE_TYPE ; //type of texture mapping
                        int     TEXTURE_COLOR ; //color of texture mapping
                        OBJECT  *NEXT ;  // address of next object
                        OBJECT operator=(OBJECT &);
                        virtual double FIND_TIME(Vector Start_ray,Vector Direct_ray);
                        virtual Vector FIND_POINT(Vector Start_ray,Vector
 Direct_ray,double hit_time);
                        virtual Vector FIND_NORMAL(Vector point);
                        virtual int FIND_TEXTURE(Vector point);
                };
```

```
OBJECT OBJECT::operator = (OBJECT & rvalue)      {
 TYPE = rvalue.TYPE;
 strcpy(NAME,rvalue.NAME);
 location = rvalue.location;
 vect1 = rvalue.vect1;
 vect2 = rvalue.vect2;
 vect3 = rvalue.vect3;
 norm = rvalue.norm;
 NEXT = rvalue.NEXT;
 return *this;      }
double OBJECT::FIND_TIME(Vector Start_ray,Vector Direct_ray)  {
          double hit_time= 0.0;
                    return(hit_time);  }
Vector OBJECT::FIND_POINT(Vector Start_ray,Vector Direct_ray,double hit_time)  {
                    return(Vector(0,0,0));  }
Vector OBJECT::FIND_NORMAL(Vector point)  {
                    return(Vector(0,0,0));  }
int OBJECT::FIND_TEXTURE(Vector point)    {
                    return(0);        }
//*******************************
//***DAta structure of WORLD***
//*******************************
typedef struct world{
          OBJECT *stack; // address of object
          int object_count; // number of object
          Light *next_light; // address of light
          int light_count ; // number of light
          Vector VRP;// View Refference Point;
          Vector VPN;// View PLANE Normal;
          Vector EYE_PT;//Eye point;
          double WINDOW_X_LOW;
```

```
        double WINDOW_Y_LOW;
        double WINDOW_X_HI;
        double WINDOW_Y_HI;
        double AMBIENT ; // ambient light intensity
        } World;
World WORLD;
//*************************
//**WORLD Initialization**
//*************************
void init_world(void)    {
 WORLD.stack = NULL;
 WORLD.object_count = 0;
 WORLD.WINDOW_X_LOW = -70;
 WORLD.WINDOW_X_HI = 70;
 WORLD.WINDOW_Y_LOW = -40;
 WORLD.WINDOW_Y_HI = 40;
 WORLD.AMBIENT = MAX_INTENSITY/2;     }
/*************************
**  TEXTURE MAPPING   **
*************************/
int map1(double x,double y) {
 double deltax,deltay;
 int WINX,WINY;
 WINX = WORLD.WINDOW_X_HI-WORLD.WINDOW_X_LOW;
 WINY = WORLD.WINDOW_Y_HI-WORLD.WINDOW_Y_LOW;
 deltax = .2;
 deltay = 0.5;
 if (((x>=0.0)&&(y>=0.0))||((x<0.0)&&(y<0.0)))   {
    if ((fabs(fmod(x/(deltax*WINX),2.0))<1.0)&&(fabs(fmod(y/(deltay*WINY),2.0))<1.0))
        return(0); //return object color
    else {
 if ((fabs(fmod(x/(deltax*WINX),2.0))>=1.0)&&(fabs(fmod(y/(deltay*WINY),2.0))>=1.0))
```

```
                return(0);
        else
                return(-1); // return texture color  } }
else {
 if ((fabs(fmod(x/(deltax*WINX),2.0))<1.0)&&(fabs(fmod(y/(deltay*WINY),2.0))<1.0))
         return(-1);
 else    {
 if ((fabs(fmod(x/(deltax*WINX),2.0))>=1.0)&&(fabs(fmod(y/(deltay*WINY),2.0))>=1.0))
        return(-1);
 else
        return(0); }}}
int map2(double x) {
 double deltax;
 int WINX;
 WINX = WORLD.WINDOW_X_HI-WORLD.WINDOW_X_LOW;
 deltax = .05;
  if (x>=0.0) {
         if (fabs(fmod(x/(deltax*WINX),2.0))<1.0)
                return(0);
         else
                return(-1); }
  else {
         if (fabs(fmod(x/(deltax*WINX),2.0))<1.0)
                return(-1);
         else
                return(0);  } }
 int map3(double x,double y)  {
  double deltax,deltay;
  int WINX,WINY;
  WINX = WORLD.WINDOW_X_HI-WORLD.WINDOW_X_LOW;
  WINY = WORLD.WINDOW_Y_HI-WORLD.WINDOW_Y_LOW;
  deltax = .1;
```

```
deltay = 0.1;
if (((x>=0.0)&&(y>=0.0))||((x<0.0)&&(y<0.0))) {
if ((fabs(fmod(x/(deltax*WINX),2.0))<1.0)&&(fabs(fmod(y/(deltay*WINY),2.0))<1.0))
        return(0);
else    {
if ((fabs(fmod(x/(deltax*WINX),2.0))>=1.0)&&(fabs(fmod(y/(deltay*WINY),2.0))>=1.0))
        return(0);
 else
        return(-1); } }
else {
 if ((fabs(fmod(x/(deltax*WINX),2.0))<1.0)&&(fabs(fmod(y/(deltay*WINY),2.0))<1.0))
        return(-1);
 else  {
 if ((fabs(fmod(x/(deltax*WINX),2.0))>=1.0)&&(fabs(fmod(y/(deltay*WINY),2.0))>=1.0))
        return(-1);
 else
        return(0);  } } }
# define MAP1_X 100
# define MAP1_Y 100
# define MAP2_X 200
# define MAP2_Y 200
# define OPEN_MAP2 "P_FILE.200" //แฟ้มข้อมูลภาพ
# define OPEN_MAP1 "P_FILE1.100" //แฟ้มข้อมูลภาพ
unsigned char map1p[MAP1_X][MAP1_Y];
unsigned char map2p[MAP2_X][MAP2_Y];
void open_map()  {
 unsigned char *dong;
 FILE *file1;
 int i,j;
 dong = new unsigned char[MAP2_Y];
 file1 = fopen(OPEN_MAP2,"rb");
 for (i=0;i<MAP2_X;i++)  {
```

```
            fread(dong,MAP2_Y,1,file1);
            for(j=0;j<MAP2_Y;j++)          {
            map2p[MAP2_Y-j-1][i] = dong[j];  } }
 fclose(file1);
 delete dong;  }
void open_map1() {
 unsigned char *dong;
 FILE *file1;
 int i,j;
 dong = new unsigned char[MAP1_Y];
 file1 = fopen(OPEN_MAP1,"rb");
 for (i=0;i<MAP1_X;i++) {
    fread(dong,MAP1_Y,1,file1);
    for(j=0;j<MAP1_Y;j++)  {
            map1p[j][i] = dong[j];  } }
 fclose(file1);
 delete dong; }
int map4(double x,double y)  {
 int dong1;
            dong1 = map2p[fmod(fabs(x),MAP1_X)][fmod(fabs(y),MAP1_Y)];
            return((dong1%MAX_COLOR)*(MAX_INTENSITY-1));  }
int map5(double x,double y)  {
 int dong1;
            dong1 = map1p[fmod(fabs(x),MAP1_X)][fmod(fabs(y),MAP1_Y)];
            return((dong1%MAX_INTENSITY)/4);  }
int map6(double x,double y)  {
 int dong1;
 double start_x,start_y;
 start_x = 50;
 start_y = 100;
 if (((fabs(x)>=start_x)&&(fabs(x)<=(start_x+MAP1_X)))&&((fabs(y)>=start_y)&&(fabs(y)
<=(start_y+MAP1_Y)))) {
```

```
        dong1 = map1p[fmod((fabs(x)-start_x),MAP1_X)][fmod((fabs(y)-start_y),MAP1_Y)];
        return((dong1%MAX_INTENSITY)/4);   }
else
        return (-1); //return object color   }
int map7(double x,double y)  {
 int dong1;
        dong1 = map2p[fmod(fabs(x),MAP2_X)][fmod(fabs(y),MAP2_Y)];
        return((dong1%MAX_INTENSITY)/4); }
//*****************************
//**data structure of plane**
//*****************************
class Plane : public OBJECT  {
                public :
                Plane() ;
                double FIND_TIME(Vector Start_ray,Vector Direct_ray);
                Vector FIND_POINT(Vector Start_ray,Vector Direct_ray,double hit_time);
                Vector FIND_NORMAL(Vector point);
                int FIND_TEXTURE(Vector point);   };
 Plane :: Plane()  {
 TYPE = PLANE;
 NEXT = NULL; }
 double Plane::FIND_TIME(Vector Start_ray,Vector Direct_ray)  {
 double hit_time;
 double vd,v0;
 vd = (location%Direct_ray);
 if (vd == 0.0)
        hit_time = 0.0;
 else  {
        v0 = -(vect1.return_x()+(location%Start_ray));
        hit_time = v0/vd;
        if (hit_time<1e-10)
                hit_time = -1.0; }
```

```
 return(hit_time);  }
Vector Plane::FIND_POINT(Vector Start_ray,Vector Direct_ray,double hit_time) {
 Vector hit_point;
 hit_point = Start_ray+(Direct_ray*hit_time);
 return(hit_point); }
Vector Plane::FIND_NORMAL(Vector point) {
 Vector normal;
 normal = location;
 return(normal);  }
int Plane::FIND_TEXTURE(Vector point)  {
 int texture_inten; //return intensity of texture mapping
  switch (TEXTURE_TYPE)       {
        case 1:{
                if (location.return_x() == TRUE)
                        texture_inten = map1(point.return_y(),point.return_z());
                else   {
                  if (location.return_y() == TRUE)
                        texture_inten = map1(point.return_x(),point.return_z());
                  else
                        texture_inten = map1(point.return_x(),point.return_y());  }
              break;   }
         case 2:  {
                 if (location.return_x() == TRUE)
                         texture_inten = map2(point.return_y());
                 else      {
                   if (location.return_y() == TRUE)
                         texture_inten = map2(point.return_x());
                   else
                         texture_inten = map2(point.return_x());  }
               break; }
         case 3:  {
                 if (location.return_x() == TRUE)
```

```
                texture_inten = map3(point.return_y(),point.return_z());
            else    {
              if (location.return_y() == TRUE)
                    texture_inten = map3(point.return_x(),point.return_z());
              else
                    texture_inten = map3(point.return_x(),point.return_y());   }
          break;   }
case 4:  {
          if (location.return_x() == TRUE)
                  texture_inten = map4(point.return_y(),point.return_z());
          else    {
            if (location.return_y() == TRUE)
                  texture_inten = map4(point.return_x(),point.return_z());
            else
                  texture_inten = map4(point.return_x(),point.return_y());  }
          break;   }
case 5:  {
          if (location.return_x() == TRUE)
                  texture_inten = map5(point.return_y(),point.return_z());
          else     {
                  if (location.return_y() == TRUE)
                  texture_inten = map5(point.return_x(),point.return_z());
                   else
                  texture_inten = map5(point.return_x(),point.return_y());  }
          break;   }
case 6:  {
          if (location.return_x() == TRUE)
                  texture_inten = map6(point.return_y(),point.return_z());
          else     {
                  if (location.return_y() == TRUE)
                  texture_inten = map6(point.return_x(),point.return_z());
                  else
```

```
                texture_inten = map6(point.return_x(),point.return_y()); }
            break; }
        case 7: {
            if (location.return_x() == TRUE)
                texture_inten = map7(point.return_y(),point.return_z());
            else    {
                if (location.return_y() == TRUE)
                texture_inten = map7(point.return_x(),point.return_z());
                else
                texture_inten = map7(point.return_x(),point.return_y());    }
            break; }
            default :       {
            if (location.return_x() == TRUE)
                texture_inten = map1(point.return_y(),point.return_z());
            else    {
              if (location.return_y() == TRUE)
                texture_inten = map1(point.return_x(),point.return_z());
                else
                texture_inten = map1(point.return_x(),point.return_y()); }
            break; }    }
return(texture_inten); }
//*****************************
//**data structure of sphere**
//*****************************
class Sphere : public OBJECT {
            public :
            Sphere();
            double FIND_TIME(Vector Start_ray,Vector Direct_ray);
            Vector FIND_POINT(Vector Start_ray,Vector Direct_ray,double hit_time);
            Vector FIND_NORMAL(Vector point);
            int FIND_TEXTURE(Vector point);    };
Sphere :: Sphere()      {
```

```
TYPE = SPHERE;
NAME[0] = NULL;
NEXT = NULL;   }
double Sphere::FIND_TIME(Vector Start_ray,Vector Direct_ray)  {
double hit_time;
 double AA,BB,CC,discr;
 Vector w;
 AA = Direct_ray%Direct_ray;
 w = Start_ray-location;
 BB = (Direct_ray%w)*2.0;
 CC = (w%w)-(vect1.return_x()*vect1.return_x());
  if (AA == 0.0)
          hit_time = 0.0;
  else  {
                discr = (BB*BB)-(4.0*CC);
                if (discr < 0.0)
                        hit_time = -1.0;
                 else
                        hit_time = (-BB -sqrt(discr))/2.0;  }
  return(hit_time);  }
 Vector Sphere::FIND_POINT(Vector Start_ray,Vector Direct_ray,double hit_time)  {
  Vector hit_point;
  hit_point = Start_ray+(Direct_ray*hit_time);
  return(hit_point);  }
 Vector Sphere::FIND_NORMAL(Vector point)  {
  Vector normal;
  normal = ~(point-location);
  return(normal);  }
 int Sphere::FIND_TEXTURE(Vector point)       {
 int texture_inten;
 Vector delta;
 double newx,newy;
```

```
switch (TEXTURE_TYPE)        {
                    case 1: {
                    delta  = point-location; //point - location of sphere
                    newx = atan2(delta.return_x(),delta.return_z())*vect1.return_x();
                    newy =
atan2(sqrt((delta.return_x()*delta.return_x())+(delta.return_z()*delta.return_z())),delta.return_y())
*vect1.return_x();
                    texture_inten = map1(newx,newy);
                    break;  }
                     case 2: {
                    delta  = point-location; //point - location of sphere
                    newx = atan2(delta.return_x(),delta.return_z())*vect1.return_x();
                    texture_inten = map2(newx);
                    break; }
      case 3:  {
                    delta  = point-location; //point - location of sphere
                    newx = atan2(delta.return_x(),delta.return_z())*vect1.return_x();
                    newy =
 atan2(sqrt((delta.return_x()*delta.return_x())+(delta.return_z()*delta.return_z())),delta.return_y())
 *vect1.return_x();
                    texture_inten = map1(newx,newy);
                    break; }
       case 4:  {
                    delta  = point-location; //point - location of sphere
                    newx = atan2(delta.return_x(),delta.return_z())*vect1.return_x();
                    newy =
  atan2(sqrt((delta.return_x()*delta.return_x())+(delta.return_z()*delta.return_z())),delta.return_y())
  *vect1.return_x();
                    texture_inten = map4(newx,newy);
                    break;  }
                     case 5:  {
                    delta  = point-location; //point - location of sphere
```

```
                newx = atan2(delta.return_x(),delta.return_z())*vect1.return_x();
                newy =
atan2(sqrt((delta.return_x()*delta.return_x())+(delta.return_z()*delta.return_z())),delta.return_y())
*vect1.return_x();
                texture_inten = map5(newx,newy);
                break; }
                 case 6: {
                delta  = point-location; //point - location of sphere
                newx = atan2(delta.return_x(),delta.return_z())*vect1.return_x();
                newy =
atan2(sqrt((delta.return_x()*delta.return_x())+(delta.return_z()*delta.return_z())),delta.return_y())
*vect1.return_x();
                texture_inten = map6(newx,newy);
                break; }
                 case 7: {
                delta  = point-location; //point - location of sphere
                newx = atan2(delta.return_x(),delta.return_z())*vect1.return_x();
                newy =
atan2(sqrt((delta.return_x()*delta.return_x())+(delta.return_z()*delta.return_z())),delta.return_y())
*vect1.return_x();
                texture_inten = map7(newx,newy);
                break; }
                 default: {
                delta  = point-location; //point - location of sphere
                newx = atan2(delta.return_x(),delta.return_z())*vect1.return_x();
                newy =
atan2(sqrt((delta.return_x()*delta.return_x())+(delta.return_z()*delta.return_z())),delta.return_y())
*vect1.return_x();
                texture_inten = map1(newx,newy);
                break; } }
 return(texture_inten);    }
 //************************************
```

```
//**find direction of reflection ray**
//************************************
Vector REFLECTION(Vector direction,Vector normal)          {
// direction = direction of ray
// normal = normal vector of object
Vector reflect;
                reflect = ~(direction-normal*(direction%normal)*2.0);
return(reflect);   }
//************************************
//**get data of sphere & put in WORLD**
//************************************
OBJECT *get_sphere(Vector c,double r,int color,double diffuse,double specular,double
spec_f,double ambient,int texture,int texture_type,int texture_color) {
 Sphere *newobj;
 cout<<endl<<"INPUT SPHERE : "<< c<<"RADIUS " << r;
 newobj = new Sphere;
 newobj->TYPE = SPHERE;
 newobj->location = c;
 newobj->vect1 = Vector(r,0,0);
 newobj->COLOR = color;
 newobj->Rd = diffuse;
 newobj->Rs = specular;
 newobj->f  = spec_f;
 newobj->Ra = ambient;
 newobj->TEXTURE = texture;
 newobj->TEXTURE_TYPE = texture_type;
 newobj->TEXTURE_COLOR = texture_color;
 newobj->NEXT = WORLD.stack;
 WORLD.object_count++;
 return (newobj);}
//************************************
//**get data of plane & put in WORLD**
```

```
//************************************
OBJECT *get_plane(Vector c,double dist,int color,double diffuse,double specular,double
spec_f,double ambient,int texture,int texture_type,int texture_color) {
 Plane *newobj;
 cout<<endl<<"INPUT PLANE : "<< c;
 newobj = new Plane;
 newobj->TYPE = PLANE;
 newobj->location = c;
 newobj->vect1 = Vector(dist,0,0);
 newobj->COLOR = color;
 newobj->Rd = diffuse;
 newobj->Rs = specular;
 newobj->f  = spec_f;
 newobj->Ra = ambient;
 newobj->TEXTURE = texture;
 newobj->TEXTURE_TYPE = texture_type;
 newobj->TEXTURE_COLOR = texture_color;
 newobj->NEXT = WORLD.stack;
 WORLD.object_count++;
 return (newobj);}
void display(OBJECT *last_object)      {
 while (last_object!=NULL)        {
 cout<<endl<<"TYPE"<<last_object->TYPE<<last_object->location<<"  " <<last_object->vect1;
 last_object = last_object->NEXT;          }        }
//*************************************************
//**change view plane coordinate to world coordinate**
//*************************************************
Vector VIEWtoWORLD(Vector mc1,Vector mc2,Vector mc3,Vector point)        {
Vector aa1;
double a1,a2,a3;
 a1 = (point.return_x()*mc1.return_x())+(point.return_y()*mc2.return_x())+(point.return_z()*
mc3.return_x());
```

```
a2 = (point.return_x()*mc1.return_y())+(point.return_y()*mc2.return_y())+(point.return_z()*
mc3.return_y());
 a3 = (point.return_x()*mc1.return_z())+(point.return_y()*mc2.return_z())+(point.return_z()*
mc3.return_z());
 aa1 = Vector(a1,a2,a3);
 return(aa1);      }
//*************************************
//**calculate diffuse light intensity**
//*************************************
unsigned int DIFFUSE_LIGHT(Light light,Vector point,Vector normal,double coef)        {
// return intensity 0-MAX_INTENSITY-1
// light-data of light source
// point-hitted point
// normal-normal vector of hitted point
// coefficient - diffuse coefficient
 Vector Us;
 double inten_d;
 Us = ~(light.return_loc()-point);
 inten_d = fabs(light.return_int()*coef*(Us%normal));
 return((int) inten_d);      }
//*************************************
//**calculate ambient light intensity**
//*************************************
unsigned int AMBIENT_LIGHT(double coef)    {
// return intensity 0-MAX_INTENSITY-1
// coefficient - ambient
 double inten_a;
 inten_a = fabs(WORLD.AMBIENT*coef);
 return((int) inten_a);      }
//*************************************
//**calculate specular light intersity**
//*************************************
```

```
unsigned int SPECULAR_LIGHT(Light light,Vector point,Vector normal,double coef,int
coef_f,Vector ray_c)      {
// light-data of light source
// point-hitted point
// normal-normal vector of hitted point
// coefficient - specular coefficient
// coef_f - factor of specular light
// ray_c -  direction of eye
 double inten_s;
 Vector Uv,Ur,Us;
 Vector R,S;
 S = light.return_loc()-point;
 R = (normal*((S%normal)*2.0))-S;
 Ur = ~R;
 Us = ~S;
 Uv = (-(ray_c));
 inten_s = fabs(light.return_int()*coef*pow((Ur%Uv),coef_f));
 return((int) inten_s);     }
int SELECT_SHADE(int amb,int diff,int spec,int map)    {
 int total_int;
 total_int = amb+diff+spec+map;
 if (total_int>MAX_INTENSITY-1)
          total_int = MAX_INTENSITY-1;
 return(total_int);          }
int SELECT_SHADOW(int amb,int diff,int spec,int soft)  {
 int total_int;
 total_int = amb+diff+spec+soft;
 if (total_int>MAX_INTENSITY-1)
          total_int = MAX_INTENSITY-1;
 return(total_int);          }
int read_pixel(int x,int y)          {
 union REGS r;
```

```
r.h.ah = 0x0d;
r.x.cx = x;
r.x.dx = y;
int86(0x10,&r,&r);
return(r.h.al);    }
unsigned int get_soft_shadow(Vector position,Light light,OBJECT *object1,OBJECT
*first_object)    {
int i;
int ray_num[ZONE+1]; //number of ray
int unblock[ZONE+1]; //number of unblock ray
double int_z[ZONE+1];// intensity of each zone
double int_z_sum;//sum of fraction of light intensity
Vector second_light;//secondary ray
double distance; //distacne from light source and point on surface
double theta,deltax,deltay,a,b,c,mu;
double circle_radius,circle_distance;
double hit_time;
unsigned int fraction;
unsigned int max_shadow;
double x,y,z;
int deg,zon;
Vector new_second ; // new vector of secondary ray
Vector jitter_dir;
 second_light = light.return_loc()-position;
 distance = sqrt(second_light%second_light);
 second_light = ~(second_light);
 unblock[0] = 1;
 ray_num[0] = 1;
 for(i=1;i<ZONE+1;i++)  {
          unblock[i] = floor(360/DEGREE);
          ray_num[i] = floor(360/DEGREE);          }
 // calculate circle of light source
```

```
theta  = asin(light.return_rad()/distance);
circle_radius = light.return_rad()*cos(theta);
circle_distance = distance - light.return_rad()*sin(theta);
// calculate jitter facter & direction of jittered hexagonal light source
deltax = JITTER*((rand()%100)/100.0);
deltay = JITTER*((rand()%100/100.0));
//calculate intensity of each zone
int_z_sum = 0;
for (zon=0;zon<ZONE+1;zon++)          {
        x = (circle_radius/ZONE)*zon;
        int_z[zon] = 1.0/(SIGMA*sqrt(2.0*PI))*exp(-1.0/2.0*(x/SIGMA)*(x/SIGMA));
        int_z_sum = int_z_sum+int_z[zon];       }
//CENTER OF LIGHT SOURCE
x= 0;
y= 0;
z = circle_distance;
new_second =Vector(x,y,z);
a = second_light.return_x();
b = second_light.return_y();
c = second_light.return_z();
mu = sqrt(b*b+c*c);
x = mu*new_second.return_x()+a*new_second.return_z();
y = -a*b*new_second.return_x()/mu+c*new_second.return_y()/mu+b*new_second.return_z();
z = -a*c*new_second.return_x()/mu-b*new_second.return_y()/mu+c*new_second.return_z();
jitter_dir = Vector(x,y,z);
//calculate fraction of unblocked light
max_shadow=AMBIENT_LIGHT(first_object->Ra);
SHADOW = FALSE;
hit_time= object1->FIND_TIME(position,~(jitter_dir));
if ((hit_time> -1e-10)&&(hit_time<1e10)  {
        SHADOW = TRUE;
    unblock[0]--;           }
```

```
if(ZONE>0)      {
 for(deg = 0;deg<360;deg=deg+DEGREE)            {
          for (zon = 0;zon<ZONE;zon++) {
                    x=
((circle_radius/ZONE)+(circle_radius/ZONE)*zon+(deltax*circle_radius))*(cos(PI/180.0*deg));
                    y=
((circle_radius/ZONE)+(circle_radius/ZONE)*zon+(deltay*circle_radius))*(sin(PI/180.0*deg));
                    z = circle_distance;
                    new_second =Vector(x,y,z);
                    a = second_light.return_x();
                    b = second_light.return_y();
                    c = second_light.return_z();
                    mu = sqrt(b*b+c*c);
                    x = mu*new_second.return_x()+a*new_second.return_z();
                    y = -
a*b*new_second.return_x()/mu+c*new_second.return_y()/mu+b*new_second.return_z();
                    z = -a*c*new_second.return_x()/mu-
b*new_second.return_y()/mu+c*new_second.return_z();
                    jitter_dir = Vector(x,y,z);
                    //calculate fraction of unblocked light
                    SHADOW = FALSE;
                    hit_time= object1->FIND_TIME(position,~(jitter_dir));
                    if ((hit_time> -1e-10)&&(hit_time<1e10))          {
                              SHADOW = TRUE;
                              unblock[zon+1]--;          } } } }
  fraction =0;
  for(i=0;i<ZONE+1;i++)  {
   fraction = fraction + floor((light.return_int()*(int_z[i]/int_z_sum))*unblock[i]/ray_num[i]);  }
  if (fraction>(max_shadow))
     fraction = max_shadow;
  return(fraction); }
  void main()      {
```

```
OBJECT *END_ADD; // end address of objects
OBJECT *object;
OBJECT *dummy; //first object hit by eye ray
OBJECT *dummy1; // first object hit by secondary ray
OBJECT *dummy2; // object hit by secondary ray for soft shadow
OBJECT *first_object;//first object hit by eye ray
int x,y;
double newx,newy;
Vector ray_s; // start point of eye ray
Vector ray_c; // direction of eye ray
Vector eye_world; //start point of eye in world coordinate
Vector start_world; // start point of ray in world coordinate
Vector direct_world; // direction of ray in world coordinate
Vector world_v;
double time_min; // minimum time that ray hit objects
double hit_time; // time that ray hit object
double hit_time1; // time that secondary ray hit object for soft shadow
double hit_light;// time that 2nd ray hit light source
Vector hit_point; // point that ray hit object
Vector first_point; // first point that ray hit object
Light light1;
time_t first,second;
unsigned int intensity1;
unsigned int intensity2;
unsigned int intensity3;
unsigned int intensity4;
unsigned int shadow_intensity;
unsigned int total_int;
unsigned int max_shadow; // maximum intensity of shadow
int color_intensity; // intensity of new object
unsigned int color_factor; // color of new object
Vector second_ray; // ray from hitted point to light source
```

```
Vector normal;
int i,j;
init_world();
Vector VtoW1,VtoW2,VtoW3;
END_ADD = new OBJECT;
object = new OBJECT;
first_object = new OBJECT;
dummy = new OBJECT;
dummy1 = new OBJECT;
dummy2 = new OBJECT;
// read data from map picture
open_map();
open_map1();
WORLD.stack = get_sphere(Vector(150,100,150),100,1,.8,.35,5,.5,ON,2,0);
WORLD.stack = get_sphere(Vector(150,260,150),60,2,.54,.45,4,.55,ON,2,1);
WORLD.stack = get_sphere(Vector(150,360,150),40,0,.34,.5,3,.65,OFF,3,1);
//*********HAND
WORLD.stack = get_sphere(Vector(75,270,150),30,0,.34,.6,3,.65,OFF,5,0);
WORLD.stack = get_sphere(Vector(225,270,150),30,0,.34,.6,3,.65,OFF,5,0);
//**********
WORLD.stack = get_sphere(Vector(150,290,95),10,0,.34,.6,3,.65,OFF,5,0);
WORLD.stack = get_sphere(Vector(150,260,85),10,0,.34,.6,3,.65,OFF,5,0);
WORLD.stack = get_sphere(Vector(150,230,95),10,0,.34,.6,3,.65,OFF,5,0);
//*********EYE
WORLD.stack = get_sphere(Vector(150,360,102),8,1,.34,.6,3,.65,OFF,5,0);
WORLD.stack = get_sphere(Vector(130,370,110),4,3,.34,.6,3,.65,OFF,5,0);
WORLD.stack = get_sphere(Vector(170,370,110),4,3,.34,.6,3,.65,OFF,5,0);
//**********LIPS
WORLD.stack = get_sphere(Vector(130,350,110),4,1,.34,.6,3,.65,OFF,5,0);
WORLD.stack = get_sphere(Vector(140,342,110),4,1,.34,.6,3,.65,OFF,5,0);
WORLD.stack = get_sphere(Vector(150,340,110),4,1,.34,.6,3,.65,OFF,5,0);
WORLD.stack = get_sphere(Vector(160,342,110),4,1,.34,.6,3,.65,OFF,5,0);
```

```
WORLD.stack = get_sphere(Vector(170,350,110),4,1,.34,.6,3,.65,OFF,5,0);
//**********
WORLD.stack = get_cylinder(Vector(600,0,700),500,50,3,.85,.45,2,.4,OFF,1,3);
//**********
WORLD.stack = get_cone(Vector(400,0,500),400,100,2,.85,.45,2,.4,OFF,1,3);
//**********
WORLD.stack = get_plane(Vector(0.0,1.0,0.0),0,0,.85,.45,2,.4,ON,1,3);
WORLD.VRP = Vector(180,120,-645);
WORLD.VPN = ~(-WORLD.VRP);
WORLD.EYE_PT = Vector(0,0,-50);
END_ADD = WORLD.stack;
light1= Light(Vector(-500,800,-900),MAX_INTENSITY-1,200);
display(WORLD.stack);
getch();
gscreen();//1024x768
SET_VGA_PALETTE();
world_v = Vector(0,-1,0);
VtoW3 = WORLD.VPN;// n
VtoW2 = ~(world_v-(VtoW3*(world_v%VtoW3)));// v
VtoW1 = (VtoW3^VtoW2); //u
first = time(NULL);
for (x = 0;x<MAX_X;x++)        {
        for (y=0;y<MAX_Y;y++)        {
                // calculate coordinate of picture plane (screen)
                HIT = FALSE;
                newx = ((WORLD.WINDOW_X_LOW)+(x*(WORLD.WINDOW_X_HI-
WORLD.WINDOW_X_LOW)/MAX_X));
                newy = ((WORLD.WINDOW_Y_LOW)+(y*(WORLD.WINDOW_Y_HI-
WORLD.WINDOW_Y_LOW)/MAX_Y));
                WORLD.stack = END_ADD;
                // change to world coordinate
  eye_world = VIEWtoWORLD(VtoW1,VtoW2,VtoW3,WORLD.EYE_PT)+WORLD.VRP;
```

```
start_world = VIEWtoWORLD(VtoW1,VtoW2,VtoW3,Vector(newx,newy,0))+WORLD.VRP;
                direct_world = start_world-eye_world;
                ray_s = eye_world;
                ray_c = ~(direct_world);
                time_min = 1000000000;
                // check that eye ray hit object (you can see that object
                for (i = 0;i<WORLD.object_count;i++) {
                        object = WORLD.stack;
                        hit_time= object->FIND_TIME(ray_s,ray_c);
                        if ((hit_time > -1e-10)&&(hit_time<time_min)) {
                                time_min = hit_time;
                                hit_point=object->FIND_POINT(ray_s,ray_c,hit_time);
                                first_point = hit_point;
                                dummy = WORLD.stack;
                                first_object = WORLD.stack;
                                HIT = TRUE; }
                        WORLD.stack = object->NEXT; }
                        //dummy = address of first object
                        //first point = first hit point
                                // if eye ray hit any object
                if (HIT==TRUE) {
                        //find shadow ray from first hit point
                        ray_s = first_point;
                        normal=(~(dummy->FIND_NORMAL(ray_s)));
                        second_ray = ~(light1.return_loc()-ray_s);
        hit_light = dummy->FIND_TIME(light1.return_loc(),Vector(0.0,0.0,0.0)-second_ray);
                        WORLD.stack = END_ADD;
                        SHADOW = FALSE;
                        // check that point is in shadow ?
                        for (i = 0;i<WORLD.object_count;i++)  {
                        object = WORLD.stack;
                        hit_time= object->FIND_TIME(ray_s,second_ray);
```

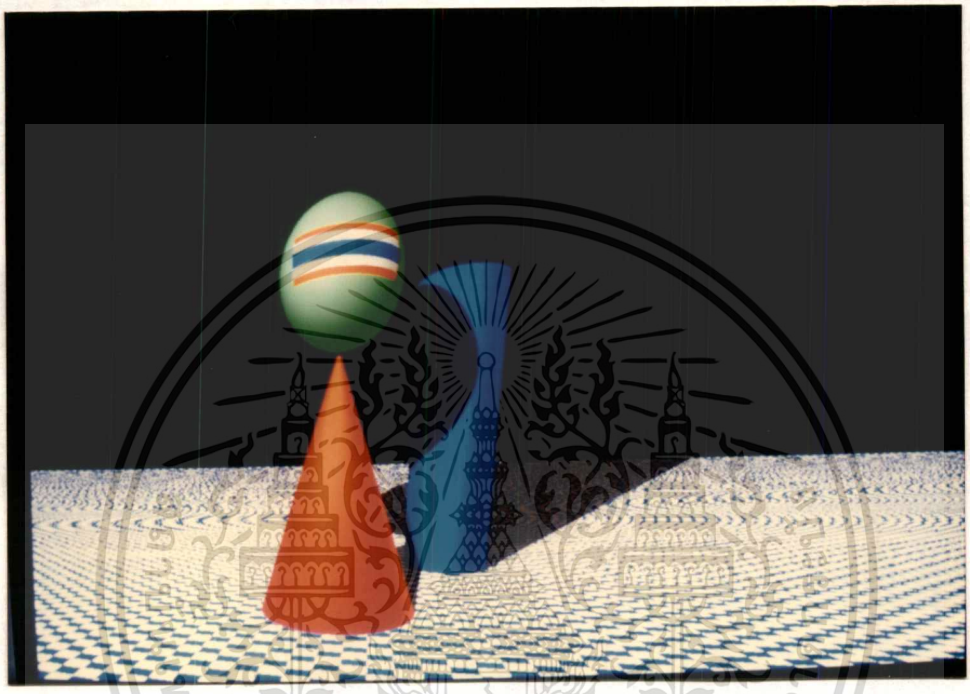
```
                if ((hit_time>=-1e-10)&&(hit_time<=1e10)&&(hit_time<=hit_light {
                        SHADOW = TRUE;
                        dummy1 = WORLD.stack;
                        break; }
                WORLD.stack = object->NEXT; }
                if (SHADOW !=TRUE)  {
                // if the point isn't in shadow
                if (first_object->TEXTURE == OFF)       {
                color_factor = first_object->COLOR*MAX_INTENSITY;
                color_intensity = -1;       }
                else      {
                color_intensity = first_object->FIND_TEXTURE(first_point);
                if (color_intensity == -1)
        color_factor = first_object->COLOR*MAX_INTENSITY;
                else
        color_factor = first_object->TEXTURE_COLOR*MAX_INTENSITY;       }
intensity1=AMBIENT_LIGHT(dummy->Ra);
intensity2=DIFFUSE_LIGHT(light1,ray_s,normal,dummy->Rd);
intensity3=SPECULAR_LIGHT(light1,ray_s,normal,dummy->Rs,dummy->f,ray_c);
total_int = SELECT_SHADE(intensity1,intensity2,intensity3,color_intensity);         }
else      {
// shadow from itsalf
if (first_object == dummy1)       {
if (first_object->TEXTURE == OFF)       {
color_factor = first_object->COLOR*MAX_INTENSITY;
color_intensity = -1;       }
else      {
color_intensity = first_object->FIND_TEXTURE(first_point);
if (color_intensity == -1)
color_factor = first_object->COLOR*MAX_INTENSITY;
else
color_factor = first_object->TEXTURE_COLOR*MAX_INTENSITY;       }
```

```
        max_shadow=AMBIENT_LIGHT(first_object->Ra);
        intensity1 = 0;
        intensity2=0;
        intensity3 = 0;
        WORLD.stack = END_ADD;
        // check point is in shadow by another objects?
          intensity4 =1e10;
          j=0;
          for (i = 0;i<WORLD.object_count;i++)  {
                  object = WORLD.stack;
                  hit_time1= object->FIND_TIME(ray_s,second_ray);
                  if ((hit_time1>= -1e-10)&&(hit_time1<=1e10))      {
                  dummy2 = WORLD.stack;
                  shadow_intensity = get_soft_shadow(ray_s,light1,dummy2,first_object);
                  if (shadow_intensity<intensity4)
                  intensity4 = shadow_intensity;      }
                  WORLD.stack = object->NEXT;   }
        total_int
=SELECT_SHADOW1(max_shadow,intensity1,intensity2,intensity3,(max_shadow-intensity4));
}
        else      {
        if (first_object->TEXTURE == OFF)      {
        color_factor = first_object->COLOR*MAX_INTENSITY;
        color_intensity = -1;       }
        else      {
        color_intensity = first_object->FIND_TEXTURE(first_point);
        if (color_intensity == -1)
                color_factor = first_object->COLOR*MAX_INTENSITY;
        else
                color_factor = first_object->TEXTURE_COLOR*MAX_INTENSITY;  }
        max_shadow=AMBIENT_LIGHT(first_object->Ra);
        intensity1=intensity2=intensity3=0;
```

```
        WORLD.stack = END_ADD;
         // check that point is in shadow ?
        intensity4 =1e10;
        j=0;
        for (i = 0;i<WORLD.object_count;i++)    {
        object = WORLD.stack;
        hit_time1= object->FIND_TIME(ray_s,second_ray);
        if ((hit_time1>= -1e-10)&&(hit_time1<=1e10))    {
        dummy2 = WORLD.stack;
        shadow_intensity = get_soft_shadow(ray_s,light1,dummy2,first_object);
        if(shadow_intensity<intensity4)
           intensity4 = shadow_intensity;  }
        WORLD.stack = object->NEXT;          }
        total_int
=SELECT_SHADOW1(max_shadow,intensity1,intensity2,intensity3,(max_shadow-intensity4));
        }  }
        WRITE_DOT(x,y,color_factor+total_int);  }   }      }
second = time(NULL);
getch();
tscreen();
delete END_ADD,dummy,object,dummy1,dummy2,first_object;
printf("Generate PICTURE TIME : %f seconds.\n",difftime(second,first));
printf("Generate PICTURE TIME : %f min.\n",(difftime(second,first)/60.0));
}
```

ภาคผนวก ง
การสร้างภาพโดยวิธี Ray Tracing

ภาพที่ d1



แสดง ภาพจำลองโดยวิธี Ray Tracing

ในการสร้างภาพจำลองแบบ Ray Tracing นั้นสิ่งแรกที่ต้องกำหนดคือคุณสมบัติของสีและลำดับค่าความเข้มของแสงในแต่ละสี สามารถกำหนดโดยใช้แบบจำลอง RGB เพื่อกำหนดระดับค่าความเข้มของแสง ในตัวอย่างนี้ กำหนดให้มีจำนวน 4 สีและในแต่ละสีจะมีค่าความเข้มของแสง 64 ระดับ

ลักษณะของวัตถุที่ใช้ในแบบจะลองจะถูกกำหนดโดย

ต้นกำเนิดแสง

-ตำแหน่ง = (600, 600, 20)

-ค่าความเข้มของแสง = 63

แสงที่กระจายโดยรอบ

-ค่าความเข้มของแสง = 32

ผู้สังเกต

-ตำแหน่ง = (900, 220, 200)

ระนาบ

-ตำแหน่ง Y = 0

-สัมประสิทธิ์การสะท้อนแสง = 0.4

-สัมประสิทธิ์การสะท้อนแสงแบบกรวย = 0.45

ทรงกลม

-ตำแหน่ง = (150, 500, 150)

-รัศมี = 100

-สัมประสิทธิ์การสะท้อนแสง = 0.5

-สัมประสิทธิ์การสะท้อนแสงแบบกรวย = 0.35

ทรงกรวย

-ตำแหน่ง = (150, 0, 150)

-รัศมี = 100 ความสูง = 400

-สัมประสิทธิ์การสะท้อนแสง = 0.4

-สัมประสิทธิ์การสะท้อนแสงแบบกรวย = 0.45

ทรงกระบอก

-ตำแหน่ง = (-250, 0, 400)

-รัศมี = 100 ความสูง = 600

-สัมประสิทธิ์การสะท้อนแสง = 0.4

-สัมประสิทธิ์การสะท้อนแสงแบบกรวย = 0.45

การสร้างภาพจำลองจะเริ่มจากการสร้างเวกเตอร์จากผู้สังเกตไปสู่ตำแหน่งเริ่มต้นของจอภาพ (0, 0) ที่แปลงให้เป็นพิกัดของระบบใน 3 มิติ เรียกว่าเวกเตอร์การมอง ใช้ทดสอบว่าเวกเตอร์การมองตกกระทบกับวัตถุหรือไม่เพื่อหาวัตถุแรกที่ตกกระทบในระบบ โดยการตรวจสอบเวลาที่น้อยที่สุดที่เวกเตอร์การมองตกกระทบกับวัตถุแต่ละชนิดในระบบ ที่ตำแหน่งตกกระทบสร้างเวกเตอร์ลำดับที่สองจากตำแหน่งตกกระทบไปสู่ตำแหน่งต้นกำเนิดแสงเพื่อหาค่าความเข้มของแสงที่ตำแหน่งตกกระทบ โดยการตรวจสอบว่าแสงลำดับที่สองตกกระทบกับวัตถุอื่นในระบบหรือไม่ ถ้ามีแสดงว่าที่ตำแหน่งตกกระทบนั้นจะเป็นบริเวณที่เป็นส่วนของเงาและค่าความเข้มของแสงที่ตำแหน่งนี้คือค่าความเข้มของแสงจากแสงที่กระจายอยู่โดยรอบในสมการที่ (29) ในกรณีที่แสงลำดับที่สองไม่มีการตกกระทบกับวัตถุอื่นในระบบ ค่าความเข้มของแสงที่ตำแหน่งตกกระทบจะเป็นค่าความเข้มของแสงที่ได้จากแสงที่กระจายอยู่โดยรอบและแสงจากต้นกำเนิดแสง สมการที่(36)

เมื่อได้ค่าความเข้มของแสงที่ตำแหน่งตกกระทบจะสามารถสร้างภาพจำลองได้ สำหรับที่ตำแหน่งอื่นของจอภาพจะสามารถทำได้โดยวิธีเดียวกัน

ภาคผนวก จ
บทความวิจัยที่ได้รับการตีพิมพ์

1. กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, เศรษพล ลิ้มปราชญา, **"การประยุกต์การแสดงผลภาษาไทยกับกราฟิกมอดูล"**, เอกสารการประชุมวิชาการ วิศวกรรมไฟฟ้า ประจำปี 2534, หน้า 89-97.

2. กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"The Beauty of Fractal Images"**, บิซิเนส คอมพิวเตอร์ แมกะซีน, ปีที่ 3, ฉบับที่ 35, มกราคม, 2535, หน้า 125-129.

3. กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"คอมพิวเตอร์กราฟิก 2 มิติ"**,บิซิเนส คอมพิวเตอร์ แมกะซีน, ปีที่ 4, ฉบับที่ 41, กรกฎาคม 2535, หน้า 195-202.

4. กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"คอมพิวเตอร์กราฟิก 3 มิติ"**, บิซิเนส คอมพิวเตอร์ แมกะซีน, ปีที่ 5, ฉบับที่ 50, เมษายน 2536, หน้า 263-267.

5. กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"แสงและการจัดลำดับความเข้ม"**, วารสารสมาคมคอมพิวเตอร์แห่งประเทศไทย ในพระบรมราชูปถัมภ์, ปีที่ 20, ฉบับที่ 102, กรกฎาคม-สิงหาคม, 2536, หน้า 27-34.

6. กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"Light and Color Model แสงและแบบจำลองของสี"**, วารสารสมาคมคอมพิวเตอร์แห่งประเทศไทย ในพระบรมราชูปถัมภ์, ปีที่ 20, ฉบับที่ 105, มกราคม-กุมภาพันธ์, 2537, หน้า 58-66.

7. กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"Objects' Illumination Model in Computer Graphics by Ray Tracing, แบบจำลองการส่องสว่างของวัตถุในคอมพิวเตอร์กราฟิกโดย Ray Tracing"**, วารสารสมาคมคอมพิวเตอร์แห่งประเทศไทย ในพระบรมราชูปถัมภ์, ปีที่ 20, ฉบับที่ 109, กันยายน-ตุลาคม, 2537, หน้า 57-64.

8. กิตติ ไพฑูรย์วัฒนกิจ, ชฎิล แก้วปลั่ง, **"ภาพและรายละเอียดพื้นผิวของวัตถุ"** , วารสารสมาคมคอมพิวเตอร์แห่งประเทศไทย ในพระบรมราชูปถัมภ์, ปีที่ 21, ฉบับที่ 113, พฤษภาคม-มิถุนายน, 2538, หน้า 29-36.

ประวัติผู้เขียน

ชื่อผู้เขียน	นายชฎิล แก้วปลั่ง
วัน เดือน ปีเกิด	วันที่ 4 เมษายน พ.ศ.2508
วุฒิการศึกษาระดับปริญญาตรี	วิทยาศาสตร์บัณฑิต สาขาฟิสิกส์
สถานที่สำเร็จการศึกษา	มหาวิทยาลัยเชียงใหม่
ปีที่สำเร็จการศึกษา	2530
ประสบการณ์การทำงาน	นักเขียนโปรแกรม
	โปรแกรมเมอร์
	นักวิจัยและพัฒนา
อาชีพปัจจุบัน	นักเขียนโปรแกรม